#2

BOLT BERANEK AND NEWMAN INC

CONSULTING · DEVELOPMENT · RESEARCH

PROPOSAL: INTERFACE MESSAGE PROCESSORS FOR THE ARPA COMPUTER NETWORK

For Dave.

CAMBRIDGE NEW YORK CHICAGO LOS ANGELES

PROPOSAL: INTERFACE MESSAGE PROCESSORS FOR THE ARPA COMPUTER NETWORK

RFQ No. DAHC15 69 Q 0002

BBN Proposal No. IMP P69-IST-5

"This data furnished in response to RFQ No. DAHC15 69 Q 0002 shall not be disclosed outside the Government or be duplicated, used or disclosed in whole or in part for any purpose other than to evaluate the quotation; provided, that if a contract is awarded to this quoter as a result of or in connection with the submission of such data, the Government will have the right to duplicate, use, or disclose this data to the extent provided in the contract. This restriction does not limit the Government's right to use information contained in such data if it is obtained from another source."

6 September 1968

Submitted to:

Department of the Army
Defense Supply Service-Washington
The Pentagon, Room 1D 245
Washington, D.C. 20310

BOLT BERANEK AND NEWMAN INC.
50 Moulton Street
Cambridge, Massachusetts 02138

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF PLATES

CHAPTER I: INTRODUCTION

Bolt Beranek and Newman Inc. (BBN) is pleased to submit a design for the IMP subnet and a proposal for the implementation of the initial system. We feel that our prior experience, the degree of our interest, and our ability to assign a singularly strong technical team provide an excellent basis for creditable performance on this system job. Appendix A discusses applicable BBN prior experience, briefly describes BBN facilities, and provides a recent BBN Annual Report.

We have studied the technical problems of the IMP subnet extensively and we present in this proposal a detailed design. This design is substantially in consonance with the requirements given in the RFQ; in some cases, we suggest variants and explain the basis for our views.

BBN has obtained the interested cooperation of the Computer Control Division of Honeywell for the provision of hardware and technical assistance on a subcontract basis. Honeywell will provide DDP-516 computers, specialized interface hardware, maintenance, systems engineering assistance, and field engineering assistance. Appendix B includes a Honeywell subcontract proposal and letter of committment, mentions the Honeywell personnel to be involved in technical assistance, provides a brochure about the DDP-516, and provides a Honeywell summary of the special equipment.

In addition, and separate from the five copies of our proposal, a single copy of several other descriptive Honeywell documents concerning the DDP-516 are provided as supporting material. Although some aspects of the computer choice are discussed in the body of the proposal, Appendix D provides a more detailed discussion of the Honeywell/DDP-516 choice.

The body of this proposal is divided into four parts:

CHAPTER II:	TECHNICAL OVERVIEW, where we discuss broad system issues and provide a basis for our design choices.
CHAPTER III:	PROPOSED SYSTEM DESIGN, where we describe the IMP subnet as we propose to implement it. Supporting this section are Appendix C on timing computations and the RFQ model, Appendix E on details of the hardware interfaces, Appendix F on software details, and Appendix H on network simulation.
CHAPTER IV:	IMPLEMENTATION PLAN, where we describe our schedule, work scope, and plans for field installation and testing. More detail on testing is included in Appendix G.
CHAPTER V:	TECHNICAL TEAM, where we discuss the BBN people who would participate in this effort and the allocation of responsibilities. Supporting this section is Appendix J which provides résumés of these people.

A cost proposal is provided as a separate document, which also discusses BBN conformance with the various legal requirements of the RFQ.

We are extremely interested in this project and would exert our very best efforts toward making the net a useful reality.

CHAPTER II: TECHNICAL OVERVIEW

The ARPA network is an experimental system. Its purposes are, first of all, to identify and find a solution to the problems associated with coupling many large time-shared computers together. A second major goal of the network is to provide a vehicle for investigating how a network is used by the community of users that it services. Finally, the network should provide the basis for developing communications, documentation, and programming techniques that will enhance the interchange among users, thereby increasing their productivity.

Because of its experimental nature, the ARPA network must be viewed as a growing and evolutionary system. The first two stages of its development are discussed in the RFQ: (1) a 4-node initial net followed by expansion to (2) a 19-node net. Parallel with these two stages, an experimental program is implicit. During evolution of the net, observation will be made and experiments will be performed to determine how the net is used, and to discover ways of increasing its capacity and in improving the facility with which it may be used. The initial design will, hopefully, prove sufficiently robust so that it may be used for an extended period, be expanded to include other nodes, or serve as a prototype for other similar nets. On the other hand, a major redesign may be indicated and a second generation network system developed.

Our proposed design should make this evolution and growth graceful. By designing with a sufficient factor of safety, the hardware configuration of the IMPs should have sufficient expansion capability so that major changes to the hardware will not be required. By allowing sufficient unused processor time and core memory, it should be possible to expand the functions that are performed by the software as new requirements are encountered.

** *We take the position that it will be difficult to make the system work.* As a consequence we have devoted considerable attention to techniques for simplification, for improving reliability, and for testing the state and performance of system elements for correcting or recovering from failures of many different kinds. Our proposal stresses an initial implementation, which, wherever possible, provides a simple, reliable, uncomplicated approach. We feel that complexities and elaborations may be added later, once the net has attained a sufficiently reliable and useful state. **

Our design for the network, discussed in detail in Chapter III, is a result of careful consideration of a number of important technical and operational issues. In the remainder of this Chapter, we discuss these issues and show how our design provides a reasonable treatment of these issues. First, we present our conception of the network task allocation. Then we describe the communications problems associated with the net. Operational, reliability, and implementation problems are considered next and then experimental procedures within the network.

A. Task Allocation in the ARPA Network

This network is envisioned as an interconnected communication facility that will allow researchers at ARPA-supported facilities to utilize capabilities available at other ARPA sites. The network will provide a link between user(s) programs at one site, and programs and data at remote sites. A typical use might involve a question-answering program at BBN working on extracts from a data base available at SDC. [Because of serious timing problems, it is not anticipated that an MIT program will be able to control the SRI robot in real-time.] was this a requirement?

All the ARPA contractors are independent research establishments, many with vested interests in the development of their own time-shared systems. To simplify the problem of communication between nodes of the network, each site is to be provided with a small computer, an Interface Message Processor (IMP). *The ARPA network could have been constructed without any IMPs.* That is to say, each Host could have been forced to deal with line disciplines and errors entirely without an interface machine. The decision to include IMPs, and to produce a subnet, implies a strong desire to save each Host some of this time and trouble and to concentrate it in one standardized place, namely the IMP. This basic decision produces an incentive for trying to do, within the IMP, all those computational and logical tasks that can be reasonably done to save the Hosts difficulty. (While some IMP tasks are obvious, such as line discipline or store and forward, there are other tasks that lie in a gray area: how shall the diverse representations of binary data be reformatted? Or where should word length repacking occur?) Unfortunately, placing all the network burdens on the IMPs leads to IMPs which grow without limit and to a network that is complex, difficult to implement, and probably unreliable.

There are a number of crucial questions of Host-IMP effort apportionment where a flexible *laissez faire* approach will probably compromise early realization of reliable network operation. These questions include:

1) Shall the IMP be dependent, for its normal operations, on the status of the Host:

   a) By having IMP programs loaded from, started by or stopped by the Host?

   b) By having the Host provide buffer storage, backup storage, or program storage for the IMPs?

2) Shall real-time modification of network functions or procedures be permitted:

   a) By allowing a local or remote Host to modify IMP programs in real-time?

   b) By allowing a Host "master" to request the running of special procedures?

3) Shall Host programmers normally write, debug, and implement IMP programs?

4) Shall the Host-IMP interface be implemented by the Host, the network contractor, or partly by both?

5) Shall the IMP be used to do data transformations:

   a) Such as reformatting ASCII? - hosts had characters in diff. formats? - should IMP cope w/those problems? No.

   b) Modify the protocol from its Host style to that of a remote Host?

After considerable thought, we have reached the conclusion that the IMPs and their operation should be initially implemented with the maximum logical separation from Hosts and Host programmers that can possibly be obtained. We feel that this logical separation is crucial for the early and reliable implementation of the net. There are several reasons for this opinion. First, if our own experience with time-shared systems is any guide, Host computer systems will undergo continual modification of both hardware and software, and there will be frequent and lengthy service interruptions. Second, each Host would handle storage of IMP data/programs in a different way, with a high probability of requiring differences in the IMP programs to handle such Host differences; at the very least, timing would certainly be unique to each location. Third, debugging and maintaining a system where one machine *controls* the program of other machines in real time is risky; establishing a situation in which mistakes in a controlling Host could damage the net is, in our view, an unreasonable initial design plan. Fourth, if the network contractor must become familiar with hardware/software of many different Host types, the cost, duration, and difficulty of the initial implementation will skyrocket.

The question of Host programmers writing code for the IMP deserves special attention. We can certainly see the pressures for such an arrangement; not only might it be technically convenient, but the idea of an untouchable computer on one's premises is a bit hard to take. The RFQ clearly showed ARPA's concern over this issue with talk of locks and memory protection. Several issues affect this question:

1) The program will be organized to perform most crucial tasks within interrupt routines, and great attention will have to be applied to issues of timing. Under such circumstances, we feel that protected memory alone offers only token protection and that a supervisor mode to protect I/O control, and yet to permit adequate timing control, would be difficult to implement reliably, and would greatly complicate the software design.

2) A Host written program presumably would be intimately related to the Host-IMP interface and the rest of the IMP program — but the control of this interface and the control of Host-IMP data rate is crucial to proper network management. We feel that any such process should be tailored as carefully as possible, with full knowledge of the entire program design.

3) Finally, it seems obvious that *initially* only the IMP contractor will understand the IMP program, while later some Host programmers may be inclined to study it and become expert. This is especially true in the early stages of net development, when the IMP program will be changing frequently.

There is an interesting and instructive analogy which may be drawn between the IMP-communication-subnet and the telephone network. Despite changing times and changing views about "foreign attachments" to the phone system, the rigid position that customers may not tamper with telephone equipment *has* contributed to the reliability of the phone network. Similarly, if customers initially avoid IMP programming, the reliability of the net will be enhanced. [illegible]

From these deliberations we reached the following conclusion: initially, Host programmers should not be allowed to program the IMPs. This should be accomplished legislatively, and no special technical barriers should be attempted. Later, when the network runs, the issue should be re-examined; if particular Host programmers are sufficiently interested to become expert, it may well make sense to have a standard authorization procedure for modification that allows or even encourages Host programmer participation.

Thus, we have tailored this proposal to arrange a high degree of isolation between things that happen in the IMP and things that happen in the Host. This general approach has a number of implications:

1) As discussed above, in early use of the network, only the network contractor will program the IMPs. Special tailoring of IMP programs for a given Host will initially be avoided to the maximum possible extent; if there is a sufficient reason for special tailoring of an IMP program at a given Host, such special tailoring shall be implemented by the IMP contractor. [We have, however, made provision in the IMP program design to permit data transformation routines to exist in a background fashion that does not upset the timing of urgent tasks.]

2) The program in the IMPs shall in no way be affected in real time by any Host. That is to say, the Host shall not be used for program storage for the IMP, the Host will not be used for message buffer storage for the IMPs, and the Host shall not be used to reload or restart the IMPs. Most activity, such as the gathering of statistics, which are desirable on the part of the IMPs, will be built in as a regularly run part of the IMP.

3) We feel that the interface between the IMP and the Host should be implemented in two physically separate parts. Specifically, we believe that the part of the interface associated with the IMP should be standard at all sites and should end in a cable connector with a specific description of the signals on that cable. We then believe that a matching half of the interface, special to each Host, should be implemented by the Host or under the direction of the Host.

This proposal does not cover the design or implementation of the Host side of the two-part Host-IMP interfaces. We believe this logical separation is most important and will permit the network contractor to avoid becoming enmeshed in the software/hardware problems of the Host sites.

We believe that the problems of bringing such a net into existence, even with these logical simplifications, are so severe that the additional complexities which might be caused by more tightly coupled relations to Hosts could make it difficult for the net to function at any early time.

B. Communications Issues

From a communications viewpoint, IMP-to-IMP communication will be substantively different from communication between an IMP and its Host in either direction. The channel between any two IMPs is a synchronous channel, synchrony being effected by fixed rate message transmission and the continual retransmission of a SYN character during all gaps. Channels between IMPs and their Hosts are asynchronous and transmissions regulated by control signals.

While a routine class of control signals has been finessed in inter-IMP communications, there are still important functions to be fulfilled by control messages. These range over the ability of an IMP to indicate the ups and downs of his Host, the prerogatives of a command Host, the possibilities of doing directed experimentation within the network, remote debugging and updating, and interrogations of other IMPs. Our initial position is again one of expedience and dictates as few such messages as possible in the genesis of the network. With expansion and evolution, the volume of messages will undoubtedly increase.

As to the question of permissible message length, our initial design philosophy has been in general accord with the details of the RFQ. Message length has been limited and messages segmented into packets for purposes of error control. While a certain freedom has been lost by limiting message length, and while segmentation imposes the burden of reassembly, the returns in error control seem more than justified. Experience with the network may indicate other parameter values for packet and message length; we have allowed for such modification.

As a packet is transmitted from one IMP to the next, it remains stored in the sending IMP until acknowledged by the receiving IMP. Thus, the way is clear for a receiving IMP to discard incoming packets, if the occasion demands, by not acknowledging them. Retransmissions are instituted if acknowledgments are not forthcoming within a suitable time period. Negative acknowledgments are insufficient, unnecessary, and not proposed.

Note that storage is required in each IMP for forwarding purposes; because an IMP is also saddled with the reassembly of messages destined for its Host, which also requires storage space, potential conflict arises. Our design ensures that neither function can preempt all available storage.

The final communications issue involves who or what qualifies as an initiating source or ultimate destination. We consider it unnecessarily restrictive to ban several simultaneous communications from one Host site to another. Our primitive, designated a "link," is the association of a program at one Host with a program at another.

C. Operational Problems

The network will be a very difficult system to operate, at least initially. It is a complex interconnection of sizable quantities of equipment distributed over much of the continental U.S. Clearly, careful attention must be given to the problems of reliability, failure detection and recovery, start up, debugging and updating IMP programs, and recovery from system congestion. These are the principal operational problems that we have considered and they are discussed below.

Reliability is a primary problem. The computer field has a history of long, difficult and time-consuming struggles with unreliable hardware. [The quantity of hardware that is required for reliable operation of the network is almost overwhelming.] A single communication between two Hosts requires that a minimum of four computers as well as a collection of interface hardware and common carrier equipment operate correctly.

Moreover, (the IMPs are expected to operate unattended for long periods,) without marginal checking or daily preventive maintenance. They must withstand power surges and the interference of nearby line printers, high-voltage scopes, or even an occasional radio or radar transmitter. It will be uneconomical to pay for on-site maintenance personnel, so on-call service will be used and an outage might take hours or days to fix, once detected.

Many features usually only included in the design of militarized hardware will enhance the reliability performance of equipment that is to be used in "comfortable" environments. Even in laboratory environments, people do accidently push up against, bang, kick, drop, shake, vibrate, heat and cool equipment and subject it to dust, unusual humidity conditions, power-line transients of various sorts, and electromagnetic interference. To help meet the reliability requirements on the system, we therefore propose the use of a computer for which standard ruggedized options have been designed and delivered. The inclusion of such ruggedized options will increase the reliability of the system at a slightly increased cost. We feel that the extra expenditure of funds is justified. Note that we do not suggest meeting military specifications but merely that we attempt to include ruggedization and electromagnetic interference protection features to a sensible extent. Although "evidence" of the benefits of ruggedized equipment is hard to provide, we cite one example: At Lincoln Laboratory, UNIVAC militarized equipment was used in laboratory environments and the reliability performance of this hardware was greatly improved over the reliability performance of computer hardware of comparable complexity in more ordinary packaging.

Careful attention must also be paid to the protection of interfaces and the isolation of IMPs from communication troubles and Host troubles. The IMPs should run without interruption whether the Host is running, not running, powered-up, powered-down, cables connected, cables unconnected, or in any of the various possible combinations. Similarly, anything which happens to the IMP must not be allowed to effect the operation of the Host. In the same fashion, the IMP must be protected from failures in the communications equipment and the failure of a given line must not be allowed to affect the operation of the IMP.

The design of the interfaces must, from the outset, provide for protection and isolation of major components of the system. Particular attention must also be given to grounding problems and the electrical issues associated with complex interconnection. There has been considerable experience that ground loops and other errors in grounding techniques have been the cause of great difficulties in bringing together assemblages of hardware by different manufacturers.

In a system of this complexity it is important that orderly procedures exist for checking and testing various portions of the system. [An IMP must be able to test itself, but, even more importantly, an IMP must be able to test all of the surrounding digital hardware to which it is connected.] In the implementation of complex real time digital computer systems, it is generally the interface equipment and connections of external equipment that cause the time-consuming debugging problems. If diagnostic programs are written for the IMP which can check all this equipment in a sensible and careful way, not only will the initial installation be facilitated, but also later debugging and maintenance will be greatly simplified.

There are two general classes of such programs which must be available. Some programs must be a normal part of the operating real time main program and these programs would periodically and routinely check the operation of the IMP itself and also check the operation of the local hardware to whatever extent is possible in normal operation. This periodic check should be coupled to an automatic alarm or restart that will alert site personnel or reload the system if a check has not been made within some designated time period. This feature will protect against an IMP halt or program loop.

A second class of programs should be specialized diagnostics which cannot be run when the system is in real time operation. One crucial aspect of this test mode operation is the necessity for testing the input and output connections to the telephone lines. Specifically, input and output lines must be able to be cross-patched on the line side of the modem. This will permit an IMP to send messages that will go through all the interface equipment, through the modem, out on the line, back across the local patch, and back into the computer system. It is anticipated that the modems will include such facilities for cross-patching, but it is important that the IMP be able to effect this cross-patch connection under program control. The IMP could then sequentially test all its input-output telephone line connections under an automatic program.

During the initial phases of network operation and even after the network becomes operational, the IMP program is likely to be changed often. When there are failures in connecting equipment, it may be necessary to use the IMP for debugging. For both of these cases, there must be a simple way to reload the main IMP program and to restart it.

There are a number of ways in which reloading might be accomplished. We considered using the local Host for this function, but this would make the operation of the IMP depend strongly upon its Host — undesirable because failure in the Host could jeopardize the network. We have also considered loading one IMP from another, but have rejected this approach for the initial implementation. At first, we would like to avoid the added complexity that loading via the network would introduce. Another level of commands would have to be provided and special features would have to be added to prevent accidental reload and propagation of errors. Another attractive alternative involves the use of a very small back-up store for program reloading. These are all viable alternatives, and are well worth considering for a large net of 20 nodes. As we gain some experience with the net, we will want to consider automatic types of reloading.

For the 4-node net the reloading problems are less critical and a simple inexpensive solution is proposed in the form of a paper tape reader at each IMP. In addition, we propose a simple start-up procedure whereby the master program can be loaded, the IMP started on its ordinary initial test sequence and sequential connection be made to the lines, followed by ordinary real time operation — with only one or two button pushes. Thus when a new issue of a program is delivered to a given Host location, the person assigned to the operation of the IMP can presumably reload and restart that program by loading the tape and pressing the buttons. As a result of a momentary failure, the IMP program may be placed in an inoperative state. In such an event, the automatic failure detection system would request a reloading of the program.

After IMPs have been installed in the field, updated versions of the software will be distributed via paper tape. As long as the network has a small number of nodes, paper tape should prove to be an adequate vehicle for communicating program changes. When the network becomes large, this procedure may prove unwieldy if program changes occur too often. In this event, it may prove desirable to implement an updating and reloading procedure using an automatic local loading procedure or the network facilities themselves. This, however, is a problem we will avoid initially.

A more serious problem occurs where hardware or software bugs are encountered in the field-installed network. If these same bugs do not manifest themselves in the laboratory prototype, then field debugging will be required. It would be desirable to be able to accomplish this debugging from a single site, but this would require facilities for examining the state of each IMP, reading and writing core and reading active registers remotely.

Once again, we do not feel that we can afford in the initial system the additional complexity that such features would introduce. Instead, we propose to adopt a somewhat less elegant but certainly less complicated and less expensive procedure, namely locating sufficiently experienced technical personnel at each of the initial 4 sites, as required, to do the debugging. In later versions of the system, more elegant debugging facilities will be provided as they prove useful.

D. Implementation

The amount of equipment in the first four nodes of the net is substantial. Furthermore, connecting a particular computer to a particular modem through a newly designed interface box represents a new configuration. To avoid field retrofits, it is important that a prototype be exercised before the design is frozen. The prototype would provide a machine for programming at the earliest possible moment. Finally, the checkout of production units with modems and with the production interface hardware could be done at a central location, under controlled conditions, prior to field installations.

The suggested prototype facility would initially consist of a non-ruggedized computer for programming use, with peripherals to facilitate programming. Within a few months, the first ruggedized unit with prototype interfaces, both to the line and to a Host, would be installed in the facility. At the same time, the initial machine would be retrofitted with prototype interfaces.

Exercising these prototype units would precede a design freeze of the deliverable equipment. Still later, the facility would be used to test the deliverable units prior to field installation.

The majority of hardware for this system should be available "off the shelf." However, there is one important exception: the interface between IMP and modem is a component for which on the order of 100 copies might be needed for a 20-IMP system. Since this component is rather complex, including check registers and considerable sensing gates, the cost of an implementation with standard available modules is somewhat high. For the 20-IMP system we estimated that more than 1/2 million dollars might be involved. At some point in time, this interface component should probably be specially engineered to reduce the production price. This development is not now included in our present bid, and the initial 4 units are considered to be constructed in a standard fashion. However, we recommend this development and we would plan to explore how it might best be accomplished.

Our considered choice for the computer to be used for the IMPs is the Honeywell (Computer Control) DDP-516. Factors affecting our decision were speed (1 µs memory), excellent I/O structure, flexible instruction set, sufficient word length (16 bits), very good reliability (especially in the ruggedized version) and reasonable cost. The DDP-516 is sufficiently powerful to provide the requisite safety factor for the implementation of the initial network. We anticipate that it will be able to handle a reasonable increase in processing demands that might occur as the network evolves.

During the implementation phase, the common carrier should assume a role of "active interested participation" and not merely a role of a supplier. Various contractors should be able to deal with a single cognizant office within the common carrier. The contracts for the common carrier should encourage the establishment of and support for a network office staffed by people dedicated to a successful network. Such strong support is necessary for several reasons. The prototype facility requires early availability of two modem units. Consultation will be necessary upon first IMP-MODEM Connection, and for MODEM field installations. In particular, talented assistance will be required to test MODEMS, and isolate line problems.

The design, implementation and checkout of the 4-node system is a large undertaking. We do not think that we can accomplish all of the work required within the very short time scale specified in the RFQ even though we have done much of the hardware and software design already. Instead, we propose a slightly longer time period for the performance of the contract.

A detailed schedule of work is shown in Fig. IV-1 of Chapter IV. Some of the principal milestones on that schedule are: delivery of prototype computer at end of month 1; completion of hardware and software design by month 2; delivery of prototype interfaces to BBN by month 3; demonstration of prototype system by month 7; delivery of first IMP during month 8; delivery of the remaining three IMPs at a rate of one per month thereafter. Documentation of the system will be kept current as shown on the schedule.

E. Network Experimentation

The ARPA network is experimental on at least two levels. The first is as an experiment in communication between a number of different and diverse communities of researchers. It will be important to learn what use they are making of the newly available remote facilities. However, other than simple location to location connection and message statistics, the IMPs will not be able to provide any information to characterize this use since the internal structure of messages is opaque to IMPs. An important aspect of this network construction or related research will be to consider what kinds of data should be gathered within Hosts to facilitate building models of user behavior. Simple stochastic models can, of course, be built with the available data.

The second level of experiment involves the network as a complex store and forward communications net. Our general policy of strong initial Host-IMP independence has important implications on how data is extracted. Statistics of operation at each IMP will be gathered by a standard program run at regular intervals. A message containing the resultant data will be sent to the UCLA, or other monitoring facility, as convenient. Thus, a cross section of network behavior will be available at all times at the network measurement center. This regular reporting is our substitute for information on demand which would require successful propagation and interpretation of control signals which we feel will compromise early success. In addition to the time slice sample of network behavior, we propose a second reporting capability which will allow individual messages to be "traced."

Simulation studies should be conducted in parallel with direct experimentation on the network. We have constructed an interactive model of the network with display capabilities. It is described in some detail in Appendix H. We have already used it to experiment with different strategies for dealing with buffer congestion.

CHAPTER III: PROPOSED SYSTEM DESIGN

A. Introduction

In this section we present our proposed system design. We begin by describing the most important features of the design, followed by a description of the overall hardware configuration of the IMP. The main part of this chapter is devoted to a detailed description of the process of message communication, including the primary aspects of network message flow and the suggested network protocol. We discuss the function of the IMP/MODEM Interface and the IMP/Host interface. The logical organization of the IMP buffer storage is then described in detail. The potential causes of network congestion are summarized along with the provisions we have included for handling this situation. Next we discuss line quality determination and rerouting. Questions of fault detection, status examination, and reporting procedures are also discussed. The end of this chapter is devoted to the main program structure and the support software.

We have introduced a great many novel features into our system design that we feel should be mentioned explicitly. These features, in conjunction with the ideas which were expressed in the chapter on Technical Overview, form the core of our original contribution to the network design.

Our experience convinced us that it was wrong to plan for an *initial* network that permitted a sizeable degree of external and remote control of IMPs. Consequently, as one important feature of our design, we have proposed a network composed of highly autonomous IMPs. Once the network is demonstrated to be successful, then remote control can be added, slowly and carefully. Messages are processed by an IMP using information which has been received from other IMPs and Host computers in the network, but special control messages or other external control signals are initially avoided to the greatest possible extent. One specific consequence of this policy is that the IMPs measure performance of the network on a regular basis and report in special messages to the network measurement center (presumably at UCLA).

A second important feature of our design is the provision of a *tracing* capability which permits the operation of the net to be studied in great detail. Any message may contain a "trace bit", and each IMP which handles such a message generates a special report describing its detailed handling of the message; the collection of such special reports permits reconstruction of the history of such messages as they traverse the system. This technique permits highly flexible sampled study of the network.

We have also included an automatic trouble reporting capability which detects a variety of network difficulties such as line quality deterioration, and reports them to an interested Host (perhaps, the network measurement center).

A principal feature of our system is a provision for letting IMPs throw away packets which they have received but have not yet acknowledged. Each IMP transmits packets to other IMPs at its own discretion. Each time an IMP receives *and accepts* a packet it returns a positive acknowledgment to the transmitting IMP. The transmitting IMP retains its copy of the packet until it receives the positive acknowledgment. The transmitting IMP will retransmit the packet if an acknowledgment is not received within a time-out period. It will continue to try transmissions, via a different route if necessary, until such time as a positive acknowledgment is returned. We have explicitly avoided the use of negative acknowledgments which we feel are insufficient and consequently redundant.

We have carefully provided for the preservation of natural word boundaries in transmissions between computers with equal word sizes (a thing which, despite intuition, does not tend to "happen naturally"). We introduce a technique of padding and marking which neatly and generally allows the beginning and end of a message to be clearly indicated to a destination Host without requiring the Host programs to count bits. [Although we have made an effort to suggest a network protocol that allows the Hosts a great deal of flexibility, this is a difficult technical area, and we would plan to examine further the problems associated with Host-Host word reformating.]

Another important feature of our design is a hardware modification to the IMP computer that permits the program to set an interrupt. This trick permits *three* levels of priority in the operational program (interrupt routines, urgent task routines, and background), which, in turn, has an important bearing on the IMP Program's ability to handle occasional time-consuming word-rate tasks (such as ASCII conversion, or other data transformation).

The Host computers have a few responsibilities for participation in the network. Specifically, the Host must provide a network-linking Program within its operating system to accept standard format network messages and to generate network messages in accordance with this standard format. The Host message includes identification information that accompanies the message from the source to the final destination. The Host computer must not present a message of over 8080 bits to the IMP. Larger transmissions must therefore be broken up by a Host into a sequence of such messages.

The network is carefully designed to protect and deliver messages from the source Host to the destination Host. The operation is self contained, and does not in any way constrain the procedures a Host may use in communicating with other Hosts.

B. General Discussion of the IMP

The overall configuration of an IMP includes a Honeywell DDP-516 computer, which has a 0.96 μs cycle-time, a 16 bit word length and 12K of memory (expandable), 16 channels of priority interrupts (expandable), a relative-time clock, and a 16 channel data multiplexor as shown in Fig. III-1. Also shown are several special interfaces, specifically one to the Host, and one to each modem. A paper tape reader has been included because we feel a very strong need for a device which does not depend upon the network or any Host computer for the loading of an IMP program. We believe that this is a simple, reliable and inexpensive way to read in new versions of a program during the initial phases of network operation. A teletype is required for maintenance of the IMP computer, but is not used by the main program and can be disconnected and removed during normal operation. A specially designed set of status-indicator lights are provided.

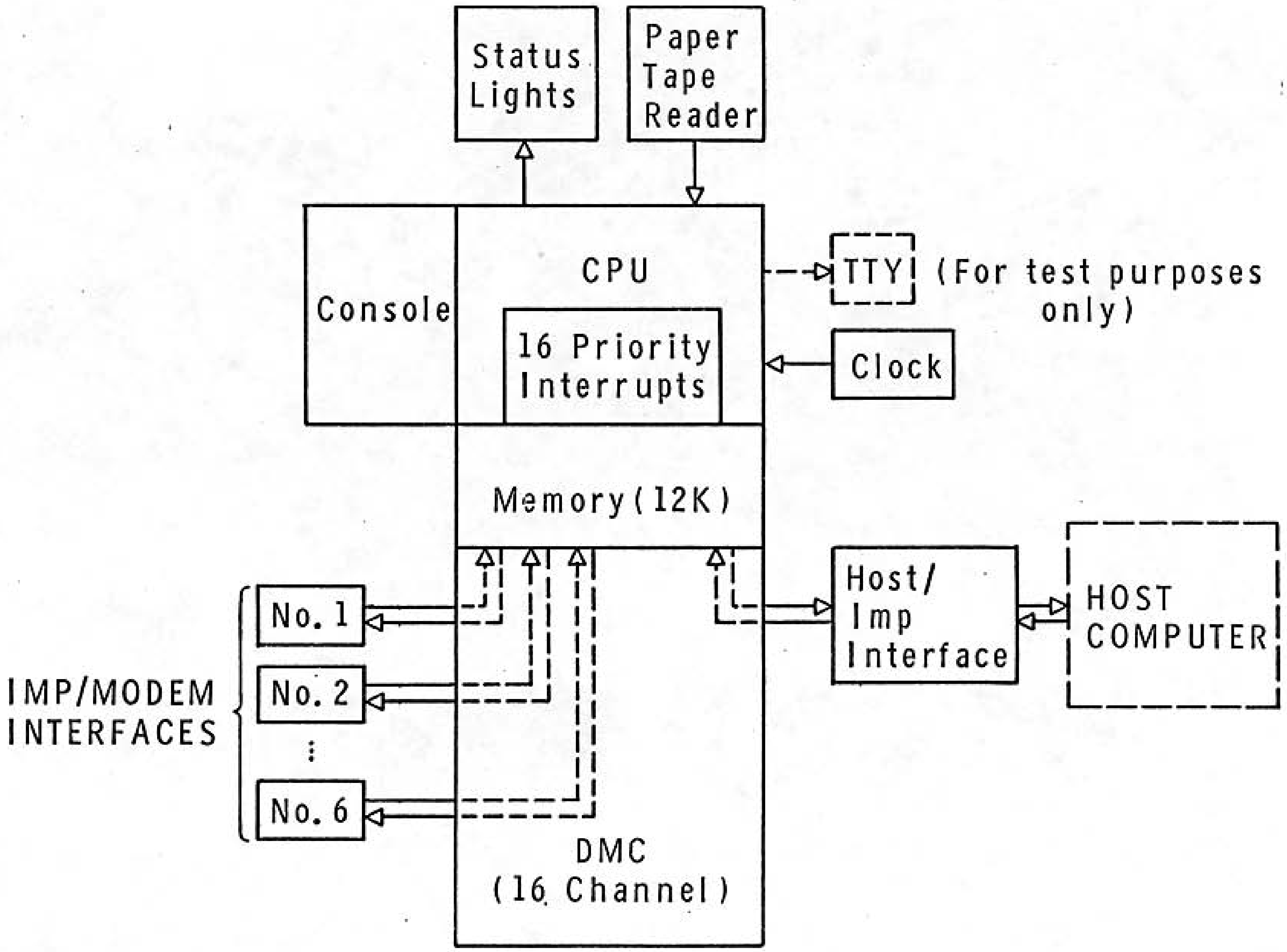


FIG. III-1 IMP CONFIGURATION.

for use by the IMP program to report trouble conditions to local Host personnel or to maintenance personnel without necessitating a halt in normal program operation.

The IMPs in the initial network will each have three built-in full duplex modem interfaces, but the interface design is modular and may be extended up to as many as six units, without a change in packaging.

The IMP, including all interface hardware, will be packaged in a single 69" × 24" × 28" rugged cabinet. (See Plate I.)

C. Host-Host Protocol and the Notion of Links

It is important to draw a sharp line between the responsibility of the network facilities in transmitting information and the responsibility of the Host organization for developing and adopting procedures for utilizing this facility. However, in considering the system design, it became clear that we would have to pay some degree of attention to limitations that the network protocol might place on the Host use of the network. We reached the conclusion that a network protocol that satisfactorily achieves the transmission requirement might nonetheless adversely affect the implementation by Host organizations of certain very desirable protocol features.

We considered the problems introduced when a multiplicity of user programs at a given Host installation are concurrently using the network and concluded that provisions for allowing such usage were rather important. The Host computers view the network as a means for passing messages back and forth between

PLATE 1. THE IMP.

parties rather than between pairs of Host computers themselves. We call a logical connection between two parties at remote Host computers a *link*. Many different links may exist simultaneously between a pair of Host computers. As illustrated in Fig. III-2, our network protocol permits many concurrent links to time-share the same physical network facilities. These links are established, identified, and maintained by a network program in each Host computer that effectively multiplexes outgoing messages from the parties into the network and distributes incoming messages to the appropriate parties as illustrated in Fig. III-3. Writing and maintaining the Host's network program is, of course, the responsibility of the individual Hosts.

An identification number is assigned by each Host computer to each network party in his machine. The party that initiates a link is known as the *caller*. The identification number of the caller is used as an identification number for the link and, in conjunction with the identity of the two Host computers, uniquely identifies the link. Each message which the Host network program presents to the network contains several pieces of information used by the network. One of these is the link identification number. The network uses this number to control the flow of messages and passes it along to the receiving Host.

A message is designated by its link and its direction of travel. (Source and destination are terms which identify the direction of travel.) Thus, complete identification for a message consists of the following four items:

1) Identity of Source Host;

2) Identity of Destination Host;

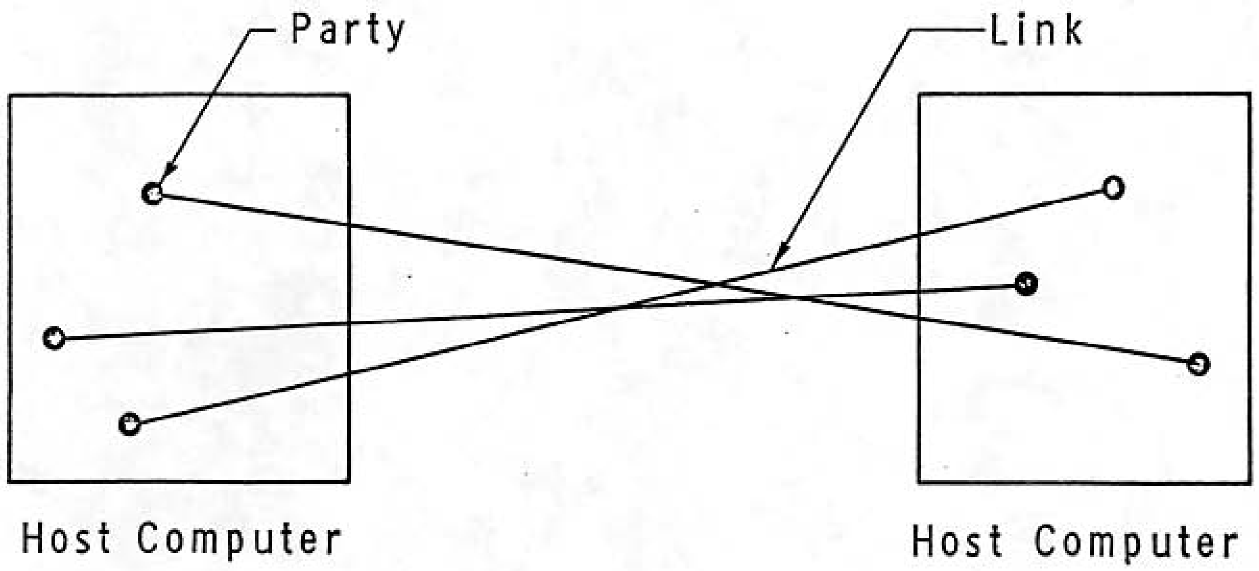


FIG. III-2 MULTIPLE HOST-TO-HOST LINKS.

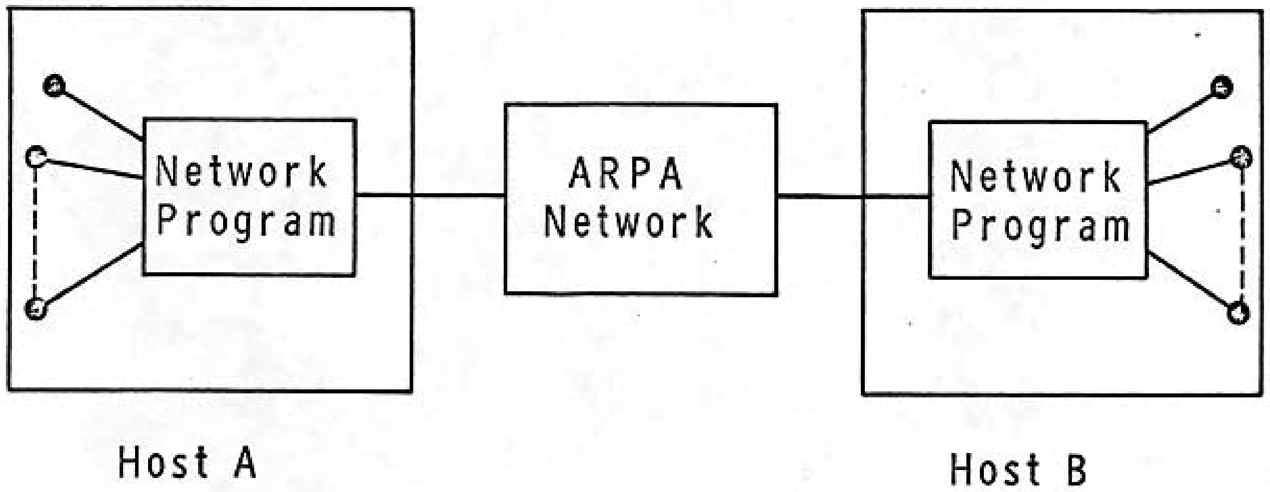


FIG. III-3 MULTIPLEXED HOST-TO-HOST LINKS.

3) Link identification number; and

4) Caller location (at source or at destination).

For example, if party n in Host A calls Host B, the message will be identified as going from source A to destination B and the caller for the link will be party n at the source. A return message from Host B on this link is identified as going from source B to destination A and the caller for the link will be party n at the destination.

We introduce the notion of a *link* early in this design discussion primarily because we wish to include the link identification number as an integral part of the identification information passed from Host to IMP, from IMP to IMP in the network, and finally from the destination IMP to the destination Host.

D. Messages and Packets; HOST-IMP, IMP-IMP, and IMP-HOST Protocol

Hosts communicate with each other via sequences of messages. A message is taken into an IMP from its Host computer in segments. These segments are formed into packets and separately shipped out by the IMP into the network. They are reassembled at the destination IMP and delivered in sequence to the receiving Host, who obtains them as a single unit. Thus the segmentation of a message during transmission is completely invisible to the Host computers.

The transmitting Host attaches identifying information to the beginning of each message which it passes to its IMP. The IMP forms a *header* by adding further information for network use. The header is then attached to each segment of the message.

The transmitting hardware computes parity check digits that are shipped with each segment and that are used for error detection. The destination IMP performs an error check, strips off the header from each segment in the course of reassembly and attaches identifying information at the beginning of the reassembled message for use by the destination Host.

A message from a Host is legislatively limited to be less than 8080 bits, and is sent to its IMP via a single block transfer. The hardware interface detects the end of the block transfer. Messages vary in size up to the 8080 bit limit. The first sixteen bits of each message which a Host sends to an IMP for a transmission are prescribed by the standard network protocol as follows:

> Eight bits are allocated to the link identification number, five bits are allocated to identifying the destination Host, one bit is presented for tagging selected messages which are to be traced through the network, and two bits are reserved as spares. The tracing is discussed more fully in a later section. The format for these 16 bits of Host information is illustrated in Fig. III-4.

The HOST/IMP Interface transfers bits serially from the Host and forms them into 16 bit IMP words. The IMP program takes groups of successive words in segments and stores them in separate buffer regions until the end of the message has been recognized. The first buffer accepts up to 64 IMP words from the Host (1024 bits including the 16 bits of Host information). Each succeeding buffer accepts up to 63 words (1008 bits). Thus, the maximum Host message of 8080 bits will be taken by the IMP in exactly 8 segments.

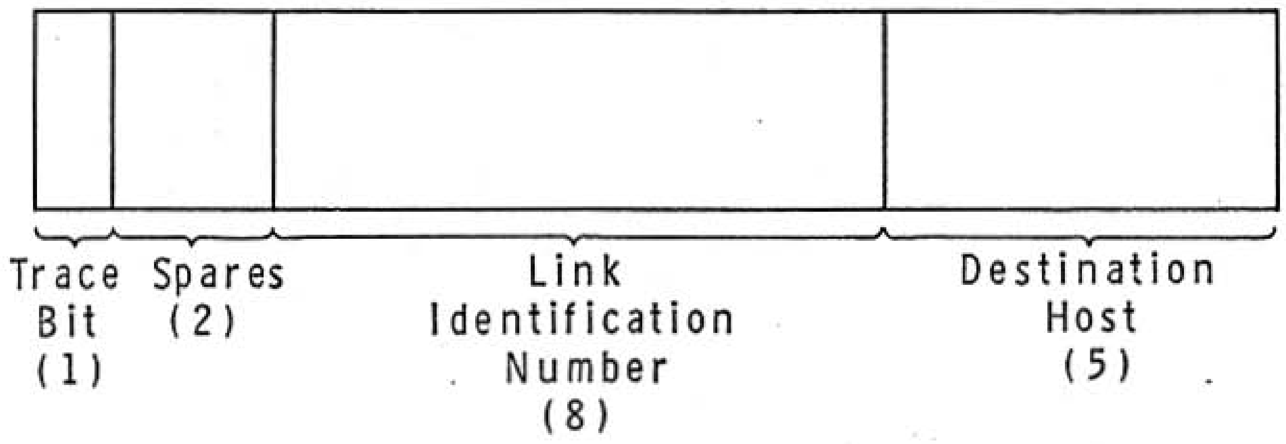


FIG. III-4 HOST-TO-IMP INFORMATION FORMAT.

The IMP now formats each segment into a *packet* for transmission into the network. The structure of a formatted packet as it appears in the originating IMP memory is shown in Fig. III-5. The output hardware prefaces the packet into the phone line with the character pair DLE STX to mark the packet beginning for the receiving channel hardware. The packet is then transmitted serially over the communcation lines beginning with the left most bit of the first header word and proceeding through the header and the text. The channel hardware computes 24 parity check digits, which it attaches after the text, and follows them with the two ASCII control characters DLE ETX to mark the end of the packet for the receiving channel hardware.

A continuous stream of the ASCII control character SYN is transmitted by the channel hardware between packet transmissions. These are used to separate packets and to obtain character synchronization in the receiving channel hardware. Thus the packet appears on the communication line as shown in Fig. III-6.

The receiving channel hardware locks into character synchronization on a bit-by-bit search for an 8 bit SYN code. Once synchronization has been obtained, the channel hardware looks for the first occurrence of DLE STX and succeeding characters are fed into the IMP memory until the DLE ETX at the end of the packet is detected. The hardware also computes a 24 bit error check based upon the received data, which should equal zero if no errors have occurred in transmission.

The received data between the STX and the ETX is written into the IMP memory and appears in the buffer as shown in Fig. III-7.

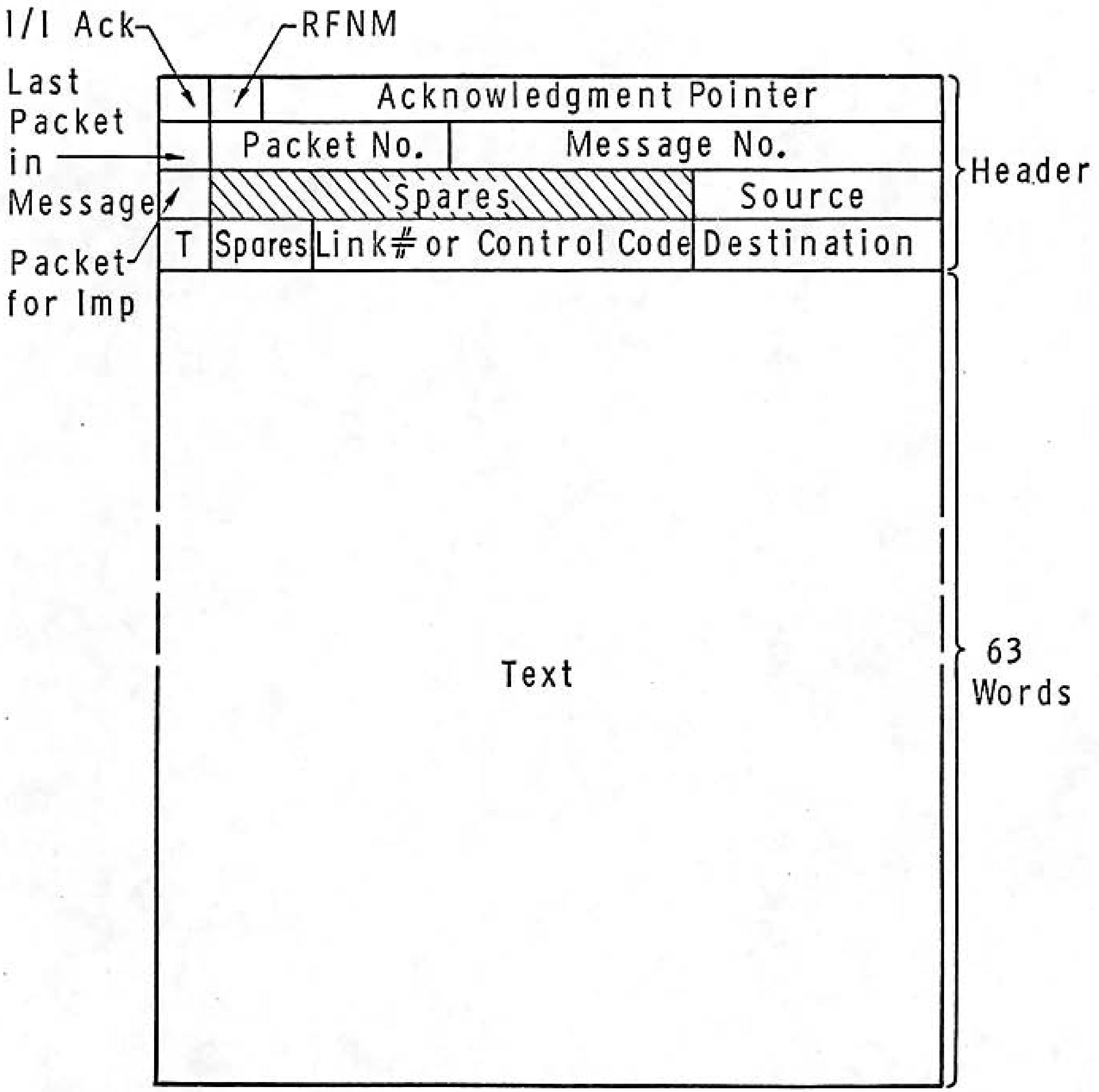


FIG. III-5 ORIGINATING IMP PACKET STRUCTURE.

```
      S S D S                          C C C D E S S
• • • Y Y L T   Header        Text     C C C L T Y Y • • •
      N N E X__________  __________1 2 3 E X N N
```

FIG. III-6 COMMUNICATION LINE PACKET FORMAT.

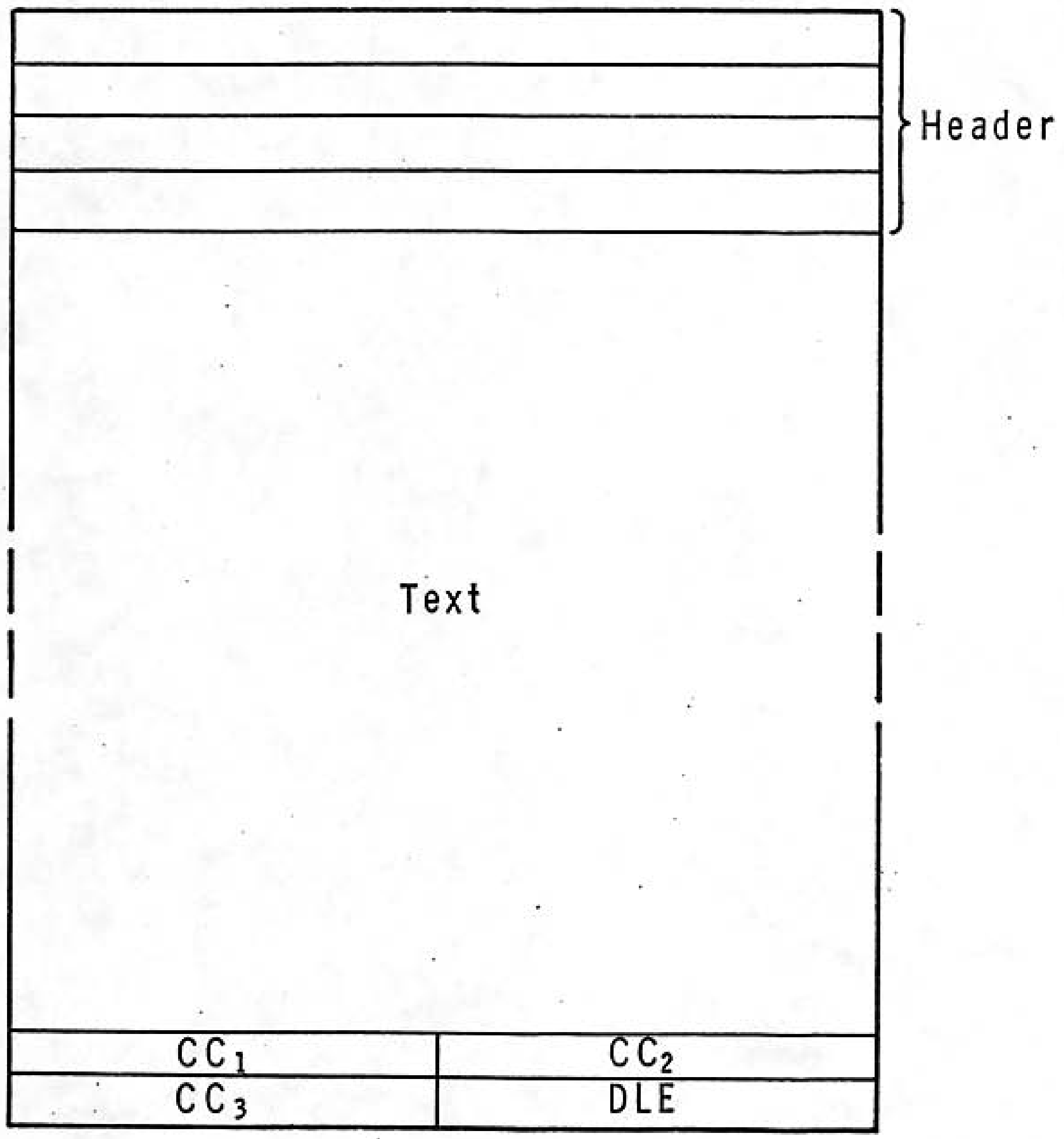


FIG. III-7 PACKET FORMAT AS RECEIVED FROM MODEM INTERFACE.

If the receiving IMP is not the final destination, everything except the last two words is fed to the appropriate output channel hardware. The channel hardware recomputes 24 parity check digits and appends these as described earlier, together with the DLE STX.

Eventually, the packet will arrive at the destination IMP. In fact, eventually all the packets of the message will arrive at the destination IMP, although not necessarily in the order of transmission.

The destination IMP sorts received packets according to the link identification as specified in the header. When all packets of the message have arrived, it delivers them in the proper order to its Host.

Packets within a given message are numbered sequentially by the transmitting IMP in the second word of the header and the last packet is specially marked by an identifying bit in the same word. This allows the receiving IMP to determine the order of the packets and to know when all packets have been received.

The receiving IMP strips off the header and the final two words from each packet before sending it on to the Host. Furthermore, 16 bits are sent to the Host preceding the text of the first packet. The Host network program uses these bits to identify the link in sorting incoming messages. The format for these 16 bits is shown in Fig. III-8.

Thus, the complete message is finally delivered to the destination Host in the same form as it left the transmitting Host, with the source in place of the destination in the Host information.

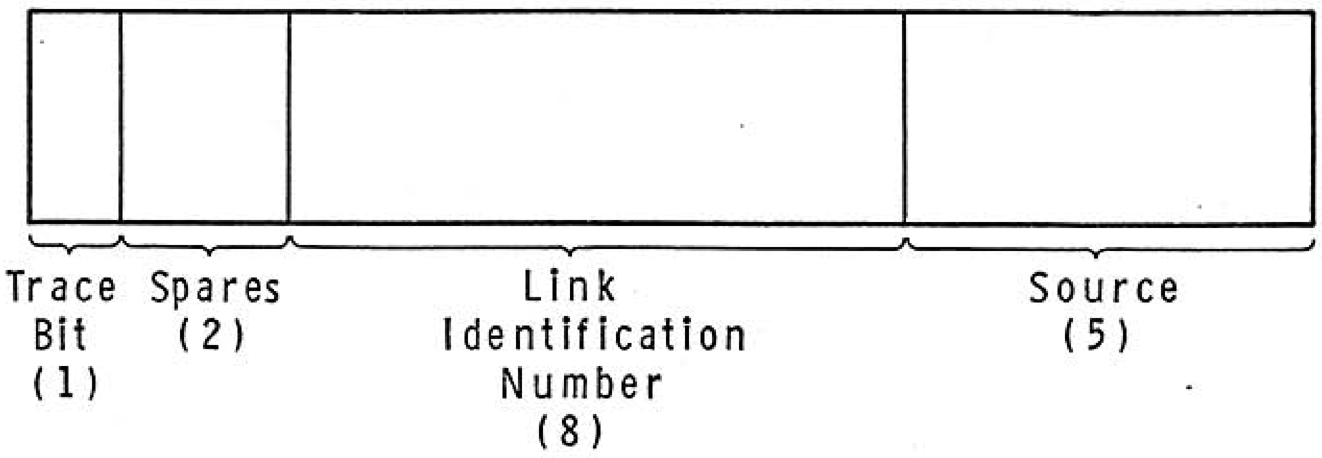


FIG. III-8 IMP-TO-HOST INFORMATION FORMAT.

flow control RFNM.

E. Acknowledgment Procedures

We now discuss two kinds of messages which will be used to control flow in the network: "IMP-to-IMP acknowledgments," and end-to-end "Requests For Next Message."

1. IMP-to-IMP acknowledgment of packets

The process of communicating a message from the source to the destination IMP uses the store and forward services of intermediate IMPs. As a packet moves from one IMP to the next, it is stored in each IMP until a positive IMP-to-IMP acknowledgment message is returned from the succeeding IMP. This ackowledgment indicates that the packet was received without error and was accepted. The acknowledgment is returned over the same line on which the packet arrived. A 14 bit acknowledgment pointer, containing the memory address of the first word of the transmitted packet, is included in the header of the packet to simplify the process of releasing that packet when acknowledged. (The packet identity data are checked before releasing the packet; the acknowledgment pointer simply avoids searching.)

To send an acknowledgment of a received packet, an IMP simply returns a packet (without text) whose header is an exact copy of the header of the received packet, but with the first bit of the first word changed to a one. This bit is called the IMP-to-IMP acknowledgment bit and is the first item sensed by the IMP program upon receipt of every packet. (The source and destination do not apply in the usual way to the acknowledgment message itself.)

Once an IMP has accepted a packet and returned a positive acknowledgment, it hangs on to that packet tenaciously until it, in turn, receives an acknowledgment. Under no other circumstances (except Host or IMP malfunction) will an IMP discard a packet after it has generated a positive acknowledgment. However, an IMP is always free to discard a packet by simply not returning a positive acknowledgment. It may do this for any of several reasons: the packet may have been received in error, the IMP may be busy, the IMP buffer storage may be full, and so forth.

Packets which are not recognized by the receiving channel hardware, which incur errors in transmission, or which are not accepted for whatever reason, are not acknowledged. At the transmitting IMP, the situation is readily detected by the absence of a returned acknowledgment within a reasonable time interval. Such packets are simply retransmitted.

Acknowledgments are themselves not acknowledged, although of course they are error checked in the usual fashion. Loss of an acknowledgment results in the eventual retransmission of the packet. The resulting duplication is sorted out at the destination IMP by use of the message number and packet number in the header.

There are no negative acknowledgments in our proposed design. They cannot be relied on to induce retransmission. If a negative acknowledgment is lost, one must resort to a time out procedure, in which case, the negative acknowledgment becomes redundant. Since the time out procedure must, therefore, always be used, we include it in our design.

2. Request-For-Next-Message (RFNM)

A central concern of network protocol is the problem of congestion at a destination IMP. This congestion must be reflected back into corrective quenching of the flow toward that point from other parts of the net. Otherwise, it would give rise to the discard of packets at the destination, blockage of those packets at the contiguous IMPs and the congestion would rapidly propogate back through the network. If the sources of packets for that destination continue sending, this congestion would rapidly affect the flow of other messages within the net.

There are at least two kinds of quenching which could be adopted.

1) We could limit the *degree* of congestion of remote IMPs that can be caused by any particular congested Host or link. For example, if each IMP only accepted, say, two messages for any given destination, the congestion would be limited to that amount and, eventually, the source would be unable to transmit additional new packets toward the troublesome destination.

2) We could try to limit congestion at the source directly by shutting off any new packets directed toward the troublesome destination. This action could be accomplished in either of two ways: a control message could be dispatched when congestion actually has occurred, or successive transmissions could routinely require a "clear-to-send" indication from the destination.

Although we have tried to avoid control messages in our design wherever possible, we decided in this case initially to use the control message technique. *We propose to avert congestion, by* *only allowing a source IMP to send one message at a time over a given link.* After sending a message over a link, a source IMP must delay sending the next message until a "Request for next message over link X" (RFNM) packet is end-to-end returned from the destination IMP. (Note that all packets of a single message, and/or messages over different links between the same two Hosts, may be sent into the net without delay.) The RFNM is passed along to the Host, who may use it to schedule the servicing of links. This technique only quenches individual links and therefore a limit is placed on the total number of links which a transmitting IMP will accept from its Host.

This technique has several important advantages and two disadvantages. The advantages are:

1) The demand for reassembly storage at the destination IMP for use by a given link is limited to eight packets.

2) When congestion occurs, flow is *automatically* quenched without any control messages. If source IMPs do *not* get new RFNM's, they do *not* send new messages.

3) Since the flow is quenched at the source, large numbers of packets from a given link neither enter the net nor flow about the net trying to get to the congested destination. Thus, congestion of other parts of the net by a single link is avoided.

Obviously, the main disadvantage is that waiting for RFNM packets may reduce the effective rate over a given single link. We have examined this disadvantage and have decided that it is not serious, for the following reasons:

1) Depending upon the number of active links, there may or may not be a reduction of the effective rate between two Hosts. When several links are established in a given Host computer, the messages will be time multiplexed. The RFNM delay in that case may already naturally appear in the system.

2) Since the message length will probably be bi-modal (very short or very long) and since very short packets are probably generated by humans, the RFNM delay is insignificant for processes at human rates. For very long messages, in the worst case of no time multiplexing and an unoccupied line, we estimate the reduction in effective rate to be only 30%.

A second disadvantage is the increase in number of control messages. Since RFNM's are very short, however, we feel that this effect is also not serious.

The use of an RFNM control message is a very clean, simple, and positive way to avoid some nasty and confusing problems. We are not fully satisfied that the doctrine is optimum, but, so far, we have been unable to see a clearly superior alternative. We therefore propose to use RFNM control of congestion in the initial design. During the implementation and testing, we will continue to consider this issue in an attempt to determine whether other alternatives appear to be more advantageous.

F. Examples of Message Flow

The chart on the following pages shows the flow of packets involved in transmitting a message from one Host to another. The

EVENT				STATE OF THE NETWORK				
Comments	Packet	From	To	h1	i1	i2	i3	h3
Host 1 has two packets for Host 3				21				
	1	h1	i1	2	1			
	1	i1	i3	2	1		1	
	2	h1	i1		21		1	
Acknowledgment returned	1a	i3	i1		2		1	
	2	i1	i3		2		21	
	2a	i3	i1				21	
	12	i3	h3				r	21
RFNM goes back to h1	r	i3	i1		r		r	
RFNM also acknowledged	ra	i1	i3		r			
	r	i1	h1	r				
Host 1 has two packets for Host 3				21				
	1	h1	i1	2	1			
	2	h1	i1		21			
	1	i1	i3		21		1	
Packet 1 acknowledgment lost	1a	i3	i1		21		1	
	2	i1	i3		21		21	
	2a	i3	i1		1		21	
Packet 1 rerouted*	1	i1	i2		1	1	21	
	1a	i2	i1			1	21	**
Packet 1 arrives second time	1	i2	i3			1	21	
	1a	i3	i2				21	
	12	i3	h3				r	21
	r	i3	i1		r		r	
	ra	i1	i3		r			
	r	i1	h1	r				

EVENT	STATE OF THE NETWORK							
Host 1 has two packets for Host 3				21				
	1	h1	i1	2	1			
Error on line (i.e., Packet 1 does not get to i3)	1	i1	i3	2	1			
Packet 1 rerouted*	1	i1	i2	2	1	1		
	2	h1	i1		21	1		
	2	i1	i3		21	1	2	
	2a	i3	i1		1	1	2	
	1a	i2	i1			1	2	
	1	i2	i3			1	12	
	1a	i3	i2				12	
Packets 1 & 2 get sorted	12	i3	h3				r	21
	r	i3	i1		r		r	
	ra	i1	i3		r			
	r	i1	h1	r				

LEGEND:

21 = 12 =	Packet 1 and Packet 2
1	Packet 1
2	Packet 2
1a	Packet 1 acknowledgment
2a	Packet 2 acknowledgment
h1	Host 1
i3	IMP 3
r	Ready for next message
ra	RFNM acknowledgment

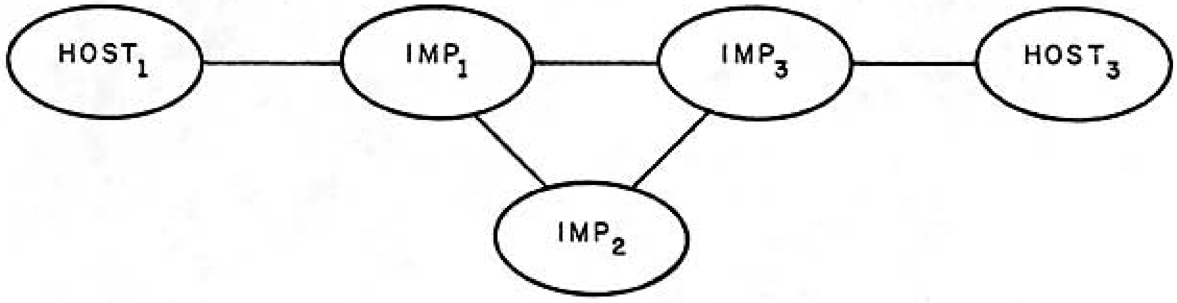
*A time out period elapses before Packet 1 is rerouted. In the third example, other events which are not shown (because they are irrelevant for this example) prevent Packet 2 from being transferred from Host 1 to IMP 1 during this interval.

**In this example, the duplicate of Packet 1 merely overlays the one in IMP memory, effectively deleting it. The reassembled message could have entered the Host any time in the bracketed interval, before the arrival of the duplicate packet. In this case, the message number of the duplicate allows it to be discarded.

packets of the message, the acknowledgment packets, and the ready for next message packet are indicated assuming that the message being transmitted contains two packets.

The chart includes three examples: in the first, transmission is completed without any problem; in the second, an IMP-to-IMP acknowledgment for one packet is lost; and in the third, a packet encounters difficulty due to line error. Although the events within the examples are ordered, we emphasize that most of the events occur asynchronously and could be ordered in many other ways. Equal time does not pass between events.

The relevant portion of the network assumed for the examples is:



G. Word Length Mismatch

We discuss two aspects of word length mismatch: first, the obvious need for formatting that occurs between computers of different word length; and second, since mismatched words may lead to messages that end in the middle of words, the need for marking the exact beginning and ends of a message to permit unambiguous recognition.

There are several logical ways in which the reformatting of a word length mismatch might conceivably be handled. One may decide upon a word-by-word algorithm, where transfers from long to short machines involve truncation, and where transfers from short to long machines deposit a partial word. Unfortunately, there are many slightly different ways to do this and, worse, it is very undesirable in many applications. A second possibility is to list a number of kinds of reformatting and have a given message carry a code for the required type of reformatting. We feel that such a plan would be unreasonable for a 19 node net. Finally, one may beg the question and just send a bit stream, leaving to the individual Hosts the task of reformatting.

We have decided to adopt almost this latter position. Our design guarantees that between Hosts of identical word length the natural word boundries are preserved. (This is not as easy as it sounds.) But, reformatting in general will be initially left to the Hosts. At a later time, the IMP program might be used to alleviate further this set of problems.

The second problem is that of recognizing the end of a message at the receiving Host. There are two general solutions to this, one of which is to locate the last bit in the message by counting from the beginning (using either a transmitted count or an agreed upon fixed value). The other general solution requires that the ends be marked in an unambiguous way. We have chosen the latter scheme, which marks the end of the message by appending a "one" followed by zeroes after the last bit in the message. This process is called *padding* and is accomplished by the hardware in the HOST/IMP interfaces. The receiving Host can therefore identify the end of the message.

As a message passes from the transmitted Host to its IMP, the hardware appends a one to the bit string when it receives the end of message signal. This bit may fall, in general, in any position of an IMP word somewhere in the last packet. The hardware then fills any remaining bits of this word with trailing zeros. The format of the last packet of a message as it thus appears in the IMP memory is shown in Fig. III-9.

The packet appears in the destination IMP in the same format (plus, of course, the check characters and the final DLE).

As the last packet is serially shifted into the Host through the interface, the last bit from the IMP (which in our example is the fifth trailing zero in the padding) will fall, in general, somewhere in the middle of the receiving Host's final word. The remaining bits in this word are filled in by the Host's special interface hardware with additional trailing zeros. (Note that a one is purposely omitted here.) Thus the packet appears in the receiving Host with a one immediately following the last bit in the message, followed by a string of zero or more trailing zeros that terminate at a Host word boundary. The last word in the receiving bit stream does not necessarily contain the last bit in the message, as it may contain nothing but padded zeros.

Another occasion for inserting a form of marking data arises at the beginning of a message. The transmitting Host, in general, arranges that the text of a message begins at a word boundary. Since the network protocol requires the first 16 bits of a message to contain Host information, there will thus, in general, be a gap between the end of that identification and the beginning

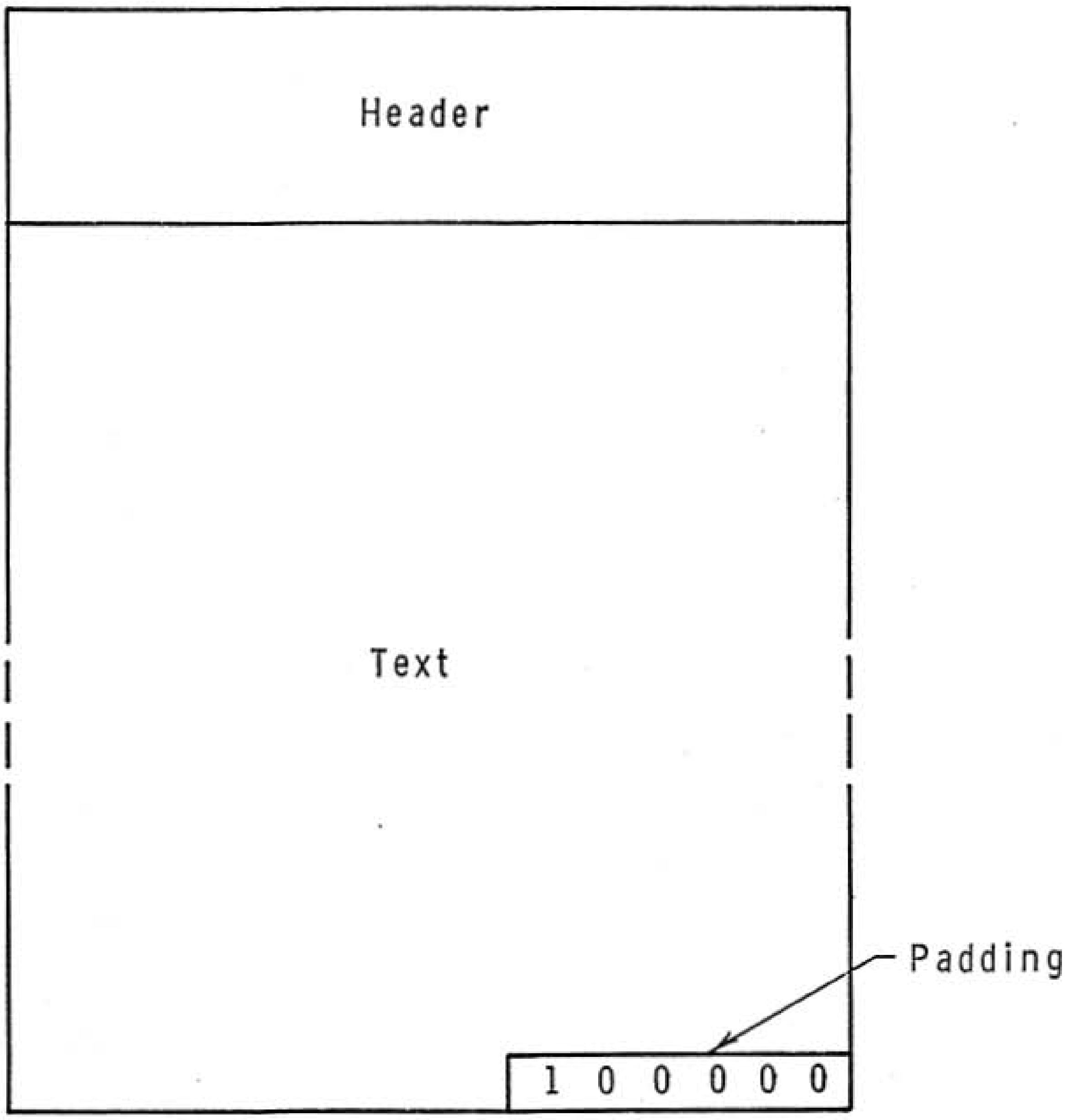


FIG. III-9 FORMAT OF LAST PACKET OF A MESSAGE.

of the text. This gap is preserved in transmission to the destination Host and must be marked in a way which the destination Host can recognize as not forming part of the message. This *marking* must be inserted by the transmitting Host's software, and consists of a one preceding the first bit of the text and, in turn, preceded by a zero or more zeros to fill up the gap.

In Fig. III-10 we illustrate one complete set of Host and IMP buffers, corresponding to a message of slightly under two full packets. We have selected in our example a 22 bit source Host word length and a 20 bit destination Host. We have specifically indicated both the padding and the marking in the figure.

H. Hardware Description and Interface Operation

A block diagram of the IMP computer and its interfaces to the Host and phone line modems is shown in Fig. III-11. The area between the heavy vertical lines shows the IMP system itself; the area to the left is specialized Host equipment; the area to the right is phone line equipment. There are from one to six full-duplex IMP/MODEM interface units and one (or optionally two) HOST/IMP interface unit. The DMC provides the only direct access to and from memory, other than that for the CPU itself. The functioning of these units is described briefly in this section and in great detail (with drawings and timing diagrams) in Appendix E.

The IMP/MODEM Interface Unit is full duplex. It serializes and deserializes data for the Modem to and from memory. In the absence of outgoing messages, it loads a continuous string of SYN characters onto the line. It does special formatting for output, and character sensing for the beginning and end of input

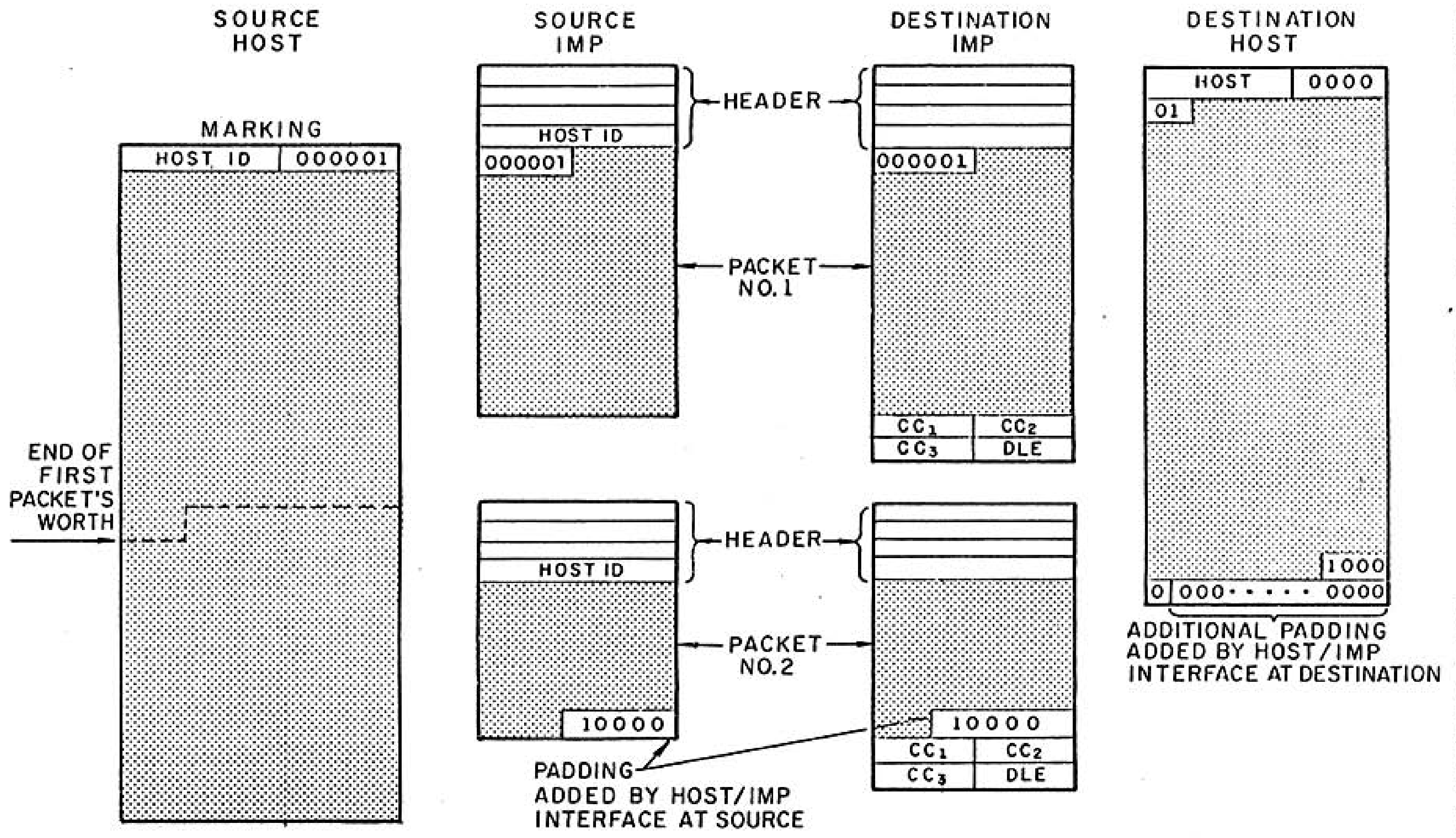


FIG. III-10 HOST AND IMP BUFFER FORMAT FOR A TWO-PACKET MESSAGE.

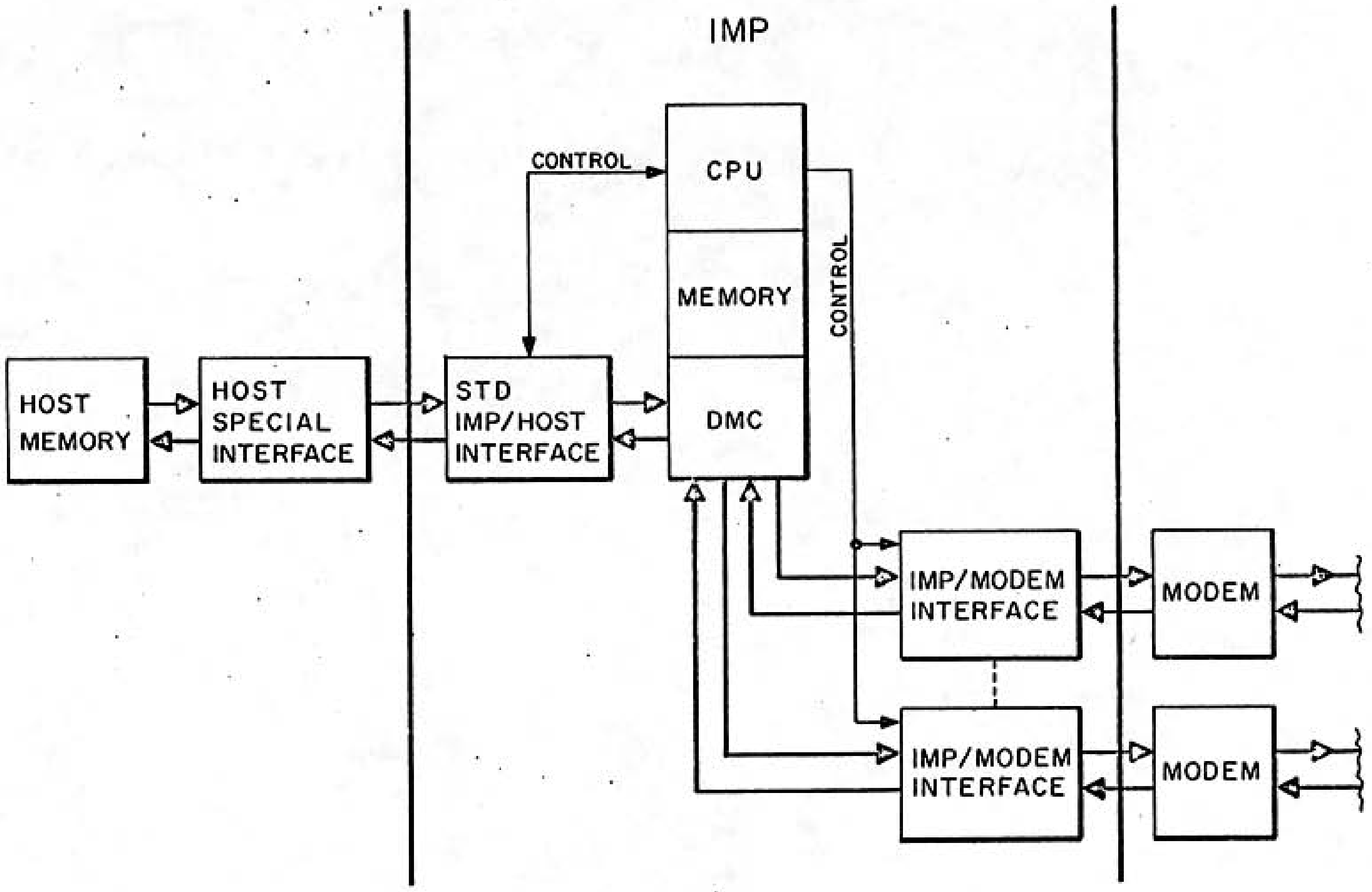


FIG. III-11 GENERAL VIEW OF A TYPICAL IMP SYSTEM.

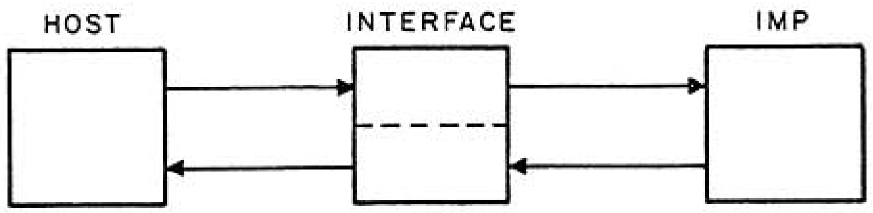
messages. It includes construction and testing of parity check digits and fault detection and reporting. Its timing is controlled primarily by the Modem.

The standard HOST/IMP Interface Unit is full duplex and passes messages bit-serially to and from the Host special interface. It also deserializes and serializes words to and from the IMP memory. Communication across the interface with the Host is asynchronous to allow for maximum flexibility.

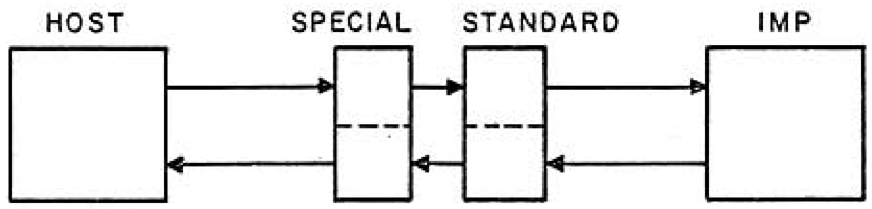
The relative-time clock is a 16-bit counter indexed every 20 μs and may be read into the Accumulator. The full clock count repeats approximately every 1.3 sec and an Interrupt is generated on the turnover of an appropriate high order bit. This bit is selected to give an interrupt frequency which is convenient for use by the program in performing time outs for retransmission of packets.

1. <u>The HOST/IMP interface unit</u>

There is no general rule whereby the HOST/IMP Interface Unit can determine in which direction (Host-to-IMP or IMP-to-Host) information will next have to be processed. The equipment must therefore be capable of starting a transmission in either direction. Transmission requests arrive asynchronously for the two directions and, rather than trying to sort them out for processing over a half duplex channel, a full duplex channel is provided. The primary advantage of this is simplicity and it also provides the capability for concurrent transmission in both directions. The HOST/IMP Interface is thus divided logically into two parallel channels — one for either direction — as indicated in the following figure.



Because Hosts vary in word length, signal forms, and logic for receiving and transmitting information, we further subdivide "vertically" the HOST/IMP Interface, into two separate units:



The right hand Unit contains logic that is standard for all HOST/IMP Interfaces. The left hand unit contains the special equipment for interfacing directly to the particular Host. Standard signals pass between these two halves; all special logic and signal adjustments (which vary from Host to Host) are handled in the left hand portion. Power for the standard unit is directly connected to the IMP's power — i.e., its power is turned on whenever IMP power is turned on. Power for the special unit is derived from the Host power system (or a separate supply) and will probably have a separate on/off switch.

Each participating Host will be responsible for the design and building of its own special unit that will mate to the standard unit according to fixed rules. In general, this special unit serves to serialize and deserialize information in whatever manner best suits the particular Host. The IMP-to-Host section of the special unit must perform the "padding with zeros" function discussed earlier.

Two levels of hardware handshaking take place between a Host and its IMP. At the meta-level, each needs to know whether the other is turned on and operational. The standard unit provides to the special unit (and it in turn to the Host in whatever way is appropriate) a signal which indicates that IMP power is up and that the IMP program has turned on a Ready indicator. The special unit presents a similar Host ready signal to the standard unit, and thence to the IMP. Each unit automatically monitors the readiness of the other, and if the other's readiness state changes, the unit will notify its parent computer; in the case of the IMP, by an interrupt. Thus, for example, should the Host computer fail or drop power, the IMP will be interrupted and can take appropriate action. Only when the Host returns to Ready, which requires not only reinstating power but also program turn on of the Host ready indicator in the special unit, will communications with the Host be re-established. Under normal operation, when either computer detects that the other has become ready, it will prepare to receive information. Thus, with both Host and IMP ready, each will be waiting for the other to transmit. As soon as information is provided by either one, it will flow across the Interface.

Thus, when the Host ready indicator comes on, the operational IMP program prepares to receive from its Host by setting up a pair of pointers used by the standard Host-to-IMP interface channel of the DMC. These pointers delineate a packet-sized buffer in the IMP memory. After they have been set, the IMP program issues an ACCEPT* command to the interface. Thereafter, when information becomes available from the Host, the standard interface unit takes it in serially and forms it into 16 bit-IMP words in an input buffer register. These words are stored into successive locations of the IMP memory buffer until the buffer area becomes full or until the message end is indicated by the Host. When either of these happens, information flow ceases and the IMP program is interrupted. In the case where the Host message ends, the hardware appends a trailing "one" followed by any "zeros" necessary to pad out a full 16-bit word. The interrupt routine will normally reset the pointers to another buffer location and restart the interface with a new ACCEPT command. Serial transmission makes the standard unit independent of Host word size, and requires only one data line driver and receiver. The interface unit is designed to accept bits from the Host at 1 MHz maximum rate (5 MHz circuits are used). The Host, of course, can slow this rate by controlling the flow of bits. Memory references in both computers will slow the rate well below the maximum.

When the IMP has set up memory pointers and is ready to transmit a packet into the Host, it starts the transmission via a GO command. The first word is then loaded from the IMP memory into the interface and the Host unit takes the bits serially. Each time 16 bits have been taken in, a new word is fetched from the IMP memory. When the buffer has been emptied, the program is

*Control commands to devices are delivered by execution of assigned OCP instructions. These instructions deliver appropriate control signals.

interrupted and normally prepares for the next transmission to the Host if any more buffers are waiting. When the IMP is ready to transmit the last packet of a message, it executes a special END command before starting the transmission with the GO. In this case, when the last bit of the packet is taken into the special Host unit, an end-of-message signal is also sent to the unit. This causes the special Host unit to pad the remaining bits of its final word with zeros before passing it to the Host with the "that's all" indication.

2. The IMP/MODEM interface unit

Each IMP connects to several (up to 6) telephone line modems each of which has a separate IMP/MODEM Interface unit. This unit converts outgoing information into serial form and assembles incoming serial information into 16-bit words which it places in the IMP memory. It also computes 24 parity check bits, which it transmits at the end of a packet and checks upon receiving a packet. As shown ~~in Fig. III-12~~, a modem consists of two logical halves, each producing clock signals and containing a single data line, one in and one out. The interface unit correspondingly contains two logically distinct sections, one dedicated to transferring output from the IMP to the modem and the other dedicated to transferring in the other direction. In the absence of outgoing messages, the output section sends a continuous stream of SYN characters to the modem. Fig. III-13 shows a typical packet buffer in the IMP memory from both the output and input points of view. In this presentation, only those elements of particular concern to the hardware are separated out. Thus header and text are not distinguished.

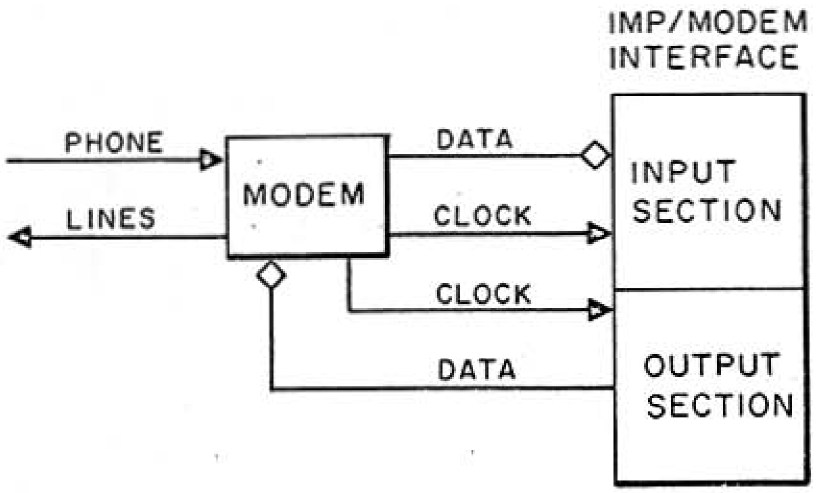


FIG. III-12 LOGIC VIEW OF MODEM.

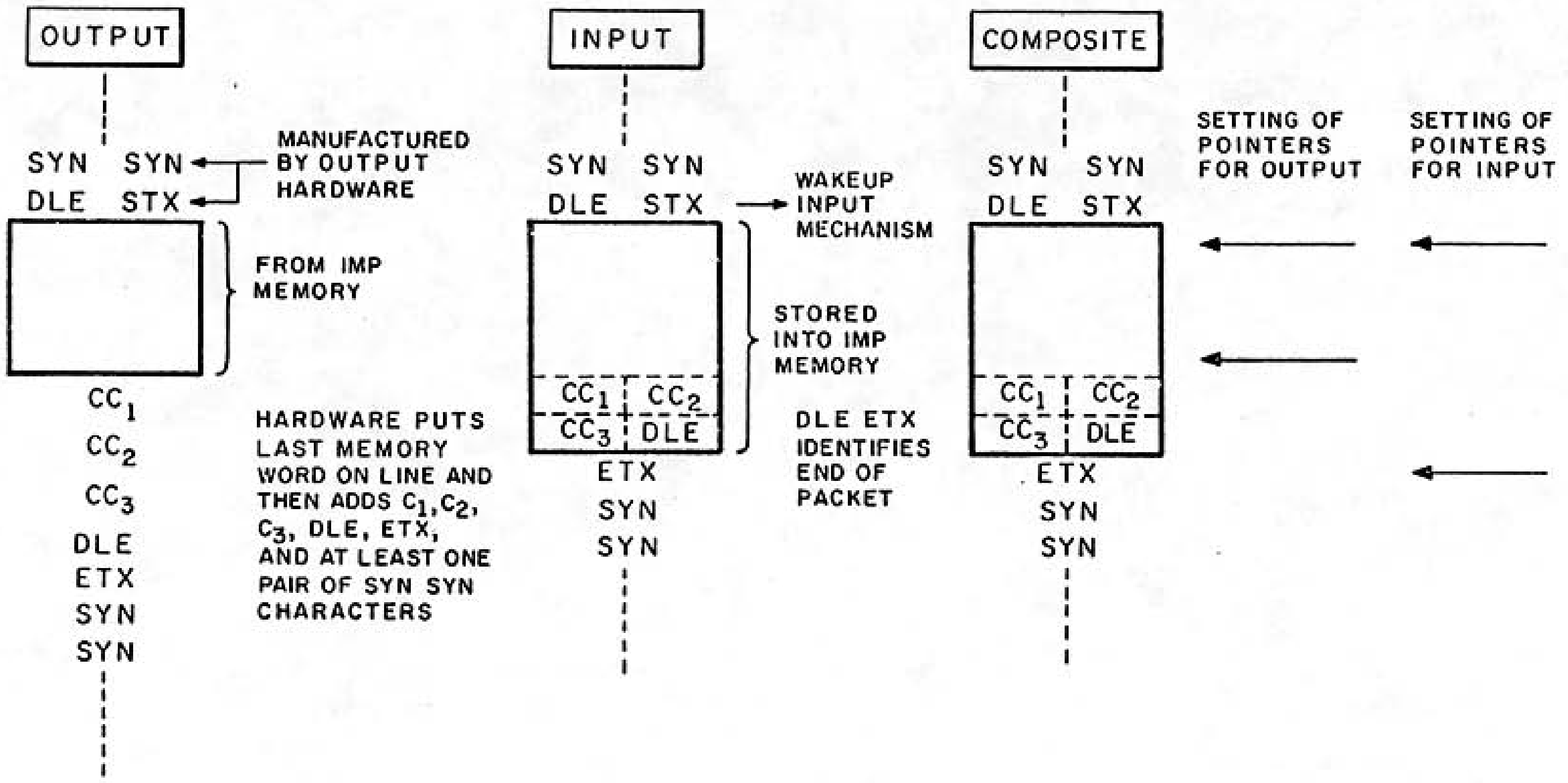

OUTPUT
SYN SYN
DLE STX
MANUFACTURED BY OUTPUT HARDWARE
FROM IMP MEMORY
CC1
CC2
CC3
DLE
ETX
SYN
SYN
HARDWARE PUTS LAST MEMORY WORD ON LINE AND THEN ADDS C1, C2, C3, DLE, ETX, AND AT LEAST ONE PAIR OF SYN SYN CHARACTERS
INPUT
SYN SYN
DLE STX
WAKEUP INPUT MECHANISM
STORED INTO IMP MEMORY
CC1 CC2
CC3 DLE
ETX
SYN
SYN
DLE ETX IDENTIFIES END OF PACKET
COMPOSITE
SYN SYN
DLE STX
CC1 CC2
CC3 DLE
ETX
SYN
SYN
SETTING OF POINTERS FOR OUTPUT
SETTING OF POINTERS FOR INPUT

FIG. III-13 PACKET BUFFER FORMAT.

After setting the output pointers, as shown, the IMP program notifies the output hardware that a packet is ready to be transmitted. The hardware then sends the character pair DLE STX and follows this with the data words taken from the IMP memory according to the pointers. When the DMC indicates that the entire packet has been sent, the hardware appends the check digits followed by the character pair DLE ETX and at least one pair of SYN characters. A string of SYN characters then follows until another transmission is initiated.

Additionally, the hardware monitors the data from memory for DLE characters and, upon finding one, immediately inserts another character, thus averting confusion resulting from a DLE within the packet. The receiving input unit deletes these extra DLEs. Of course, extra DLEs are not inserted with the hardware-generated DLEs.

The input hardware detects the DLE STX, which marks beginning of a message and loads into the IMP memory all characters between (but not including) the STX and ETX. Thus, the three check characters go into the memory on input followed by the DLE, which rounds out the last word. Any error indicated by the parity check is signaled to the computer. Note that the STX is not itself fed into memory but serves only to cue the input hardware to the start of the packet on the line. The bottom input pointer points to *one location beyond* the point where the last data word of a maximum-sized legal packet would be put. Normally, the input hardware recognizes the end of input by spotting the DLE ETX at the end of the packet. To assure that, if it misses this, input does not proceed to flood the IMP memory, input is cut off if the allocated IMP buffer fills up —

i.e., if *one more* than the expected maximum number of words arrives in a packet. An error is indicated to the IMP program in this case. Since the receiving input unit recognizes when a packet begins and ends by the DLE STX and DLE ETX characters enclosing the packet, there is no possibility of confusing the start or end of a message since DLE STX or DLE ETX character pairs can never occur *within* a message without being preceded by another DLE. The receiving input unit deletes the extra DLE's.

J. Organization of IMP Storage

Message packets are read into buffers in IMP storage as we have already discussed. Each incoming packet is allocated one free buffer selected from a free buffer pool. Pointers are set by the CPU to the beginning and end of the buffer and an input transfer is enabled. When a packet is read into memory, an interrupt signals the program upon completion of the transfer. If an error is detected, the buffer is returned to the free buffer pool. The packet, in effect, is discarded, since the buffer is now free to be overwritten. Otherwise, the packet is assumed to be correct.

Within an IMP, a packet is never moved from one buffer to another. It is read into one location in memory with a set of input pointers and taken out of the same location with a set of output pointers.

Approximately six thousand words of memory will be occupied by programs and the remainder will be available for buffers and program expansion. Each of the buffers contains about 70 words.

One of these is a free word allocated at the end of the buffer to detect the case where the buffer is about to be overflowed, due to the loss of the end of message indication. An interrupt will be generated during input if the moving pointer ever coincides with a pointer to this last cell. Approximately two additional words at the beginning of each buffer are used for holding queue pointers as discussed below and in Appendix F.

We distinguish between three types of packets in the IMP which we call store and forward packets, packets for the Host and packets for the IMP. A store and forward packet is one whose destination is another site. A packet for the IMP, defined implicitly, is handled by special IMP routines and does not require lengthy storage since the buffer is quickly released back into the free buffer pool.

The Host computer generates only store and forward packets or packets for its IMP. Packets that arrive over the communication lines may be either store and forward packets, packets for the Host, or packets for the IMP.

A packet for the Host computer may be a single packet message or part of a multiple packet message. Single packet messages, which are uniquely identified by the last-packet-in-message bit on packet number one, clearly require no reassembly and may be directly transmitted to the Host computer. When the first packet is received for a multiple packet message, seven additional buffers are removed from the free buffer pool and reserved. As each additional packet of this message arrives and is stored in a free buffer, one of the reserved buffers is released into the free buffer pool. When all packets of the message have been reassembled, the remaining unused reserved buffers are released and the complete message is sent to the Host. Waiting until the full message is assembled avoids the risk of tying up the channel to the Host in the middle of a message. The storage for these packets is called reassembly storage.

Each communication line has a buffer assigned to it which is unassigned upon receipt of an incoming error-checked packet, whereupon another buffer from the free buffer pool is assigned in its place.

A correctly received store and forward packet is placed on a queue for transmission over the first choice output communication line. An IMP with three communication lines has three such queues, one assigned to each line. Packets on each of the three queues are transmitted sequentially over the communication lines. There is also a similar queue for reassembled messages going to the Host.

We now discuss the maintenance of these queues. Upon arrival, each store and forward packet is placed at the end of a first choice queue which is determined from an entry in a routing table. Each queue is linked in two directions; so that from a given position on the queue, both the packet ahead of the current position and the packet behind the current position may be directly referenced and so additions or deletions in the middle of the queue can be made rapidly. In addition, the last packet in the queue is linked to the first packet, thus forming a circular queue. The last position on each circular queue is defined to be the position just behind the current service position.

There are certain packets which, upon arrival or generation, may be placed at the head of a queue at the current service position where they will be next in line for transmission. These may include all packets for IMPs and all short packets.

K. Buffer Congestion

We now discuss the subject of buffer congestion and the techniques that we have introduced to deal with it. We indicate the principle causes of buffer congestion, describe the kinds of difficulties which are caused by it and develop a number of simple strategies which either attempt to prevent buffer congestion from occurring or ensure the recovery from it.

Certain Host computers will be primary receivers of network messages and their corresponding IMPs will have a substantial portion of the buffer storage containing messages for the Host computer. Other IMPs will function essentially in the store and forward mode, containing significantly fewer messages for their own Host computers than for other IMPs in the network. IMPs such as these, which primarily store and forward messages, are critical links in the network. When they become congested, they affect the overall pattern of traffic flow.

An IMP is said to be congested whenever the contents of the free buffer pool falls below a level equal to the number of communication lines. There are several different causes of buffer congestion, the most serious of which is a malfunction. We discuss the effects of a malfunction later in the chapter. However, congestion can also occur during normal operation of the network due to transmission errors, line concentration, or reassembly.

Line errors may be expected to occur on the order of seconds apart. At 50,000 bits per second and line bit error probability of 10^{-5}, one error is expected every two seconds. However, the errors will undoubtedly be clustered so that the interval between error bursts will probably be over 10 seconds on the average. An IMP stores packets from the time they arrive until an acknowledgment is returned. Sufficient storage has been allocated to handle the reasonable peak loads of offered traffic and to allow for line errors.

Line concentration refers to the situation when messages arrive on several different communication lines and are intended for transmission over the same outgoing channel. Since a packet must be transmitted contiguously in time over a communication line, two packets cannot be simultaneously transmitted and therefore at least one of the packets must wait.

Buffer congestion may also occur if insufficient reassembly storage is available. For example, if 10 network users are logged into one system, all messages have 8 packets, and a buffer is 70 16-bit words, then 5K of core would be needed for reassembly alone, with all users simultaneously being reassembled. We may expect to be confronted from time to time with the situation where the IMP simply does not have enough buffers to do reassembly. Furthermore, if a Host computer does go down or if messages are fed to it over many links, the backup of packets into the rest of the network could cause the entire network to overload. The process of automatic rerouting which takes place when messages fail to get through on a primary route (as discussed in the following section) will tend to alleviate this situation.

In Section E (above) we already discussed the use of RFNM's for averting congestion. We now discuss several more techniques designed for coping with buffer congestion. To prevent buffer congestion from affecting reassembly, we lock in (i.e., reserve) seven more buffers for reassembly at the destination IMP when the first packet of a message arrives. A reassembly packet is accepted only if the addition of the seven additional buffers will not trespass on the 25% minimum store and forward buffer space. Buffer storage is conceptually divided into two sections, one to hold messages to and from the Host computer and the other used for store and forward packets. There is no fixed allocation of buffers into one category or the other. The amount of storage allocated to each is adjusted to meet the network demands. However, some fixed minimum percentage of the total number of buffers is always reserved for store and forward traffic. That is, an IMP is never allowed to block network traffic by assigning all its buffers for reassembly packets and outgoing messages from its Host. The minimum number of buffers that must always be available to the rest of the network for store and forward packets is an IMP program parameter. Initially, we will dedicate at least one quarter of the IMP buffers for such store and forward packets.

L. Line Quality Determination and Rerouting

We define the *quality* (Q) of a line as the time varying relation of received acknowledgments of a line to the total number of packets requiring acknowledgment transmitted over the line. Thus, the quality is a simple and direct measure of transmission success on the line. The quality of a broken line will rapidly drop to a very low value. Similarly, the quality of a line to a congested IMP which does not regularly acknowledge packets will also drop. This quality factor is used in two ways: to detect difficulties with the functioning of a line for statistics gathering and trouble reporting, and *as a criterion for rerouting.* In addition to the line quality, there is an *a priori* weighting of the lines that reflects the desirability of using each line to reach a given destination. This weighting is designated by the letter K. The determination of K for each line to each destination is a complex judgmental matter, reflecting not only the topology of the net but also knowledge, as it is gained, about known average traffic patterns. Such information comes from human analysis of network performance. The values of K are thus selected in advance, loaded into the IMP as required, and kept in a routing table.

Unless a line is disabled, when a packet first arrives in an IMP, ready to be sent to some other IMP, the packet is placed on a queue for the line with largest value of K. The line quality is thus not normally used in the initial transmission, thereby guaranteeing that lines are tried frequently in order to maintain an up-to-date estimate of Q. Of course, routing for *retransmission* is based on both the line quality and the K factor.

Regular checks are made on the status of all entries in the queues as part of a time out procedure, in order to consider the possiblity of retransmission. The algorithm which selects packets for retransmission works as follows: Each buffer on a queue has a "sent" bit which is set to one when the contents of the buffer have been transmitted. The bit is reset to zero if the buffer is to be retransmitted. During each time out procedure, a check is made to determine if a time out has occurred since the packet was last transmitted. If the packet was transmitted but has not timed out, the sent bit is left on. If the packet has timed out, a calculation is made to determine the most desirable route and the packet is routed accordingly. The calculation will be a simple function of the line quality and the preassigned weighting of the line.

We have not attempted to specify the alternate routing algorithm in greater detail at this time for two primary reasons. First, any reasonable algorithm will perform acceptably in the initial net since the connectivity is so limited. Secondly, we did not want to include as part of our proposed design, an *ad hoc* solution to a problem upon which the network performance will be critically dependent under heavy load. We plan to provide an algorithm which is adaptive, free from recurring loops, and reflects our best judgment on this matter.

We have designed and operated a network simulation program on our 940 computer. The program drives a CRT display that may be used to assist in the testing and simulation of various algorithms. See Appendix H for a brief description of this simulation program. This simulation will be a valuable instrument in studying improved routing algorithms. The algorithms can then be tested by actual network experimentation.

M. Network Introspection

As the network operates to service Hosts, it must monitor its own performance to detect faults, take corrective actions as required, and report on its own activity to various points in the network. The reporting function includes urgent messages about malfunctions, prompt comments about changing conditions, and more leisurely periodic summaries of statistical performance. In order to permit such monitoring, fault recovery, and reporting by the program, adequate "test points" must be built into the hardware and the operational software. In addition, decisions must be made as to where reports of various types should be sent: reports might go to a local Host, or to a "special" IMP run by the network contractor, or to ARPA, or to a particular special Host, or to some combination of these places. We do not feel that the choice of destinations is a crucial issue at this time, and for purposes of discussion we have assumed the existance of a "network measurement center" (NMC). This NMC is presumed to be a particular interested Host.

In the remainder of this section, we first discuss detection, reporting, and recovery from three kinds of faults, namely, Host faults, line faults and IMP faults. We then discuss the techniques to be used for gathering detailed information about network performance, and the reporting of that performance; finally we summarize the kinds of abnormal messages which will be generated in these processes.

1. Faults

1.1 Host Faults

If a Host actually goes off the air, either voluntarily or through a traumatic failure such as loss of power, a special Host ready indicator which resides in the IMP/HOST Interface (and is described in the Appendix on hardware) will be turned off. Any change of state of this indicator produces an interrupt of the IMP; thus, the IMP program may note the change and take action. If the shutdown was voluntary, the IMP may have been notified previously and therefore suitably modified its tables. If no prior notification has been received, the IMP informs the current remote users. A message saying "My Host is down" will be sent to users who try to login at unavailable Hosts. The normal result of a traumatic Host failure is not only the immediate quenching of additional messages from the sources, but a discarding of all packets in the net addressed to that Host upon their arrival at the destination IMP. When a Host comes back up after a down period, the ready status will change to on and the IMP will note this change. Test messages may also be used in this case to confirm proper operation of the channel to the Host.

A more difficult case occurs when the Host fails in some way which does not change its ready status, but which nonetheless destroys its ability to interact with the network. Such failures, for example, may be caused by software bugs, or minor hardware transients, which can cause programs to loop. In order for the IMP to detect such a situation, it will keep an indicator of the quality of communication with the Host. If normal IMP-Host message flow is greatly diminished for some comparatively long time, the IMP will assume that the Host is down and will take the same action as if the ready indicator had been turned off. To determine when the Host is again available involves the use of test messages from the IMP to the Host. The outage of the Host, even for extended periods, does not in any way affect the IMPs role in storing and forwarding other network messages.

1.2 Line Failures

The normal operational IMP program maintains up-to-date indications of the quality of every incoming and outgoing line. If the estimate of quality on a given line falls below a preset clip level (a program parameter), the IMP will inform local personnel by changing lights in the lights register, and will inform the NMC by producing a trouble report. This provides a relatively straightforward and positive procedure for keeping track of line troubles.

Checks of the lines will also be done during initialization of the IMP program, and also during scheduled and unscheduled maintenance of the line. A special IMP program will be able to cross patch each line under program control and test the Modem and Interfaces of each line. It is conceivable that such cross-patch testing could be built into the operational program at a later stage in the development of the network, but we do not plan to include it initially.

1.3 IMP Faults

Despite the extreme provisions for reliability built into the IMPs, faults will sometimes occur. Detection of these faults is necessary to ensure smooth operation of the network. In some cases (such as total failure), an IMP will be unable to detect trouble itself. Provision must be made for neighboring IMPs (which do detect such failure) to report this. Communication outside the network channel (e.g., by phone) will then be used to inform personnel at the site of the IMP of that IMP's malfunction.

On the other hand, the majority of IMP failures should be able to be detected at the IMP itself by making the operating program periodically reset a timing device. Failure to reset the timer before it times out will set a failure indicator.

This internal failure detector can communicate the failure to the failed IMP or to a maintenance person without resort to external communication. For this reason, we have included an internal failure detector utilizing a time-out period.

Having detected failure, there are several methods for implementating a restart. Certainly the simplest to implement at the outset is to arouse the Host operator with an alarm and allow him to load the system via the paper tape reader following the same *simple* procedure employed in start-up of new program versions. As the system evolves, automatic restart procedures could reduce the outage time caused by transient failure. Ideally, the IMP could restart automatically from an auxillary storage device capable of multiple restarts. Alternatively, one could restart by automatically reloading the IMP from its Host. (We do not favor involving the Host with this task.) Still another alternative is to reload one IMP from another by causing a loader to be put into operation in the failed IMP. This IMP, in turn, requests and checks the reloading of the operational program from a neighboring IMP.

We would tend to order these automatic restart alternatives on the basis of IMP autonomy and simplicity, and would thus tend to favor first an auxillary storage device, followed by restart from a neighboring IMP and, lastly, restart from the Host. The actual choice and implementation of automatic restart should be the subject of further study and experiment in the 4 node network. Initially, the IMPs should be restarted manually with paper tape following a hardware alarm. The 4 node IMP equipment will support experimental investigation of alternative automatic restart methods; the IMP will have a limited amount of protected memory and a suitable timer for this purpose.

An IMP which fails may be a critical node which cuts off some existing links. For example, a destination IMP failure cuts off all links to its Host. The network must respond appropriately to such an outage. All links through the IMP will quickly be blocked since no RFNM messages will get back to the sources. Packts trying to get through a down IMP will circulate in the system, trying to circumvent it. When the IMP comes back on the air, the messages will eventually reach the destination and be discarded.

Should an IMP be down for an extended period, some sort of mechanism is required to purge the system of undeliverable packets. We have not settled on a particular technique but have considered two possibilities. The first of these is to include in each packet a handover number that would increase on every IMP-to-IMP transfer and that would allow a discard of the packet when a (high) clip level is reached. An alternate approach is to have a Host generate special messages for this purpose.

2. Performance measurements

We propose two main techniques for gathering performance information on the operation of the network: (1) Regular measurement by each IMP of its internal performance; and transmission of that information on a periodic basis to the NMC and (2) the tracing of messages through the system, resulting in the generation of report packets about that message proceeding to the NMC for reconstruction of the message path.

a) Regular Data Gathering

Each IMP will include in its operational program a routine that will be run on a clock interrupt. Thus the program will run periodically independent of the load on the IMP at that time. This program will sample some program parameters and either save the values or running averages of these values. The following list provides examples:

1. Empty buffer count
2. Number of messages being reassembled
3. Queue length of output queues
4. Number of sent but not acknowledged buffers in each queue
5. Quality measures
6. Rate of inputs

The list of sample parameters will then be included in a special report message directed to the NMC. We believe that this regular technique of reporting will provide a comprehensive history of what the IMPs are doing. It naturally assumes some attention on the part of the NMC, but obviously remains a matter of choice.

b) Tracing

The other data gathering facility, which we believe will be exceptionally useful, we call tracing. A common notion in computer programming, tracing allows one to obtain either a small amount of information or a large amount of information as the trace proceeds. We believe that our network trace feature has the same extremely desirable flexibility.

Any or all messages may include a trace bit in the header. Messages with trace bits may be initiated by the NMC or by other Hosts. For example, trace bits could be put in some set fraction of each Host's messages. In fact, we can think of a number of techniques whereby trace bits could be added to messages on a sample basis. To give one more example, each IMP could be asked to include a trace bit in every mth IMP message. We believe this technique will permit occasional sampling or complete tracing of messages in the network.

When an IMP receives a message that includes a trace bit, it incurs the additional task of noting in detail how it handles that particular message. When the IMP has finally released that message, it must generate the special report about that message and send the report to the NMC. The NMC will thus receive a sequence of report messages for each message that contains a trace bit. It should then be possible for the NMC to generate a good representation of the path taken by that message, or by a group of messages in the network.

3. Summary of abnormal messages

Results of the introspection discussed above are transmitted by "abnormal" messages that are generated by IMPs for these special purposes; these abnormal messages are not part of the normal flow of data between Hosts. We believe that there will be a large number of packets of this type, but it is impossible to list them now with any confidence. However, we can distinguish between several kinds of packets, and provide an initial estimate of what types might exist.

We group the class of special packets into three categories. The first category contains those packets which only cross IMP/MODEM Interfaces and contain all IMP-to-IMP messages. The second category contains those messages which only cross an IMP/HOST Interface. The third category defines messages which cross one HOST/IMP Interface and one or more IMP/MODEM Interfaces. (If two HOST/IMP Interfaces are crossed, the message is a Host-to-Host message and considered to be part of the Host protocol.)

We list some of the special messages in each of these three categories:

1. Across IMP/MODEM INTERFACES
   a. Query
   b. Response

   c. IMP going down
   d. IMP back up
   e. Acknowledgment
   f. Ready for next message
   g. My Host is down

2. ACROSS IMP/HOST INTERFACES
   a. Query
   b. Response
   c. I am going down
   d. Ready for next message

3A. IMP TO REMOTE HOST
   a. Fault detected
   b. Report generation

B. HOST TO REMOTE IMP
   a. Change routing table

The above list contains some entries such as "My Host is down." In connection with messages such as these, we wish to here introduce the notion of busy signals. In making a telephone call, there is no indication, at the telephone and before the call is tried, that a line will be busy, out of order, or not answered. We feel that this is a powerful concept as applied to the network. For example, when an actual user at a Host site tries to use the network to call some other Host, *at that time* the network should try the call and then send back a message, finally reaching that user, which says, "Sorry, the Host you just tried to call is down." This arrangement has the advantage that as a given Host goes up and down it is not necessary for large numbers of control messages to flow around the network. To keep everyone informed of the instantaneous status of that

Host. Instead the status is made available "on request." This approach can be applied to many situations within the network, and we propose to apply it where possible. Naturally some status information will, in fact, be kept distributed, but we will try to minimize the number of different kinds of status tables that must be kept up-to-date.

N. The Operational IMP Program

Inasmuch as the operational program implements the strategy and protocol of the network, some discussion of general philosophy and its significant features is in order.

Because of the experimental nature, the diffuse geography and the multiplicity of Host types of the network, it is essential that the program be simple and crisp. The program should be divisible into clearly defined functional units with as few interconnecting pathways as possible. This approach will greatly simplify the debugging of the software. Since the network will evolve as we learn more about networks and their uses and constraints, the program must be designed to allow for changes and modifications.

To cope with a wide range of real-time data rates, particular attention must be paid to timing requirements. In addition, since much of the IMP memory is given over to buffer storage (both to and from the local Host and for store and forward), the program must be as compact as possible. Of the 12K of memory, we expect the program will eventually occupy approximately one-half to two-thirds. The network software is outlined in this section and a set of flow diagrams is included in Appendix F.

We feel that the only sensible language in which to write the IMP software is DDP-516 assembly language. This will enable the IMP programs to be as compact and efficient as possible, which is something a higher level language typically subverts. Optimum efficiency is essential here; when a program must deal with low level hardware considerations in real time, a high level language becomes more of a nuisance than a convenience. Although a high level language makes programs more readable and easier to debug, we do not feel we can afford the luxury.

Figure III-14 is a schematic diagram outlining the control logic of the operational program. It has five basic pieces: an initialization routine, interrupt routines, task routines, shared subroutines, and background routines. The program is started at the initialization routine, which first goes through a machine and interface checking routing. It then sets up inputs for all input channels (from Host and phone line Modems) such that, when an input is complete, an interrupt will occur. It also enables the clock interrupt and does all other initialization that is necessary and then turns control over to the background loop.

The routines of the background loop are cycled through repeatedly until an interrupt switches control to some other routine. When all interruptions have been serviced, control is returned to the instruction in the background routine which was about to be executed when the first interrupt occurred.

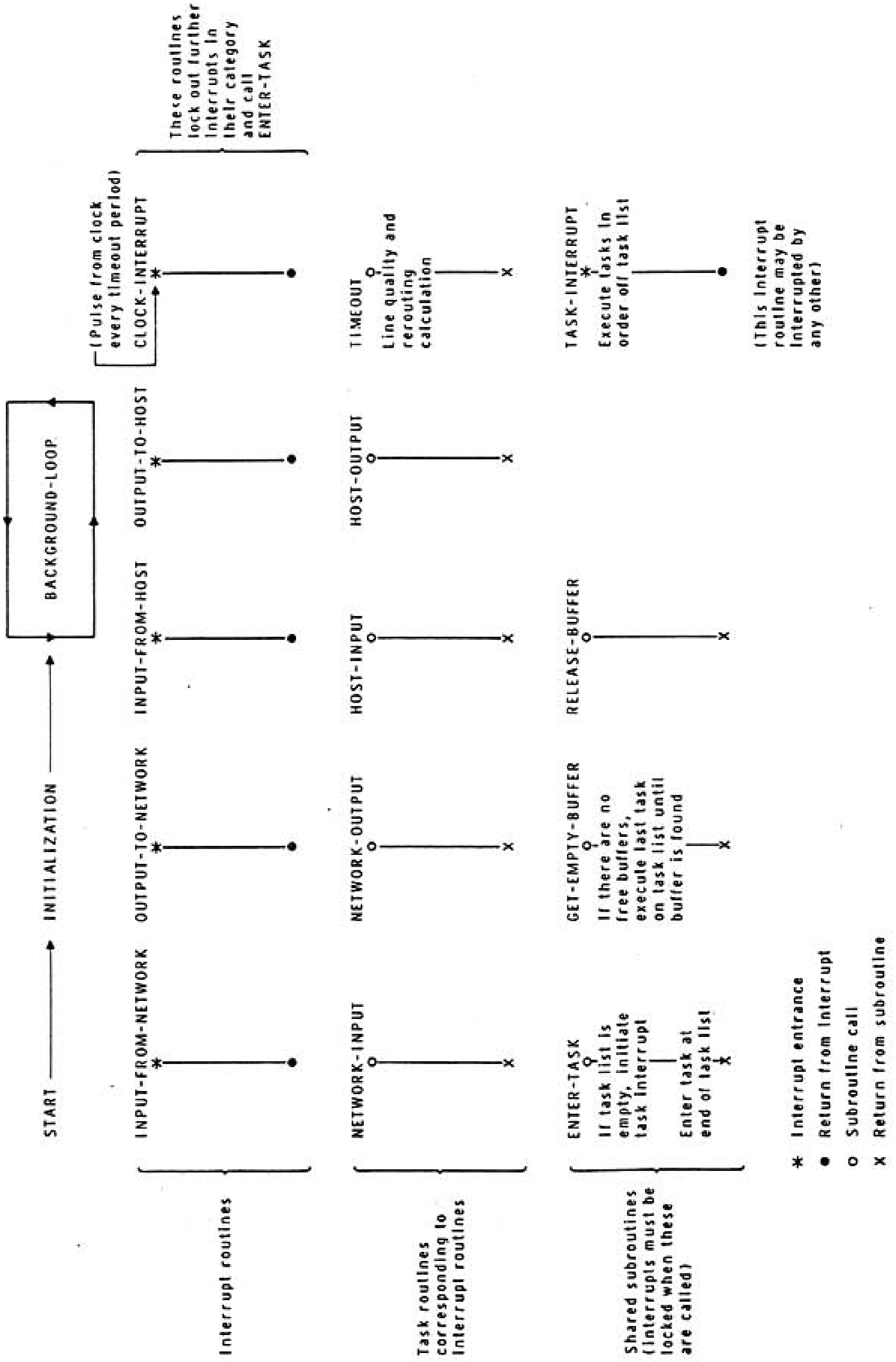

START
INITIALIZATION
BACKGROUND-LOOP
INPUT-FROM-NETWORK
OUTPUT-TO-NETWORK
INPUT-FROM-HOST
OUTPUT-TO-HOST
CLOCK-INTERRUPT
(Pulse from clock every timeout period)
These routines lock out further interrupts in their category and call ENTER-TASK
Interrupt routines
NETWORK-INPUT
NETWORK-OUTPUT
HOST-INPUT
HOST-OUTPUT
TIMEOUT
Line quality and rerouting calculation
Task routines corresponding to interrupt routines
ENTER-TASK
If task list is empty, initiate task interrupt
Enter task at end of task list
GET-EMPTY-BUFFER
If there are no free buffers, execute last task on task list until buffer is found
RELEASE-BUFFER
TASK-INTERRUPT
Execute tasks in order off task list
(This interrupt routine may be interrupted by any other)
Shared subroutines (interrupts must be locked when these are called)
* Interrupt entrance
● Return from interrupt
o Subroutine call
x Return from subroutine

FIG. III-14 IMP PROGRAM CONTROL LOGIC

When an interrupt occurs, a call to the routine associated with that interrupt is executed. This call saves the point of interruption so that control can later be returned to the proper place. The interrupt routine also saves the state of the machine for restoration upon return. An example of an interrupt condition is the completion of the input of a packet from a neighboring IMP. The input hardware calls the interrupt routine, which sets up another input, rearms the interrupt line and designates the received packet for subsequent processing. The input interrupt routines are indicated just below the initialization routine in the diagram. These interrupt routines prohibit calls *of themselves* while they are running by locking out further interrupts of the same kind upon entry to the routines.* Consequently, these routines must be very fast so that interrupts can be re-enabled quickly and not be missed. Most of the time-consuming work is taken out of the interrupt routines by having them merely stack calls to other routines (called task routines) on a task queue which will be executed in what is, in some sense, high priority background time. This allows some time buffering of packet handling if the handling routines take more than real time for a short period.

The question arises as to how the tasks contained in the task queue are ever processed, since the interrupt routines return control to another interrupt routine (if interruption occurred there) or to the background routines when all interrupts have been serviced. This is done as follows: each time a task is entered onto the task list, a check is made to see whether there

*The DDP-516 provides for this with a convenient interrupt selection mask and enable scheme.

are any previous tasks on the queue. If not, a special hardware feature is used for a program-initiated interrupt (called the "task interrupt"), which is set so that, when the "normal" interrupt routine returns to the background loop and re-enables interrupts, the "task interrupt" will take control and allow entries to be processed in the task queue. (The ENTER-TASK and TASK-INTERRUPT routines are shown in the bottom left and the bottom right of Fig. III-14.) When the task list is empty, control is returned to the point of interruption in the background loop. The interrupt routine which executes tasks can be interrupted by any other interrupt routine but will never interrupt itself. Because calls of the task routines are executed sequentially, there is no need to make the task routines reentrant and indeed this is the fundamental reason for queueing tasks. Appendix F includes an example of the use of task and interrupt routines.

There remains a set of routines called the shared subroutines. These are the routines that make entries on the task list, the routines that handle empty buffers, etc. Other interrupts which may call these routines are locked out when these routines are called.

In summary, then, there are really three levels of priority, each corresponding to programs which perform a particular type of function:

1) interrupt routines that interrupt task routines and background routines and even some other interrupt routines;
2) task routines which (in some sense) interrupt the background routines; and
3) background routines.

The interrupt routines for the interfaces are activiated as buffers fill, or are emptied. In general these routines reset pointers, make entries in the task queue for handling filled buffers and releasing emptied ones, and reactivate the interface in question. The clock interrupt routine indexes a higher order clock counter which is maintained in core memory and adds to the task list the task that tests for packet time out. Some of the task routines are: allocating and reclaiming empty buffer storage; handling short buffers with high priority; timing out for IMP-to-IMP acknowledgments and retransmitting (when appropriate); processing end-to-end Requests-For-Next-Message; locating the next buffer to send; identifying incoming messages and placing them on the proper queue for transmittal either to the Host or into the proper output line; transmitting IMP-to-IMP acknowledgments; reassembling messages for the local Host and transmitting Requests-For-Next-Message after reassembly is complete; breaking off destination information from the top of messages from the local Host and fabricating and attaching link identification; and other header information to outgoing packets of a message.

The concept of three priority levels, and the availability of the background loop permits the IMP to perform much more extensive computations on an occasional basis. This is particularly important if the need arises for word-rate jobs on occasional packets. If Host-peculiar programs are required for ASCII conversion, or for other data transformation tasks, such jobs may be accomplished without disrupting the tight timing of the interrupt routines or the task queue. Background programs also include such jobs as transmitting and checking received network test messages and miscellaneous statistics gathering.

1. Summary of IMP program routines

Initialization

Checks hardware of machine and interfaces, sets up initial inputs, enables interrupts, and does other necessary initialization.

Background loop

Set of routines executed cyclicly, in order when not interrupted.

Execute task

Executes entries on task list in order.

Input from network

Answers interrupt, sets up new input from network line, and enters task on task list.

Output to network

Answers interrupt and enters task on task list.

Input from host

Answers interrupt and enters task on task list.

Output to host

Answers interrupt and enters task on task list.

Timeout

Answers interrupt and enters task on task list.

Interrupt Routines

Input from network

Puts acknowledgment on output queue and dispatches* to the input processing routines.

Output from network

Finds next unused buffer, marks it sent and sets up output.

Input from Host

Appends header to buffer, etc., puts buffer on output queue, and sets up new input.

Output to Host

Sets up output to next buffer to Host.

Timeout

Searches output queues for any unacknowledged buffers and reroutes them.

} Task Routines

Enter task

If task list is empty, initiates program interrupt and enters task on task list.

Get empty buffer

Calls Execute task if no empty buffers remain and returns buffer.

Return empty buffer

Rerouting

} Shared Subroutines

*These routines do most of the work of IMP program.

We feel that the program structure just described meets the goals discussed earlier. The program is constructed of functional modules that are logically independent, thus giving them a simplicity that will make their coding, debugging, and understanding easy. Such modularity also enables natural and easy addition and deletion of functional modules.

Recursion (i.e., reentrancy), which is costly in time, is eliminated through use of the task list that also provides a single consistant manner of calling and passing arguments to subroutines. Speed is also attained by moving pointers rather than buffers and by keeping buffers on doubly linked lists for easy insertion and deletion from queues.

While the proposed program structure does not waste space, it is not designed to be as short as possible. We feel it is not worth the additional complexity that results from routines which share short pieces of common code, especially since the routines run on interrupts and interrupt each other. Of course within a routine we will use all of the cleverness at our disposal.

2. Timing and space considerations

In this section we estimate the running time of the crucial routines of the IMP program, review the consequences of these times, and estimate the storage requirement of the IMP program.

Appendix F details the functions of the various IMP program routines. A study of these routines yields our timing estimates. We first consider in detail the running time of the INPUT-FROM-NETWORK interrupt routine (we actually coded sample routines).

The coding requires 40 instructions with an average time of 2.5 µs/instruction. We next estimate quite closely the running time of the NETWORK-INPUT task routine, including the STORE-AND-FORWARD input processing routine which we feel approximates an average path through the NETWORK-INPUT task routine. This we also estimate to be 40 instructions.

We also estimate that the OUTPUT-TO-NETWORK interrupt routine and the NETWORK-OUTPUT routine will each take about 20 instructions.

The time required to handle the Host is under the IMP's control and is also down by a factor of four from the time required to handle the four modem lines and may thus be temporarily discounted; rerouting happens rarely, as it is clocked.

Thus, the bulk of the work may be tabulated:

40 instructions	–	INPUT-FROM-NETWORK
40 instructions	–	NETWORK-INPUT
20 instructions	–	OUTPUT-TO-NETWORK
20 instructions	–	NETWORK-OUTPUT

Since the number of instructions required to pass a packet into an IMP is 80 and the number of instructions required to pass a packet out is 40, we take the average number to handle a packet to be 60 instructions. Adding a factor of one-half to take into account things we have forgotten (overheads of various types, Host routines, and a share of the rerouting time for each packet), we arrive at an estimate of ninety instructions required, on the average, to pass a packet across an IMP boundary.

Using these numbers, Appendix C draws the following conclusions: assuming the RFQ model (i.e., 4 links, 15Kb lines, 344 bit packets, etc.), 14% of the machine time is used. Assuming the RFQ model, but with all 50Kb lines, 43% of the machine time is used.

We finally estimate, based on experience rather than actual coding, that the storage necessary for the main IMP program outlined in Appendix F — the program which does the hard, fast, "necessary" work — will fit in 2000 words of DDP-516 storage. The remainder of the program (the background routines, the special IMP-TO-HOST message routines, etc.) is much less well defined but we estimate that it will occupy somewhere around 4000 words. This leaves about 6000 words for buffers and program expansion.

3. Test programs

Typically, many of these programs are short and simply pump test patterns through the interfaces for observation on an oscilloscope. Programs for loop and inter-computer tests in general will not involve complex error analysis although they will include error detection. The more sophisticated test programs transmit and receive (in loop or inter-computer configuration) random patterns, checking for identity upon receipt. No program means exists for generating errors in the cyclic check mechanism of the hardware, but failure can be introduced by temporarily disabling check character generation in the sending hardware.

4. Utility programs

The DDP-516 comes with an assembler, a primitive editor, a program loader and an octal debugger. Assembly of programs will be done at the test facility on a 516 which will have a high-speed punch. Programs will be composed and edited on BBN's PDP-1d computer under time-sharing and will be punched in ASCII for the 516 assembler. This requires the construction of no additional sophisticated utility programs, allows multiple users access to program composition facilities, and causes no disturbance of the standard DDP-516 assembly and debugging system.

O. Optional Site Arrangements

We have given some consideration to three special sorts of site installations: one with two hosts to be served, one in which the IMP acts as a terminal controller, and one in which the IMP services the Host as a data concentrator. For the site with two hosts, two IMP/HOST hardware interfaces will be required. While the standard interface is modular in nature and two such interfaces can be installed in an IMP, this installation creates a special situation. First of all, either additional priority interrupts will be required or some of the normal priority interrupt channels will have to be reassigned. In either case, some special tailoring of the standard program will be required, at the very least, to enable it to handle the interrupts properly. The 16 channels of the DMC are sufficient to cover this case. However, we feel that generalizing the standard program in such a way as to make it directly suitable for either a one or two host installation is not sensible: the additional required sorting and routing is simply too expensive in terms of time and space to warrant its inclusion in the standard version. On the other hand, the program is amenable to modifications that will enable it to handle the two host situation — but with some degradation of performance.

If a proposed network node does not have a Host computer, it may be useful to put into the IMP those functions of a Host computer that allow users at Teletypes to converse with distant nodes.

To do this, one might first conceptually partition the IMP computer into two parts — one for the IMP network program and one for a program similar to the Host network program which each normal Host has. This partition is easy to make since both programs will run asyncronously on interrupts. Additionally, a Teletype scanner must be attached to the I/O channel for the pseudo-Host network program. This program maintains an input and an output buffer for each Teletype line and gathers characters for the buffers as the scanner collects them. When a buffer is full, it is passed to the IMP network program as a packet. The IMP program which normally deals with the Host interface is now no longer necessary.

This scheme subtracts from the time available for the IMP network program to service store and forward packets. The method does not detract from the space available for buffers, since the pseudo-Host program replaces the IMP Host interface program,and the pseudo-Host program shares buffer storage with the IMP network program.

If there is a Host computer at a network node, it might be feasible to use the IMP as a data concentrator for the Host. In this case, the pseudo-Host program described above is still necessary; instead of passing packets to the IMP network program, the program passes them to the Host. The Host can arrange to process these special packets as, for example, line-at-a-time Teletype input to the standard Host operating system.

Once again, no timing problems occur since the separate IMP programs are run asynchronously on interrupts, but the additional IMP program does subtract from the available space since the IMP/Host interface program cannot be omitted.

We have not investigated these issues in any real detail. There are many other possible, perhaps better, methods of simultaneously using an IMP for a data concentrator or terminal.

CHAPTER IV: IMPLEMENTATION PLAN

This section discusses the key features of our proposed implementation plan, provides a detailed schedule, and indicates the major tasks and major milestones. We first discuss some key features of the plan:

First, we propose to organize a development and test facility at BBN including an additional (fifth) computer. Initially the facility would consist of a production, non-ruggedized, DDP-516 for early use by BBN programmers. Slightly later, this machine would be retrofitted with single-thread interface units and would then become a "test set" for the checkout of the first ruggedized IMP unit, and for the prototype trial of hardware and software for a single IMP (loop test) and a two IMP connection. Finally, the facility would be used for the system test of the remaining three IMP units prior to shipping to the field. In essence, we feel that five computers must be purchased to obtain four in the field. There are several possible eventual dispositions for this machine: it might be used as a test set in connection with the production of the larger net or, by retrofitting additional interfaces, it might become a *spare* or it might easily allow ARPA to outfit a complete fifth IMP site.

Second, we believe that a *real prototype phase* of development must be undertaken. We are not willing to produce a design, build four copies, and install them in the field. We wish to build a prototype, test that prototype, have time in the schedule for limited hardware modifications, and then build the final units. This opinion is based on many factors, and includes:

1) The combination of relating new 50 KB modems and two new interface designs with the normal difficulties of computer communication makes the system too complex to trust an untested design;

2) The software system is sufficiently complex and involves sufficient timing problems that a trial is very important before freezing the *hardware* design.

This need for a real prototype phase does, however, have an unfortunate total effect on the schedule. BBN does not believe that nine months is a realistic period for putting four IMPs into the field. We have chosen to offer a schedule variant in the belief that the described method is the *fastest* way to obtain *reliably operating* IMPs in the field. We propose to install the first two units in the field by the nine-month point but effective four-IMP tests would not commence until the eleventh month. In order to include the three month "available" period described in the RFQ, our total proposed contract period is 15 months rather than 12.

Third, we offer the novel idea of using the extra IMP machine to simulate a Host, rather than building a completely extra Host interface to one of BBN's local time-shared systems. We carefully considered using our 940 system as a Host; this initially appeared very attractive, since the SRI machine is a 940, and we had hoped the complete (two-piece) Host-IMP interface could be built at BBN for testing and then moved to SRI. Upon closer examination, this proved to be wishful thinking, since the entry point to the two 940 systems is completely different. Further, even a 940 test program would not be transferable, since the two executives are different. It then occurred to us that an

IMP is an *ideal* Host simulator, since, back-to-back, the interfaces match perfectly with no hardware problems. Further, using an IMP avoids a completely wasted software effort and expense that is not inconsiderable.

A. Schedule

Figure IV-1 shows the schedule proposed by BBN to accomplish the requirements of the contract. Related tasks are connected by arrows, major milestones are indicated, and delivery dates for formal reports are shown. The dates shown in Figure IV-1 are the result of a careful analysis of the tasks to be performed and the lead-time for delivery of the hardware to BBN.

This schedule intermixes work to be performed at BBN with work to be performed by Honeywell. In general, BBN has already designed the special interfaces (see Section III-F and Appendix E) but Honeywell will take responsibility for implementation of the interfaces and integration with the DDP-516 computers. The "system development and test" as well as all software efforts will take place at BBN, but Honeywell will provide design engineering assistance, on location at BBN, during the early phases of IMP integration with modems, and this support is integrated into our cost proposal. Similarly, Honeywell will assist, both with design engineering personnel and field engineering personnel, in the field installation phase on the West Coast.

In order to minimize the time required to complete the IMP deliveries, the various tasks will be performed concurrently to the greatest extent possible. Design of production hardware will

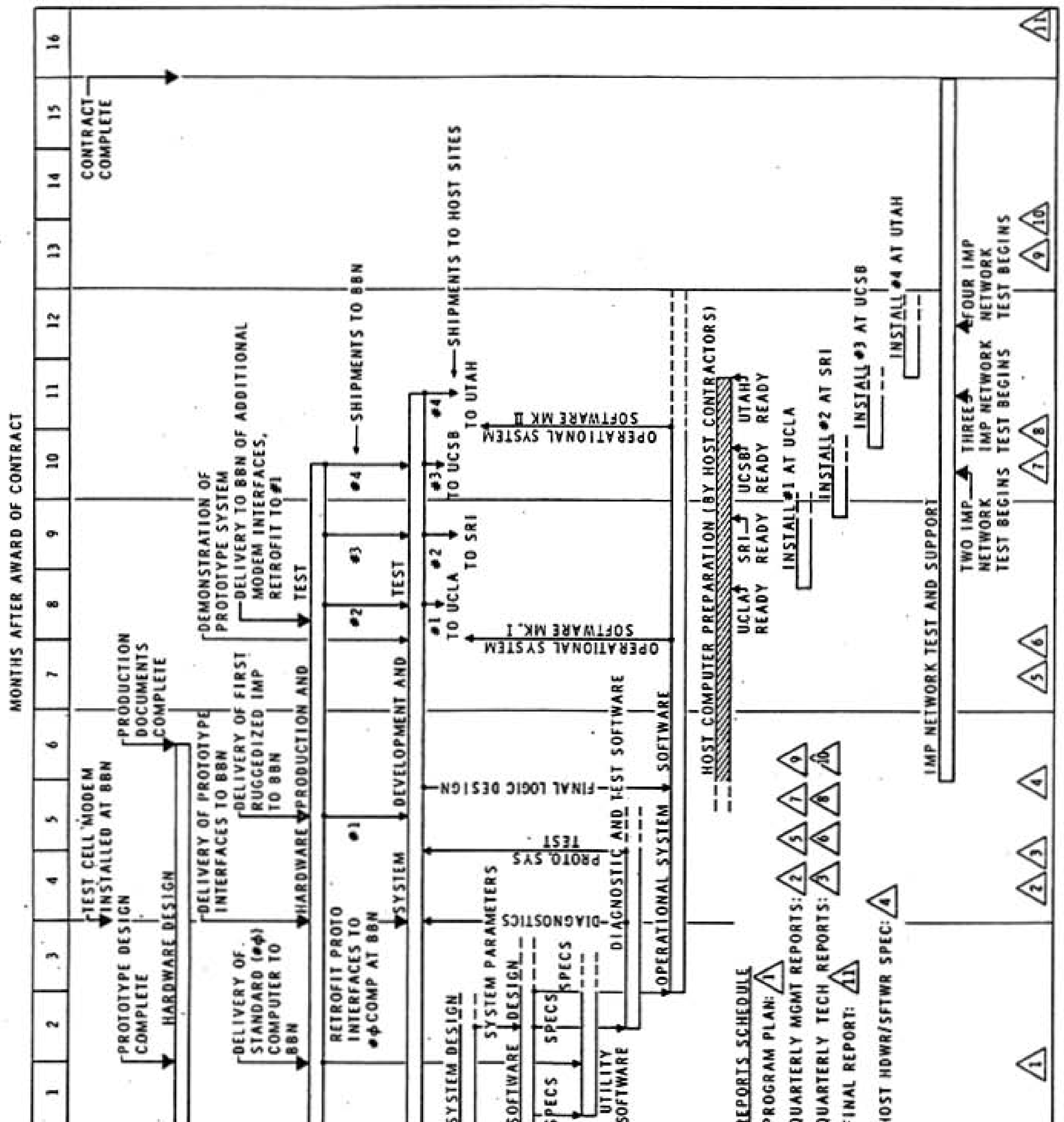


FIG. IV-1 IMP PROGRAM SCHEDULE.

begin on Day-One of the contract, since most of the hardware design parameters are known and are not likely to change. This task will receive inputs continuously from the Hardware Design and System Development Tasks; so that when the logic design freeze occurs at the end of the fifth month, only two weeks will be required to complete the final production documents — for example, revised wire-tables. In order to accelerate the construction of software, BBN proposed to take delivery of a standard DDP-516 computer at the earliest possible date (no later than thirty days after contract award). This computer will then be retrofitted with prototype interface hardware as soon as possible, but the computer will be available during the interim for initial programming. The various software tasks will be overlapped to minimize the time required to produce the first operational system. The proposed schedule calls for utility (program preparation) software and diagnostic and test routines to become available as required by the hardware schedule. The #1 IMP will be delivered within 4-1/2 months after contract award and will be used in conjunction with the first prototype for IMP-to-IMP and IMP-to-"simulated Host" tests leading to a prototype system demonstration at the end of the seventh month.

Our schedule shows the three remaining machines arriving at BBN starting in the eighth month. This estimate is actually more conservative than Honeywell's figures (see Appendix B). We feel that extra time built into the schedule at this point will permit either the greater amounts of redesign that might be required, or will permit some slippage in earlier dates, without affecting delivery to the sites. If everything goes really well, we would be in the happy position of bettering the site delivery schedule, and more closely meeting the RFQ dates — but we do not wish to promise those dates.

As soon as the #2 IMP is delivered to the BBN test facility (middle of the eighth month) the #1 machine will be installed at UCLA. This will allow BBN one month to verify proper operation and to identify unexpected interfacing and installation problems before the #2 IMP is shipped from BBN.

In summary, BBN has made every effort to develop a schedule, using realistic estimates, that accomplishes the contract requirements in the least time consistent with reasonable confidence that the dates proposed can be met.

B. Work Scope

In accordance with Section II of the request for quotation, BBN proposes to design the full 19-node network and to install the four-node test network; and to have full systems responsibility for the project.

We will "design the COMMUNICATION SUBNET." In fact, this proposal contains a relatively detailed design. We would expect modifications to the design might take place as a result of discussions with ARPA and/or ARPA consultants, and we would anticipate completing the design during early phases of the contract period.

We will construct a prototype IMP, including IMP-carrier and Host-IMP interfaces. We will write, checkout and demonstrate the communication programs operating in this prototype. We will carry out closed-loop communication tests.

We will construct and install four IMPs and associated interfaces at SRI, UCLA, UCSB, and Univ. of Utah. We will demonstrate operation of the IMP subnet and, assuming Host readiness, will participate in network tests.

We will provide system documentation for all system components.

C. Task Breakdown

Some of the key tasks and milestones are discussed in this section.

1. IMP system design and analysis

This task is concerned with the need to assure that the IMP system as proposed will achieve the desired performance effectively. Simulation studies of both the test network and the full network will allow us to verify that performance specifications will be met and to study various system parameters and doctrines (such as routing doctrines). In addition, system operation will be analyzed to establish specifications for prototype and production system tests.

2. Prototype hardware design and fabrication

The prototype hardware design task will consist of verification and refinement of the hardware interfaces proposed herein and implementation using standard Honeywell components. The former will be performed by Honeywell personnel with verification of the final circuit design by BBN. The prototype interface will be fabricated by Honeywell and given unit checkout before shipment to BBN. To minimize the time required to complete this task, standard components will be used throughout and minimum effort will be devoted to design economies. The first set of prototype interfaces will be packaged with cables designed to permit it to be retrofitted to the prototype DDP-516 computer. The second set of prototype interfaces will be integrated with its DDP-516 at Honeywell. In order to facilitate preparation and check-out of software, the prototype DDP-516 will include a paper tape punch.

3. Prototype hardware test

This task will be performed at the BBN test facility. Its objective will be to assure that the prototypes are ready for system tests by making careful and systematic tests of all the hardware, both standard and special.

4. Prototype system test and demonstration

One phase of the System Development effort shown in Fig. IV-1 will be the test and demonstration of the prototype system using the two prototype IMPs. The first objective of this task will be to verify that the hardware and software have been properly integrated and, in particular, that the specially designed interfaces meet the design specifications. The second objective will be to demonstrate the overall functioning of an IMP to simulate first a Host and then another, IMP, so that the demonstration IMP can be presented with each of the various situations it will encounter in the actual network.

5. Production hardware design and fabrication

This task will be performed by Honeywell personnel under the overall supervision of BBN. As described in Section A above, the task will run concurrently with the prototype design fabrication and test. Since much of the fabrication effort is independent of last minute changes in logic design and also includes long lead time components, it will be possible to keep production design nearly current. Figure IV-1 shows production design complete two weeks after the logic design freeze and delivery of the #2 IMP to BBN two months later.

6. Production hardware check-out

This task will be substantially the same as the Prototype Hardware Test described in Section C.2 above. Of course, particular emphasis will be placed on the testing of the units which have been redesigned and more attention will be placed on testing to verify field maintenance features of the hardware. The objective of the task will be to assure that the hardware is ready for system test.

7. Production system test

As production IMPs are received, each one will be tested and proper operation demonstrated using the procedures developed under the previous task. The #1 IMP will be demonstrated using the first prototype. Subsequently, the #2 IMP will be operated with the prototype to assure that they are compatible. This process will be repeated with each subsequent production IMP and none will be shipped to a Host site until proper operation has been fully demonstrated. Careful attention, will be given to demonstration of fault detection, isolation and correction to assure that field support procedures are complete and effective.

8. Host site preparation

Providing each Host site with complete site preparation specifications will enable the Host facilities to design and implement the hardware and software necessary to interface with the IMP. BBN will then issue a complete specification at the end of the fifth month of the contract. In order to be ready for the prototype IMP, it will be necessary for UCLA to complete its preparation about two and a half months later. To ease the impact of this short response time, every effort will be made to give UCLA useful data covering such things as AC power requirements, temperature and humidity specifications, physical dimensions of the IMP and modem, and cable length limits prior to the formal release of the specification so that some work can be accomplished ahead of time. The remaining sites will have about four months after formal release.

9. Production system installation

This task is to install production IMPs at the Host sites, to check out each IMP with its Host using the procedures developed during prototype and production system tests, and then to check out the IMP-to-network interface. In effect, the task will be repeated at each Host site. As indicated in Fig. IV-1, BBN proposes to begin installation with the #1 IMP at UCLA during the eighth month of the contract. This will allow enough time to use the experience gained during installation and checkout to refine the procedure used for IMP's #2, 3 and 4, since their installation will begin one month later. (UCLA is proposed for the initial installation because support will be forthcoming from BBN's nearby facility in Van Nuys, California; but this is a negotiable detail.)

10. Utility software

This task will be to provide the software necessary for rapid and efficient production of test and operational software. To speed up production of software, BBN proposes to use an in-house time-shared computer for editing of IMP software. The time-shared computer will produce punched paper tape output from edited programs which can then be assembled using the first prototype DDP-516.

11. Diagnostic and test software

This task consists of the production and checkout of required diagnostic and test software including informal documentation describing its use and including sample test input and output data. Diagnostic software will be ready at the end of the third month when the first prototype hardware is ready.

12. Operational system software

This task will provide system software, that is to say the software resident in each IMP during normal operation. BBN proposes to have two releases of this software. The first version will be based on specifications provided by the software design task, and will use the parameters resulting from the system analysis task. It will make use of results obtained during system development and will be ready at the end of seven months. The next version of this software will be provided for use in supporting the operational network by the end of the tenth month of the contract.

13. Production hardware documentation

This task will run concurrently with production hardware fabrication. During hardware and system tests, careful records will be maintained so that necessary changes and corrections are made. Honeywell will deliver complete and formal hardware documentation with IMP #2.

14. Software documentation

The objective of this task will be to assure that, at the completion of the contract, operational software will be fully and formally documented and will include the following:

1) Utility Programs. In addition to an annotated symbolic listing and to symbolic and binary tapes of the assembler, the standard DDP-516 programming manual will be expanded to include a description of the special hardware.

2) Diagnostic and Test Programs. Formal documentation will be provided for a complete diagnostic and test package suitable for support of the production IMPs at the Host site. The

package will include an annotated symbolic listing, symbolic and binary tapes, an instruction manual describing how the software is used for fault isolation, and appropriate test data tapes.

3) Operational Software System. Documentation will include an annotated symbolic listing and symbolic and binary tapes. A manual, provided to describe fully the system operation, will include an overview of the system, discussions of hardware logic timing considerations, and a detailed description of the formats used. It will also include a narrative description of key routines, and an annotated symbolic listing. It will be assumed in writing the manual that the reader has no prior knowledge of the IMP. The final Host software and hardware specifications will be included.

15. IMP network support

The RFQ was not specific as to what tasks, specifically, might be performed during the three months after installation of the four node net. We suggest the following activities as being suitable for that time period.

1) Design and construction of a network test and demonstration plan. This plan will be based on the experience gained in testing the individual IMPs and the IMP System Design and Analysis task and will include observation and measurements during network operation.

2) Conduct of the network test plan. The objective of this sub-task will be to prepare a technical report, based on an analysis of the test results, that describes quantitatively the actual performance parameters achieved. The

report will include a projection of the parameters expected from the full-scale network, the identification of problem areas, and recommended corrective action.

CHAPTER V: TECHNICAL TEAM

We feel that one of the very best arguments for the award of this contract to BBN is the character and strength of the proposed technical team. All of the proposed senior team members have directly applicable experience with the design, implementation and field installation of real-time digital computer systems. In addition, BBN's overall involvement in computer systems places a large additional pool of seasoned computer system experts at the disposal of the project. The project will be centered in the Information Science and Technology Division of BBN; this division is supervised by J. Elkind and J. Swets.

The Project Manager will be Frank E. Heart, a highly experienced computer systems engineer and manager. ① At BBN, Mr. Heart has been involved with the Hospital Computer Project, a major Navy computer system project, BBN's TELCOMP Service, and efforts to utilize computers in education. In his prior work at Lincoln Lab, Mr. Heart was directly responsible for the implementation of several real-time computer systems, including the HAYSTACK pointing system, the LET (a mobile communication terminal), the developmental LASA system, and the Westford pointing system. In each case, these systems were successfully used in the field for many years.

The Associate Project Manager will be Hawley K. Rising, another very experienced senior engineer. At BBN, Mr. Rising has been in charge of a project to assist the Navy in obtaining a major signal processing/data management system. In his prior work at MITRE, Mr. Rising led several large groups in work varying from basic hardware development to computer systems implementation and programming. Notably, Mr. Rising supervised work on building the PHOENIX Time Sharing Computer, and was a major participant in the implementation of the System Design Lab containing the STRETCH.

The senior hardware responsibility will be held by Severo Ornstein, who will also handle technical liaison with Honeywell. Mr. Ornstein, a computer designer as well as a good programmer, is a dynamic and flexible engineer. (The interface designs in this proposal were provided by Mr. Ornstein.) At BBN Mr. Ornstein has been involved with the use of computers in education and with the hardware aspects of special peripheral gear for large time-sharing systems. In his prior work at Lincoln Lab, MIT, and Washington University in St. Louis, Mr. Ornstein interchangeably worked as a programmer and a computer logic designer. Mr. Ornstein worked with Wes Clark for many years and participated in a major way in the LINC Program and the development of MACROMODULES.

The senior software responsibility will be carried by William Crowther and David Walden.

> Mr. Crowther, a very recent BBN employee, is an experienced, prolific, and creative real-time system designer and programmer. His primary experience has been at Lincoln Laboratory, where he was fully responsible for the design and implementation of the program in the UNIVAC 1218 for the LET (Lincoln Experimental Terminal). He also played a key role in many other major real-time system programs.
>
> Dave Walden is also an experienced and prolific real-time systems programmer with, in addition, a broad knowledge of current software technology in the area of languages, utility systems, etc. At BBN Mr. Walden has worked with the

Hospital Computer System, the design of data management systems, and compiler design. In his prior experience at Lincoln Lab, Mr. Walden worked ~~with Mr. Crowther~~ on the LET Project, and participated in several other major real-time programming projects.

Primary theoretical and systems design responsibility will be carried by Robert Kahn. Dr. Kahn will also assist in technical liaison with the common carrier, the host organizations, and the client. Dr. Kahn is an expert in the area of information theory, random processes and queueing theory. At BBN he has been studying computer network problems and investigating the statistical structure of natural and artificial languages. His prior experience was at Bell Telephone Laboratories.

Primary responsibility for organization, scheduling, and planning activities will be carried by Robert Jacobson. At BBN, Bob has managed the TELCOMP activity and more recently has been working with Mr. Rising on a major Navy computer system project. Mr. Jacobson's prior experience was at Raytheon, where he participated in several major military system projects, including a mobile medium range ballistic missile command and control system design study, several ICBM Penetration Aids projects and the Apollo Guidance Computer Project.

Others who may spend significant amounts of time on the project include:

HARDWARE	SOFTWARE	DOCUMENTATION
J. Henry	A. McKenzie	M. Fein
R. Gagne	B. Cosell	
R. Kokoska	F. Webb	

In addition, another group of senior computer systems people will be available for consultation and limited assistance. These include: Dr. J. Elkind, Dr. D. Bobrow, T. Strollo, J. Barnaby, Dr. R. Alter, Dr. J. Markowitz.

Finally, we plan some utilization of personnel from our Van Nuys office to assist in the field installation and maintenance phase of the project; this assistance will include Roland Bryan as the Van Nuys team leader, and the following other Van Nuys personnel will be available: Ralph Graeber, Ralph Athearn, and Ron Freeman.

Resumes of all the listed personnel are included in Appendix J.

APPENDIX A: BBN EXPERIENCE AND FACILITIES

1.0 EXPERIENCE

Bolt Beranek and Newman Inc., organized in 1948, is a science-based firm specializing in consultation, research, and development in the areas of physical science and technology, oceanology, information science and technology, architectural acoustics and noise control, applied chemistry, and education and training. BBN has served as consultants to industry, educational institutions, and several branches of the government, including the National Aeronautics and Space Administration, the Department of Defense, and the U. S. Public Health Service. The enclosed BBN 1967 Annual Report provides further information.

In the computer field, BBN is currently pursuing work in the development of time-sharing systems and is active in a wide variety of projects directed toward the achievement of "natural" communication with computers. BBN is working on the application of time-shared, on-line, remote-access computer systems for the processing of bio-medical information as well as for the development of computer-aided instructional systems for the educational field. In addition, BBN provides a commercial on-line time-shared computation service, called TELCOMP. The following sections present BBN's experience in designing, developing, and implementing real-time and time-sharing computer systems and specialized software packages.

1.1 System Design and Implementation

Through our TELCOMP system, BBN has gained substantial experience in designing and implementing an on-line, time-shared service for scientists and engineers in the research community. This service, commercially available since September 1965, currently provides a user-oriented interpretive language based on the RAND Corporation's JOSS. Service is provided by BBN-designed systems installed in the Boston, New York, and London areas. Because of the highly interactive conversational language (TELCOMP), researchers with no programming experience are able to solve successfully statistical and computational problems. The hardware is based on a PDP-7 or PDP-9 modified to BBN specifications and a PDP-8 to manage the messages to and from the modems rather like the IMP-Host relationship. The system also includes drum and mass-memory interfaces designed by BBN.

In August 1966, we acquired an SDS-940 system which is used for basic computer science research under ARPA support. To date we have implemented and improved a number of interactive systems including an expanded LISP, a new version of CAL, an improved version of FORTRAN II, and a very talented text editor (TECO). These, of course, are all available concurrent with the standard SDS-940 subsystems (ARPAS, QED, DDT, SNOBOL, etc.,) under the control of our own executive which is described in TSS 1.85, BBN, January, 1967.

BBN has implemented a real-time hybrid I/O interface to the SDS-940. The interface, called a Hybrid Processor, can accurately time real-time data transfers between real-word devices and the core memory of the SDS-940 without CPU intervention. BBN uses the Hybrid Processor to control an on-line video device, to sample and generate speech waveforms at up to 33 kHz, to interface to a CRT display, and to control a hybrid analog computer (Applied Dynamics 4). The Hybrid Processor is interfaced to the SDS-940 through the SDS Data Multiplexor Channel. We have also designed a scheduler which will handle a number of synchronous and asynchronous real-time processes. This scheduler will permit a process to get on the system only if the system can guarantee service to the real-time demands of this and all other processes.

Currently BBN is installing a PDP-10 computer system for continued research work under ARPA contract support.

Our initial experience in developing time-sharing software and hardware systems was gained through the National Institutes of Health sponsored Hospital Computer Project, conducted in conjunction with the Massachusetts General Hospital. As a preliminary step to this project, BBN developed one of the earliest time-sharing systems; a three-terminal system using a PDP-1b was first publicly demonstrated in September 1962. Based on this original prototype, BBN developed a system which can handle 64 active terminals. The Hospital Computer System has provided trial service operation in the Massachusetts General Hospital in the areas of admission-discharge census, laboratory data handling, and general-purpose file handling and computational capabilities for clinicians and medical researchers.

BBN is currently designing and installing a computer system for the Pacific Coast Stock Exchange which will automate the handling of securities orders. Called COMEX, the system makes use of an existing communications network. COMEX will accept orders from member firms, compute the price, log the order and return a confirmation in less than a minute. At the same time, the system will continuously monitor both the New York and Pacific Coast Stock Exchange prices.

1.2 Software Systems

BBN has broad interest and capabilities in the field of computer sciences. Research and development projects have involved many subjects: advanced computer organization and design, automatic programming, pattern recognition, computer problem solving, natural language systems, computer-assisted instruction, time-sharing, query systems, speech recognition and analysis, information retrieval, and man/machine interaction.

One of the major areas in which BBN has made a major contribution to the computer state-of-the-art has been in *interactive* handling of data files. One large software package, developed under the Hospital Computer Project, is the Information Storage and Retrieval (ISR) System. During the past two years, the ISR System has been extensively used to manipulate medical records, and physiological and drug information data, as well as in a variety of other data-handling applications. The ISR System is used to gather statistics, examine exceptions to patterns, catalogue various types of information, and to perform other data-handling operations. At present, the System is under consideration for use in the preparation of analyses of experimental data for a new-drug FDA application.

The ISR System allows individual hospital users to create and manipulate private data files. Through a user-oriented, conversational language, users construct data files by describing the data structure and format of a file; each user defines the structural skeleton of his file and specifies the format and syntax of data to be assimilated. The user can make selective retrievals of data from the file through the retrieval program. According to the user's specifications, this program can list a whole file or a subset of a file, do arithmetic and/or print cross tabulation matrices based on either a whole file or a user-specified subset of a file. Another set of software packages handles statistics and other special-purpose computational data analysis operations.

RISCOMP, derived from the RAND Corporation's JOSS, is a general language developed by BBN to program a variety of individual applications ranging from statistically analyzing data to automated inputting of text string files. RISCOMP is a user-oriented language that solves problems in the problems' terms rather than in the computer's terms. The RISCOMP vocabulary is a combination of English and algebra which is both independent of the physical computer and familiar to people having no previous exposure to computers.

1.3 Graphic Communications Devices

BBN has invented or developed several graphic communication devices, including: the Teleputer System, a general-purpose remote graphic terminal that can communicate with a computer over telephone lines; the Datacoder, a medium-resolution graphic-input device combined with a computer-controlled slide-projection system; the Grafacon Rho-Theta Transducer, a high-resolution graphic-input device using an arm and stylus combination together with an analog-to-digital converter; a composite display apparatus for high-speed display of a number of preselected stored images; the Grafacon Model 1010A, a two-dimensional, digital, graphic-input device that was developed by the Data Equipment Division of Bolt Beranek and Newman Inc. and is based on the RAND tablet.

BBN has also built special-purpose graphic systems. For example, we are currently working on a /360 compatible system for the graphic arts department of a large industrial firm.

BBN is currently developing an elaborate, high-speed, display processor for the SDS-940 computer system. This processor will utilize a monitorscope, light pen, Grafacon tablet, pushbuttons, and keyboards. This graphic display system will provide for time-sharing of separate display consoles and will give the user a high-level language for working with the display. This general-purpose hardware-software system, presently in the development stage, will allow the user to manipulate both mathematical and texual data.

2.0 FACILITIES

The IMP Project will be conducted at BBN's principal facility in Cambridge, Massachusetts. Ample floor space is available for all IMP-related activities and BBN does not contemplate the need for any additional facilities. Four fully equipped computer rooms are currently being used for research activities. A specific area will be assigned to the IMP Project for hardware checkout and test.

In addition, an IMP Project Office will be established at BBN's Van Nuys facility to provide field support during installation of the four IMP networks. The following pages describe BBN's facilities more generally.

APPENDIX B: QUALIFICATIONS OF HONEYWELL/COMPUTER CONTROL DIVISION

The Computer Control Division, "3C," formed in 1953, as the Computer Control Company, was acquired by Honeywell in 1966. 3C manufactures a line of general purpose computers as well as logic modules, memories and digital test equipment. The rationale behind the selection of the DDP-516 by BBN is given in Appendix D. As a result of this comparison of computers, including informal contacts with existing DDP-516 users, and discussions with 3C design, engineering and management personnel, BBN has concluded that 3C is fully qualified to perform the tasks assigned in this proposal. Both BBN and 3C intend that the role for 3C in the IMP Program be more than that of a vendor. As indicated by the attached letter to BBN from Mr. Rothrock, Regional Marketing Manager for 3C in New England, 3C will play an active part in the design, development and installation phases. Thereafter, qualified field service personnel and ample spares will be available to provide continuing maintenance of the IMP's hardware. Of the nineteen Host sites listed in the RFQ only two are more than one hundred miles from a 3C field service office.

3C proposes to appoint Mr. Lawrence Prager as IMP Project Engineer with direct responsibility for all phases of 3C's participation. He has been selected because of his substantial prior experience as project engineer for data communications system.

Other systems engineering support as required will be provided by the Information Systems Department headed by Mr. C.B. Newport.

The following documents from 3C are attached to this appendix:

1) Letter to BBN from Mr. James Rothrock.

2) Proposal No. 82516-54 from Honeywell to BBN including Attachment C-IMP Processor Description. NOTE: Attachments A-Price Schedule and B-Maintenance Policy are included with BBN's Cost Quotation.

3) Résumés of key Honeywell personnel.

4) μ-Comp DDP-516 General Purpose I/C Digital Computer.

5) Ruggedized DDP-516 General Purpose Computer.

6) DDP-516 Test and Maintenance Routines.

7) Honeywell documents describing quality assurance and reliability, and documentation.

H·O N E Y W E L L

I N C .

August 29, 1968

Bolt, Beranek & Newman Inc.
50 Moulton Street
Cambridge, Massachusetts

Attention: Mr. Frank Heart

Gentlemen:

The intent of this letter is to confirm to you our verbal agreement to supply support services in the event that Bolt, Beranek & Newman is successful in obtaining the contract to supply the Interface Message Processors for the Advanced Research Projects Agency (ARPA) computer network.

Specifically, Honeywell looks forward to working closely with you as a digital partner and supplying both the hardware required for this contract and the necessary field service and system engineering support required by you, and as outlined in our proposal to you. We appreciate the opportunity to respond to your requirements and wish you success in this venture.

Very truly yours,

HONEYWELL INC.
Computer Control Division

James C. Rothrock

James C. Rothrock,
Regional Marketing Manager

/mm

COMPUTER CONTROL DIVISION 275 WYMAN STREET
WALTHAM, MASSACHUSETTS 02154 893-2610 CODE 617 TELEX: 92-3435

H O N E Y W E L L

I N C .

September 4, 1968
Proposal #: 82516-54

Bolt, Beranek & Newman Inc.
50 Moulton Street
Cambridge, Massachusetts

Attention: Mr. Frank Heart

Gentlemen:

Honeywell Inc., Computer Control Division is pleased to submit this proposal for a DDP-516 computer system to be utilized as an Interface Message Processor (IMP), and the support services you have requested.

This proposal is for five (5) systems consisting of one (1) prototype IMP and four (4) production IMP's. The four production units will be ruggedized and EMI protected. The prototype, which will not be ruggedized or EMI protected will be partially delivered soon after contract award and later retrofitted to a full prototype system.

Honeywell proposes to deliver as a prototype, a standard DDP-516 system with 12K memory, high speed reader and punch and an ASR-35 teletype, 30 days ARO. This system will be retrofitted three (3) months ARO to include: One (1) Full Duplex Modem Interface, one (1) Full Duplex Host-IMP Interface, one (1) Special Real Time Clock, eight (8) Priority Interrupt Lines, one (1) Direct Multiplex Controller, one (1) 24 position Light Register, and one (1) Special Interrupt Flip-Flop. This retrofit will complete the prototype system.

The four production systems will be ruggedized and EMI protected DDP-516 systems with 12K memory, high speed reader and ASR-33 teletype. These systems will also include: three (3) Full Duplex Modem Interfaces, one (1) Full Duplex Host-IMP Interface, one (1) Special Real Time Clock, sixteen (16) Priority Interrupt Lines, one (1) Direct Multiplex controller, one (1) 24 position Light Register, and one (1) Special Interrupt Flip-Flop. Production System #1 will be delivered 4-1/2 months ARO with one (1) Full Duplex Modem Interface. This system will be retrofitted with two additional Full Duplex Modem Interfaces 6 months ARO bringing it up to full production level. The three subsequent production systems can be delivered at 6-1/2, 7 and 7-1/2 months ARO.

Attachment A contains the purchase price and on-call monthly maintenance rates of all the components required to make up the prototype system and the four (4) production units.

Attachment B contains a description of our "On-Call Monthly Maintenance" Policy. We have reviewed the 19 site locations of the ARPA network relative to our service points to determine where additional mileage would occur. The note in Attachment B explains the results of this review.

In response to your request for System Engineering support in the field, it should be explained that a project engineer at Honeywell Computer Control Division is at least a level E3 and more often a level E4. Honeywell will meet your requirements on a contract basis for 120 days of Engineering support in the field at the rates shown in Attachment A.

Initial installation for each system will be accomplished free of charge at Bolt, Beranek & Newman in Cambridge by Honeywell if a maintenance contract becomes effective on delivery date. Packing for shipment to the ARPA site and reinstallation at the ARPA site will be billed at the non-contract maintenance rates shown in Attachment B.

Quantity discounts on these systems will be in accordance with the non-OEM discount policy or the OEM agreement which have both previously been supplied to you.

Attachment C contains the specifications of the special system options. Enclosed are the cost breakdowns of these options.

A Comprehensive DDP-516 Price Schedule is also enclosed. This will be useful in the event you need pricing information on DDP-516 options other than those described in Attachment A. It also indicates which of the options are discountable.

Our prices are F.O.B. Framingham, Massachusetts and do not include any Federal, State or Local taxes. Our terms are net thirty (30) days.

This proposal will remain valid for 120 days.

If you have additional questions, please feel free to contact me or Jim Campbell at the Waltham Sales Office.

Very truly yours,

HONEYWELL INC.
Computer Control Division

[illegible]

Edward L. Keough,
Account Representative

ELK/mm
Enclosures

IMP PROCESSOR DESCRIPTION

The IMP processor is built around a DDP-516 general purpose computer with 12K words of 16 bit memory. Memory may be expanded to 16K, 24K or 32K words. A Host-IMP interface connected to a Direct Multiplex Control Unit (DMC) provides simultaneous communication with a Host computer. Up to six full duplex single line communication line controllers may be installed on each IMP to link IMP's together via a 50 KC synchronous communication lines. Data passes between memory and the single line controller by means of the DMC. A pair of DMC subchannels is provided with the Host-IMP interface and each of the six single line controllers. This allows a total of 14 data communication I/O transfer as well as internal computer processing to proceed at the same time. A Real Time Clock is provided for use by the program for message control and accounting. Sixteen priority interrupt lines are provided. A pair of interrupts is associated with the Host-IMP interface and with each of the single line controllers. One interrupt is associated with the Real Time Clock and the last is used to respond to a program OCP command.

An ASR-33 unit is provided primarily to load and print diagnostic maintenance functions. It is also handy for use as an IMP operator tool for logging of events and keyboard entry of control parameters to the IMP processor without stopping the computer. A status panel is provided with up to 24 lights which indicate status of the various IMP communication channels. This panel in addition to the standard DDP-516 Operator Console is used by the IMP operator to control the system.

EQUIPMENT LIST

A complete list of standard DDP-516 and specially developed equipment of a fully equipped IMP is as follows:

QTY	MODEL NO.	DESCRIPTION
1	516-03	DDP-516 General Purpose Computer with 12,288 words of 16-bit memory.
1	516-20	Direct Multiplex Control Unit.
1	516-25	Group of four (4) Priority Interrupt lines.
3	516-25-1	Additional group of four (4) Priority Interrupt lines.
1	516-53	ASR-33 Teletype Unit.
1	N/A	Host-IMP Interface 16-bit, Full duplex serial to parallel and parallel to serial operation (includes two (2) DMC sub-channels)
6	N/A	Single Line Controller for Full duplex 50 KC communication line (includes two (2) DMC sub-channels for 16-bit I/O, 24-bit polynomial checksum and EIA model interface.
1	N/A	Real Time Clock, 20 μs count, 16-bit programmable.
1	N/A	Special Display Panel with 24 lights.
1	516-50	Paper tape reader, 300 cps.

HOST-IMP INTERFACE

The Host-IMP interface is a bidirectional data channel which allows simultaneous data transfer between the Host computer and the IMP processor. The unit consists of a parallel to serial output section with a DMC subchannel and a serial to parallel input section with another independent DMC subchannel. The maximum transfer rate in both directions simultaneously with alternating DMC subchannel actions would give a bit rate of 2 megacycles in each direction. This would saturate the IMP and preclude communication line operation. This can be prevented by assigning the Host-IMP interface the lowest DMC priority and the effective maximum rate will then depend upon single line controller activity and will fall between one and two megacycles. The actual rate will depend upon IMP activity and the capability of the attached Host and its matching interface.

Host to IMP Program Controls

1. Interrupt on Host ready or DMC end of range or end of transmission.
2. OCP clear interrupt.
3. OCP accept (IMP ready).
4. SKS sense DMC end of range.
5. SKS sense end of transmission

IMP to Host Program Control

1. Interrupt on DMC end of range
2. OCP clear interrupt.
3. OCP request to send to Host
4. OCP end of transmission

Common Controls

1. OCP interface ON
2. OCP interface OFF
3. SKS sense host ready

Interface cable signals

A. To IMP

1. Host Transmission accept
2. Data bit line
3. Data bit ready
4. Data bit accept for IMP
5. End of transmission

B. To Host

1. IMP transmission accept
2. Data bit line
3. Data bit ready
4. Data bit accept from Host
5. End of transmission

SINGLE LINE CONTROLLER

The Single Line Controller proposed will handle one full duplex synchronous communication line operating at 50 KC. The unit consists of independently operating transmit and receive sections each with its own DMC sub-channel and priority interrupt line.

The transmit section contains a 16-bit buffer register to receive full computer words via DMC output from memory. An eight bit shift register outputs bits to a modem and into the parity generator. The parity generator is the 24-bit polynomial parity generator specified by ARPA. Provision is provided to enter SYN, DLE, ETX, and STX characters to the output shift register. Control logic is included to sequence output operation for both a Sync mode and Data mode with the clock pulses supplied by the modem such as the ATT Data Set 303.

The Transmit section Program Controls are:

1. Interrupt on End of Message (DMC end of range)
2. OCP Reset Interrupt
3. OCP Turn OFF transmit section
4. OCP Turn ON transmit and Start Sync mode
5. OCP Start DMC output and Transmit Data mode

Transmission operation is started by a program OCP which starts sync mode. Sync characters are forced into the shift register and outputted serially to the modem continuously until another program OCP is executed which switches to data output mode, clears the parity generator and starts the DMC subchannel output. The hardware generates the lead DLE, STX. While a data word is being shifted out to the modem and the parity generator, the DMC is requested to deliver another word to the buffer register to anticipate the next output for continuous 50 KC data transmission. When the DMC reaches an end of range, four end operations are executed as follows:

A) Shift out last data word, B) Shift out 24 bit parity, C) Shift out hardware forced (DLE, ETX) and D) Set transmission back to sync mode. Sync characters are again continuously outputted until the program again starts the data mode. At least two sync characters are always outputted between data mode message blocks.

DLE characters within the message are doubled for easy recognition and removal by the receive section.

The receive section consists of a sixteen bit buffer to hold data ready for DMC input, an eight bit input shift register from the modem and a 24 bit parity generator similar to that in the transmit section. A decode off the buffer register for DLE, STX, ETX, and SYN characters as well as a parity zero check are provided as part of the control logic for Sync and Data modes.

Receive section program controls are as follows:

1. Interrupt on end of Message block or error
2. Reset interrupt
3. OCP turn OFF receive section
4. OCP start receive sync mode and enable DMC
5. SKS sense for error

Receive operation is started by a program OCP which starts Sync search mode and enables DMC input. Data stream from modem is monitored for (Sync) and nothing is inputted to the computer. When (Sync) is detected, a sync flop is set and sync characters are allowed to pass until the first 16 bits are received following the last sync character. If these 16 bits are not (DLE, STX), the Sync search mode resumes. If they are (DLE, STX), the parity generator is cleared, and the data input mode started. Data words are inputted to memory via DMC, parity generated and every word input is checked for (DLE). If (DLE, ETX) and parity = zero, it is a good end of message. The computer is interrupted and sync mode search is restarted. If (DLE, ETX) and parity not zero set error, interrupt computer and resume sync mode. If DMC end of range occurs set error, interrupt computer and resume sync search mode. If DLE DLE, eliminate one DLE and continue normal data input mode. In addition to the normal transmit and receive operation, a test mode is provided. An OCP switches the serial output of the transmit section to the serial input of the receive section for test. Another OCP switches back to the modem connection. In the test mode all other regular controls are also operative.

Alternately, these OCP's could be made available beyond the interface for the purpose of implementing program control of the Modem Test Mode. The Modem test modes allows the modem to connect its output to the line to its input from the line, thus forming a loop to the line side of the modem for testing.

REAL TIME CLOCK

A programmable Real Time Clock is provided which consists of a crystal controlled oscillator divided down to provide 20 μ-sec interval increments to a 16-bit clock counter. Whenever the counter reaches zero, an interrupt is generated. Provision is provided to input to the computer the current value of the clock count. Also, it may be reset by output from the A register. In this way interrupt intervals may be controlled to any interval between 20 μsec and 20 x 2^{16} μsec.

SPECIAL PROGRAMMED INTERRUPT

Added to the clock logic block is a special interrupt which is generated by execution of an OCP in the program.

DISPLAY STATUS PANEL

A special Display Status Panel is proposed which contains up to 24 indicator lights. These indicators may be wired directly to selected control flip-flops in the single line controllers and Host-IMP interface or be set from the I/O bus. The latter case is limited to 16 indicators which may be lit or not lit by ones and zeros output from the A register.

SYSTEM PHYSICAL LAYOUT

A maximum IMP system as listed above will fit into one high-boy cabinet containing six standard tilt-out DDP-516 drawers. The 3 tilt-outs mounted in the lower half of the high-boy contain a power supply in one drawer, the memory up to 16K in a second with the computer main frame with DMC in the third. The upper three consist of one power supply and two option drawers. The priority interrupt, the ASR-33 interface, the Real Time Clock, the Host-IMP interface and two single line controllers would be housed in one tilt-out drawer. Up to four more single line controllers will fit into the other tilt-out. Each individual I/O option is relocatable in the drawer tilt-out and is cable connected to the computer below.

In a standard high-boy cabinet, the computer console is mounted in the middle of the front door. The special display panel would be mounted on the door just above the computer console with a cable connect back to the I/O logic.

LAWRENCE PRAGER, Senior Systems Engineer, Information Systems Department

Mr. Prager has overall project responsibility for digital systems used for message switching, data concentration, and management information. His responsibilities include system development, logic design of special interfaces, supervision of construction, test and installation. He also participates in proposal efforts for data communication systems.

Prior to joining CCD, Mr. Prager served as a systems engineer for the Digitronics Corporation where he contributed to the development of remote data communications terminal equipment. The terminals included paper tape reader and punch, card reader and punch, line printer and magnetic tape. He was also reponsible for the design and development of several data conversion and data communications systems for commercial and government use.

Earlier, he served as project engineer for the Bunker-Ramo Corporation where he participated in the design of digital systems used for airline reservations and banking. In this capacity, he was responsible for the design of an airline reservations agent set, a magnetic drum interface,,and communication interfaces.

EDUCATION

Graduate of the RCA Institute of Technology.

CHRISTOPHER B. NEWPORT, Manager, Information Systems Department

Dr. Newport is responsible for the activities of the Communications Section, Scientific Control and Display Section, and the Application Development Section within his department. The work in these sections involves real time computer systems for message switching, data concentration, information handling, real time scientific computing, and display control.

Prior to joining Honeywell CCD, Dr. Newport was employed by the Marconi Company, Ltd., of England, as a Systems Manager in the Automation Division. He was responsible for designing and developing real time industrial and communications computer systems. He was also responsible for the hardware and software design of many fully-duplicated message switching systems and some special computer-based communication systems. Among his accomplishments at Marconi was the design and manufacture of a dual computer control and monitoring system for a nuclear power station. The system had approximately 7000 digital and analog inputs, and used 20 alpha-numeric CRT displays as the primary operator output.

Before his employment with Marconi, he worked at the Atomic Power Division of English Electric, Ltd., and was responsible for such tasks as the design and construction of a large general purpose analog computer which used 1500 operational amplifiers. His experience at English Electric also included power station automation and the design of special purpose and computer-based automation systems.

Education

Dr. Newport obtained a B.Sc. (1st Class Honors) in Electrical Engineering in 1954, and a Ph.D. in Electrical Engineering in 1959, from Birmingham University in England. He was a research assistant in the Electrical Engineering Department of Massachusetts Institute of Technology in the years 1956 to 1957.

Professional Affiliations

Member, Institution of Electrical Engineers, London, England.
Member, Association of Computing Machinery.

5 QUALITY ASSURANCE AND RELIABILITY

5.1 Highlights of DDP-516 Reliability

High reliability for the DDP-516 has been achieved by the latest integrated circuit technology, together with extensive experience on the DDP-24, DDP-224, DDP-116, and DDP-124 digital computers. The DDP-124 computer and the ICM-40 integrated circuit memory represent the first commercially available integrated circuit computer devices. The DDP-516 contains many identical μ-PAC types used in the DDP-124 and therefore has the same reliability features.

Company-sponsored reliability programs at Honeywell Inc., Computer Control Division (3C) are instituted at all phases of circuit design, computer design, test and customer field service to achieve the highest possible reliability. As a specific example, the integrated circuits for the DDP-516 were specially fabricated after careful consideration of circuit requirements for fan-in/fan-out capability, noise rejection, speed and compatibility with system requirements. After initial design the integrated circuit devices were manufactured and qualified in accordance with 3C specifications. A portion of the qualification test is included in the DDP-516 Reliability Manual to emphasize the importance of these tests and their impact upon high reliability performance. The majority of the integrated circuit devices for DDP-516 are manufactured in high volume by outside vendors with long established reputations for making semiconductor devices. Moreover, 3C maintains its own integrated circuits laboratory with complete facilities for production from silicon wafer to finished device, in order to meet special requirements and advance the state-of-the-art of integrated circuit technology as applied to digital computers. All aspects of design are included and considered including device fabrication, printed circuit design, subsystem design (including memory devices), and system design at the computer level. The development and fabrication of the DDP-516 is a complex process which involves many different disciplines each with a critical impact on final computer reliability. The results of these efforts can be briefly summarized as an integrated organization for integrated use of integrated circuits.

Since many complex functions are involved in the design and fabrication of a computer, 3C has initiated a company sponsored Product

Reliability Evaluation Program (PREP) to evaluate data received from device fabrication, μ-PAC fabrication, system assembly, and customer usage. This program obtains all relevant reliability data to demonstrate reliability and to pinpoint specific problems should they occur. The portions of the program applicable to customer usage are included in a section of the DDP-516 Reliability Manual entitled "Field Service Reporting System." This information is periodically tabulated on punched cards, with automatic summary tabulations of all reliability data.

In addition to design and performance data, the third major area of reliability evaluation for the DDP-516 includes reliability predictions and demonstrations. Predictions are based on circuit analysis by component type, quantity and power dissipation in accordance with procedures defined by the USAF Rome Air Development Center, RADC Handbook and other military and NASA procedures for reliability predictions. In addition, quantities of Honeywell μ-PACs, S-PACs and other standard products are periodically placed on life test to determine long-term stability and inherent failure rates. For example, the S-PAC discrete component life test has been operating for over five years with only a single diode failure. Since μ-PACs and integrated circuits are somewhat newer devices, it has not been possible to obtain the same number of operating hours. However, similar programs are presently in operation for μ-PACs as indicated in the DDP-516 Reliability Manual. As a result of these predictions and demonstrations it is possible to state that the typical μ-PAC has a demonstrated reliability better than the RADC prediction. As a result, the reliability of the DDP-516 is conservatively estimated at 4000 hours MTBF.

5.2 Product Assurance Provisions

3C products are designed for long life and high reliability performance. The company continually conducts life tests on the digital logic module lines it produces. The results of these tests point up the effectiveness of the company program of superior circuit design, incorporation of field information, and Reliability Analysis.

The Corporate Product Assurance function reports to the Director of Engineering Services who reports directly to the Vice President in charge of Engineering. Reporting to the Director are the following functions:

Product Assurance
Reliability Engineering
Engineering Records
Test Equipment Calibration

The Product Assurance function ensures that the standards for all products reflect the established basic corporate quality objectives.

Reliability Engineering maintains the life-test program on the S-PAC and μ-PAC product lines to provide reliability numerics and determine component life characteristics. Reliability Engineering participates in selection of components, writing of purchase specifications, and the evaluation of the vendors capabilities.

Engineering Records prepares and publishes engineering and manufacturing standards for design, quality, and workmanship. It prepares and issues specifications and procedures encompassing purchasing, inspection, instrument calibration, and basic manufacturing processes. It also coordinates requests for engineering investigations and changes.

Test Equipment Calibration maintains the corporate electrical standards and ensures maintenance and periodic calibration of all types of test equipment.

All quality procedures and standards as well as records of inspections and tests are available for review by qualified customer representatives. The Company welcomes the opportunity to escort and assist visitors surveying our facilities.

5.3 Quality Control

The 3C Quality Control Program satisfies all applicable requirements of MIL-Q-9858 (Quality Control System Requirements). It audits the capacity of suppliers to meet 3C quality standards and Engineering, Manufacturing, and supplier conformance to established specifications. The Quality Control Program and manual have been coordinated with the U.S. Navy Quality Assurance Representative.

Inspection and Test, performed under the Quality Control System, is governed by specific written procedure. It includes the following functions:

> Incoming inspection and test of all material, parts, and assemblies;
>
> Scheduled in-process inspection of 3C digital modules during fabrication;
>
> Final mechanical inspection of digital modules;
>
> Complete electrical tests on digital modules;
>
> Scheduled in-process inspection during system assembly; and
>
> Final mechanical inspection of assemblies.

5.4 DDP-516 Reliability Estimates

A reliability goal of 4000 hours MTBF was established for the DDP-516 prior to completion of the initial design. As such, this goal represents typical customer requirements, as well as a comparison with other 3C computers. For example, the design goal for the DDP-124 integrated circuit computer was 2000 hours MTBF or on the average of a single μ-PAC failure during typical operation over a one year period with 40 hours per week. On a design basis, therefore, the goal of 4000 hours MTBF for the DDP-516 represents twice the reliability of the DDP-124. (Because both computers use similar types of integrated circuits, the differences in reliability are mostly a function of the larger size and complexity of the DDP-124.)

Table 5-1, a tabular listing of the DDP-516 options and μ-PAC complements, includes a reliability apportionment based on 4030 hours for option DDP-516-02 general purpose digital computer with 8192 words of core memory. Derivation of this MTBF requires a failure rate modifier of 0.339 or approximately three times better than the RADC predictions. This is substantiated by data derived from μ-PAC and S-PAC life tests which indicates that a failure rate modifier of 2.15 better than RADC can be demonstrated for typical μ-PACs and S-PACs depending on the length of time of the test.

Model No.	Description
516-01	DDP-516 general purpose digital computer with 4096 words of core memory
516-02	DDP-516 general purpose digital computer with 8192 words of core memory
516-03	DDP-516 general purpose digital computer with 12,288 words of core memory
516-04	DDP-516 general purpose digital computer with 16,384 words of core memory
516-05	DDP-516 general purpose digital computer with 24,576 words of core memory
516-06	DDP-516 general purpose digital computer with 32,768 words of core memory
516-11	High-speed arithmetic unit
516-20	Direct multiplex control unit (DMC)
516-50	Paper tape reader, 300 cps
516-52	Paper tape punch, 110 cps
516-53/55	ASR-33/35 Teletype unit
516-61	Card reader, 200 cpm

Table 5-1.
DDP-516 Options and μ-PAC Complements

		STANDARD														COMPUTER																					MEMORY								
	μ-PAC Types	DC-335	DG-335	DI-335	DL-335	DM-335	DN-335	FF-335	LC-335	LD-335	OD-335	PA-335	PA-336	ST-335	TG-335	CC-002	CC-012	CC-034	CC-035	CC-036	CC-037	CC-038	CC-039	CC-043	CC-044	CC-045	CC-046	CC-054	CC-055	CC-057	CC-073	CC-074	CC-085	CC-089	CC-091	CC-092	CM-003	CM-006	CM-022	CM-032	CM-033	CM-075	CM-106		
	Maximum Expected Failure Rates (%/K Hr) At 45°C	.07	.06	.05	.05	.10	.03	.07	.27	.83	.14	.03	.03	.21	.06	.45	.15	.12	.09	.15	.13	.11	.14	.11	.17	.05	.08	.00	.22	.08	.05	.18	.03	.04	.08	.06	.11	.31	.07	.08	.04	.01	.28	Total Failure Rate (%/K Hr)	MTBF (Hr)
MODEL NUMBERS	516-01	9		10	20		6				2	4	12		10	6		1	1	1	8	16	8	1	1	4	1	4			6		3				2	34	1		8	1	4	23.9	4,180
	516-02	9		10	20		6				2	4	12		10	6		1	1	1	8	16	8	1	1	4	1	4			6		3				2	34	1	8		1	6	24.8	4,010
	516-03	9		10	21		6				2	8	12		12	6		1	1	1	8	16	8	1	1	4	1	4			6		3				4	68	1	8	8	2	10	37.3	2,680
	516-04	9		10	21		6				2	8	12		12	6		1	1	1	8	16	8	1	1	4	1	4			6		3				4	68	1	16		2	12	38.1	2,620
	516-05	9		10	22		6				2	12	12		14	6		1	1	1	8	16	8	1	1	4	1	4			6		3				6	102	1	24		3	18	51.5	1,940
	516-06	9		10	23		6				2	16	12		16	6		1	1	1	8	16	8	1	1	4	1	4			6		3				8	136	1	32		4	24	64.9	1,540
	516-11	3		2	6		3								1											1													5					1.2	83,330
	516-20		3	2	1		1						2		5										1									1	2				2					1.2	83,330
	516-90	1		1			1		1						1																			1		1			1					0.6	166,660
	516-52	1		1			1	1						1	3		1												3					1										1.5	66,670
	516-53/55	1		2	1		2						1		2															1		1	1	4					1					1.0	100,000
	516-61	1		2		1	2		2	1					3																			1		2			1					2.1	47,620

Installation Manual (Doc. No. 130071625)

Interface Manual (Doc. No. 130071624)

Instruction Manual for Teletype Models 32 and 33 Typewriter Sets, Keyboard Send-Receive (KSR), Receive-Only (RO), Automatic Send-Receive (ASR). (Vol. I Doc. No. 130071453; Vol. II Doc. No. 130071455)

Instruction Manual for Teletype Parts Model 32 and 33 Page Printer Set -- ASR, KSR, and RO -- (Doc. No. 130071454)

6.2 Programming Documentation

- Programmers Reference Manual (Doc. No. 130071585)
  - Programmers Reference Card (Doc. No. 130071623)
- FORTRAN IV Compiler Program (when applicable)
  - FORTRAN IV Manual (Doc. No. 130071634)
  - Operating Instructions
  - Compiler Listing
  - IOS Listing
- Assembly Program
  - DAP-16 Assembler Manual (Doc. No. 130071629)
  - Operating Instructions
  - DAP Listing
  - IOS Listing
- Users Guide (Doc. No. 130071627)
- Input/ Output Library (Doc. No. 130071631)
  - Input/ Output Subroutine Listings
  - FORTRAN IV I/ O Control and Driver Subroutine Listings
  - Conversion Routine Listings
- Math Library (Doc. No. 130071632)
  - Arithmetic Routine Listings
    - Fixed Point
    - Floating Point
    - Complex
  - Standard Functions Library Listings
    - Fixed Point (without MPY/ DIV) or Fixed Point (with MPY/ DIV)
    - Integer

Floating Point
Complex
FORTRAN Control

Utility Routine Listings (Doc. No. 130071635)
- DAP/ FORTRAN Relocation Loaders
- Memory Dump
- Checkout Package
- Symbolic Program Update System

Verification and Test Program Listings (Doc. No. 130071633)
- Central Computer
- Core Memory
- Interrupt
- Timers
- I/O Devices
- Test Program Loaders

6.3 Special Documentation

When a system includes special features, supplementary data regarding these features will be supplied in the same format as the standard documentation and will be either appendices to standard manuals or in separate manuals according to the requirements of the system.

APPENDIX C: TIMING COMPUTATIONS AND THE RFQ MODEL

A central computation in the design and evaluation of the network is the determination of the actual amount of IMP processing time. It affects the selection of the IMP computer, the performance and utilization of the chosen computer, and forms a basis for the required RFQ model calculations. It also strongly affects the design of the hardware interface and, in conjunction with the chosen computer, forms a principal measure of the expansion capability of the network.

However, this computation cannot be performed without making some estimate of the traffic which an IMP is expected to handle. The results which are obtained are extremely sensitive to the initial assumptions. In this appendix we will discuss two sets of assumptions which we label as A and B.

Assumption A: This is the assumed traffic in the RFQ model. Each channel carries 15 kilobits/sec and the Host line carries 20 kilobits/sec. The average packet size on a channel is 344 bits and the average packet size on the Host line is 576 bits. There are four channels and one Host line.

Assumption B: This corresponds to a "reasonable" peak load condition and is identical to assumption A except that all channels as well as the Host line are assumed to carry 50 kilobits/sec.

We determine the total number of bits per second, R, and the average number of packets per second, P, that cross an IMP interface in any direction.

A: $R = 8 \times 15{,}000 + 2 \times 20{,}000 = 160{,}000$ bits/sec

$$P = \frac{120{,}000}{344} + \frac{40{,}000}{576} = \sim 420 \text{ packets/sec;} \tag{1}$$

B: $R = 10 \times 50{,}000 = 500{,}000$ bits/sec

$$P = \frac{400{,}000}{344} + \frac{100{,}000}{576} = \sim 1325 \text{ packets/sec.} \tag{2}$$

There are two primary components to the calculation of the IMP processing time, namely the time required for I/O transfers and the time required for internal packet processing. We first consider the total cycle time, T_T, required to do input-output transfers.

We assume four cycles per I/O transfer (core counters are assumed instead of hardware counters for reasons of economy) and set

W = Word length in bits
C = Cycle time in µs
I = Instruction time in µs.

$$\text{A:} \quad T_T = \frac{160{,}000}{W} \times 4C \quad \mu\text{s/sec;} \tag{3}$$

$$\text{B:} \quad T_T = \frac{500{,}000}{W} \times 4C \quad \mu\text{s/sec.} \tag{4}$$

Within each IMP, the bulk of the processing is performed on a per packet basis. We have estimated the average number of instructions required in the IMP program to process these packets. There are four basic components of the processing (see Appendix F on Software).

$$\left.\begin{array}{ll}\text{INPUT INTERRUPT ROUTINE} & \text{- 40 instructions}\\ \text{INPUT TASK PROCESSING} & \text{- 40 instructions}\end{array}\right\}\begin{array}{c}\text{80 for}\\ \text{input}\end{array}$$

$$\left.\begin{array}{ll}\text{OUTPUT INTERRUPT ROUTINE} & \text{- 20 instructions}\\ \text{OUTPUT TASK PROCESSING} & \text{- 20 instructions}\end{array}\right\}\begin{array}{c}\text{40 for}\\ \text{output}\end{array}$$

We average these quantities to obtain a figure of 60 instructions/ packet in crossing an IMP boundary. We further estimate that all additional tasks will average another 30 instructions/packet. Therefore we use the figure of 90 instructions/packet as the average number of instructions which must be performed by the IMP program to process each packet which crosses the IMP boundary. Note that a packet which *traverses* the IMP is thus assigned a total of 2 × 90 = 180 instructions.

The total program instruction time, T_I, is given by

$$\text{A:}\quad T_I = 420 \times 90I = \sim 38{,}000I\ \mu s/sec; \tag{5}$$

$$\text{B:}\quad T_I = 1325 \times 90I = \sim 120{,}000I\ \mu s/sec. \tag{6}$$

We now wish to estimate the individual instruction time, I, for a small sized computer. It is reasonable to assume that in a hypothetical 20 bit machine, indirect addressing should never be required to access any word of memory (in a typical IMP configuration of less than 16K). We assume that such a 20 bit machine requires an average of 2 cycles per instruction and that a machine with a shorter word length, W, will require approximately 2 × 20/W cycles/instruction due to an increasing frequency of indirect addressing with decreasing word size.

Thus, we have the following expression for the instruction time

$$I = \frac{20}{W} \times 2C \ \mu s$$

and the total program instruction time, T_I, for handling packets is

$$\text{A:} \quad T_I = 38{,}000 \times \frac{20}{W} \times 2C = 1.52 \times 10^6 \ \frac{C}{W} \ \mu s/sec; \qquad (7)$$

$$\text{B:} \quad T_I = 120{,}000 \times \frac{20}{W} \times 2C = 4.8 \times 10^6 \ \frac{C}{W} \ \mu s/sec. \qquad (8)$$

On adding Eq. 3 to Eq. 7 and 4 to 8 we obtain an estimate, $T = T_T + T_I$, of the total cycle time required to handle the IMP traffic.

$$\text{A:} \quad T = 6.4 \times 10^5 \ \frac{C}{W} + 1.52 \times 10^6 \ \frac{C}{W} = 2.2 \times 10^6 \ \frac{C}{W} \ \mu s/sec; \qquad (9)$$

$$\text{B:} \quad T = 2 \times 10^6 \ \frac{C}{W} + 4.8 \times 10^6 \ \frac{C}{W} = 6.8 \times 10^6 \ \frac{C}{W} \ \mu s/sec. \qquad (10)$$

[Results 9 and 10 help in the discussion of the IMP computer choice and they are used for that purpose in Appendix D.]

From this point we will simply assume that

$$C = 1$$
$$W = 16$$
$$I = \frac{20}{W} \times 2C = 2.5 \ ,$$

since these are the appropriate values for the DDP-516, and proceed with the computation of the timing and the model.

A: $T = 2.2 \times 10^6 \times \frac{1}{16} \cong 0.14 \times 10^6$ μs/sec *or 14% of capacity*;

(11)

B: $T = 6.8 \times 10^6 \times \frac{1}{16} \cong 0.43 \times 10^6$ μs/sec *or 43% of capacity.*

(12)

Therefore, under assumption A, only 14% of the machine capacity is used, while at the "reasonable" peak loads of condition B approximately 43% of the machine capacity is used.

1.0 RFQ MODEL CALCULATIONS

1) Under assumption A, the number of bits/second *input* into an IMP is obtained from Eq. 1, as one-half the total number of bits which cross a boundary. This quantity is 80,000 bits/sec.

2) The number of packets/sec entering and leaving a node is obtained directly from Eq. 2 and is 420 packets/sec.

3) The time available to process a packet is the reciprocal of 420, or 2.3 ms/packet.

The chosen processor has already been shown to have this computation capacity, since only 14% of the machine will be used. The required total time per packet may also be calculated as 90 instructions/packet times 2.5 μs/instruction plus 344/16 × 4 μs memory access

$90 \times 2.5 + 86 = 311$ μs/packet (out of an available 2.3 ms).

(13)

4) We now calculate the components of the message delays which are incurred in each IMP.

Let us first consider the channel between the Host and the IMP. As a sample case, we assume that a typical time-shared Host gives the network a share of a multiplexer channel, and is thus willing to read an IMP word at least every 25 sec. The time required to cross the Host-IMP interface is therefore 30 μs/word (including 1.6 μs of shift time in the interface and 4 μs of IMP memory access). Thus the Host interface can process

$$\frac{16}{30 \times 10^{-6}} \cong 533{,}000 \text{ bits/second,}$$

well above the maximum it will be expected to handle if all IMP modem lines are running at 50K bits. Consequently, the primary delays do not occur in the receiving Host queue, if the Host is behaving responsibly toward network users. We now calculate the line, modem, and IMP delays for the following path which is shown below.

$$\begin{tikzcd} \text{Host A} \arrow[r] & \text{IMP 1} \arrow[r, "300\text{mi}"] & \text{IMP 2} \arrow[r, "300\text{mi}"] & \text{IMP 3} \arrow[r, "300\text{mi}"] & \text{IMP 4} \arrow[r] & \text{Host B} \end{tikzcd} \tag{14}$$

a. *Communication delay.* The RFQ indicates that this quantity is 3.17 μs link, and is composed of a propagation delay, and two modem delays.

b. *Full packet transmission delay.* For simplicity in the remaining calculations, we consider a worst case assumption in which every packet is a full 1040 bit packet. The average queueing and transmission delays for this case will be conservative estimates of the actual network delays. Since, each modem takes one bit every 20 μs, the full packet requires 20.8 ms to transmit. During this time, no other packet may be transmitted over the Line.

c. *IMP processing delay.* We have already estimated the total amount of instruction time to be $90 \times 2.5 = 225$ μs per packet.

d. *Queueing delays.* We now estimate the average queueing delay for the assumed network rates.

This is a most difficult quantity to estimate since the queuing discipline is dynamic and the distribution of Host-generated traffic is unknown. We therefore make some simplifying assumptions. First of all we assume the internal distributions of IMP traffic is equally distributed to all the outgoing lines. We assume a constant service time on each line which we take to be equal to the overall average service time. Finally, we assume that the total arrival rate to each line, which is an aggregate from all of the other lines and the Host, may be characterized by a Poisson arrival process. This assumption has as its limitation the fact that the number of arrivals that may occur in any finite time interval is not bounded, but is one of the two situations we are able to analyze — and it seems to be the most reasonable assumption to make.

We next compute the constant packet service time in an IMP and the packet arrival rate on each channel. The constant service time $1/\mu$ is taken to be the transmission time plus the processing time, which is

$$\frac{1}{\mu} = 20.8 + 0.225 \cong 21 \text{ ms.} \qquad (15)$$

The full packet arrival rate on each channel is

$$\lambda = \frac{15000}{1040} \cong 15 \text{ packets/sec.}$$

The average delay time in an IMP is equal to the constant service time plus the average queueing delay. The expression for the average delay is

$$\frac{\beta}{1-\beta}\,\frac{1}{2\mu}$$

(see Saaty, *Elements of Queueing Theory*, p. 161), where $\beta = \lambda/\mu$ is defined to be the load.

On performing this calculation, we obtain $\beta = 15 \times 0.021 = 0.31$ and the average queueing delay, Q, is

$$Q = \frac{0.31}{1-0.31} \times \frac{21}{2} \cong 4.7 \text{ ms.} \qquad (16)$$

Therefore, the total delay, D, per link is the sum of the four components of the delay, namely

$$D = (3.17 + 20.8 + 0.23 + 4.7) \cong 28.9 \text{ ms/link} \qquad (17)$$

and the total delay over three links is three times this quantity, or ~86.7 ms.

5) For an average path of three links, a maximum delay of 1/2 second requires that the delay per IMP not exceed 500/3 = ~167 ms. We therefore encounter 167 - (3.17 + 20.8 + 0.23) = 142.8 ms. of queueing delay per IMP. This will not occur until each line is almost used to capacity.

More precisely, we use the queueing delay formula to solve for β, which gives us

$$\beta = \frac{142.8}{142.8 + \frac{21}{2}} \cong 0.93. \qquad (18)$$

From this value of β, it follows that the full packet arrival rate for which the maximum delay is one-half second is

$$\lambda = \frac{0.93}{0.021} \cong 44.3 \text{ packets/second.} \qquad (19)$$

This corresponds to a line rate of $44.3 \times 1040 \cong 46{,}000$ bits/sec which is just below the line capacity as expected. Thus, we see that the 1/2 second delay does not begin to arise until each line is used close to capacity.

6) With all other nodes quiet, the maximum packet Input rate on the Host line is $50{,}000 \times n$ bits/sec where n is equal to the number of channels. Therefore, the total input packet rate is

$$2 \times \frac{50{,}000\ n}{576} \cong 175\ n \text{ packets/sec.}$$

The actual maximum input rate from the Host must be somewhat below 50,000 n bits/sec, however, to assure that the total delay does not exceed 1/2 second.

APPENDIX D: COMPUTER CHOICE

Many factors are involved in a computer choice in today's market. We started the selection process with several key factors in mind:

1) The machine should be a *shelf* product, with many copies in the field. The machine should be physically small, as cheap as possible, and suitable for long unattended operation, preferably offering standard ruggedized options.

2) The machine should permit numerous independent buffered block transfers, all with interrupts on completion, without special engineering if possible, and at a reasonable cost. (This cost factor mitigates against hardware count registers and toward the use of core registers for I/O control.)

3) The machine must handle "reasonable" *peak* loads with a safety factor of, say, *two* or *three*, and handle *expected* loads with a safety factor of, say, *four to six*, to allow for *estimation error*, growth, and Host-unique tailoring.

4) The manufacturer should be sensibly set up to assist in the design and production of specialized interface hardware, to assist in integration, field installation, and maintenance. Preferably, the manufacturer should also be physically convenient to BBN and have service facilities convenient to the network sites.

We will first consider the timing question. From Appendix C (Timing Computations and the RFQ Model), we take Eqs. 9 and 10:

$$A: \quad T = 2.2 \times 10^{6} \frac{C}{W} \quad \mu s/sec$$

$$B: \quad T = 6.8 \times 10^{6} \frac{C}{W} \quad \mu s/sec \quad ,$$

where A implies average rates and is about 1/3 of B, where B implies 50 KB/sec "reasonable" peak rates, and where

T = total time
C = cycle time in μsecs
W = word length in bits

We will consider the peak rate case and evaluate Eq. 10 for small machines:

		W	
		12	16
C	1	57%	43%
	2	114%	86%

Now, if we use assumption set A, the numbers in this table are reduced by a factor of approximately 3.

We made these computations several different ways, with different assumptions, and were led *strongly* to seek machines with $\leq$ 1 μs cycle times, at least 16 bits, and with powerful order codes. Put another way, this job is *tight* for a small machine, and we obviously wanted a "large powerful entry" from the small class, if we were to avoid considering a much more expensive breed. We rapidly discarded the PDP-8 and smaller group of machines using this analysis.

We next simply considered all the machines we knew of whose "basic" price was $\leq$\$50,000, and started filtering on the basis of the stated factors, with factor #3 translated to a cycle time of 1 μsec, a capacity of at least 16 bits, a powerful order code, and indexing.

By a considerable margin, the DDP-516 was the best selection we could make. We believe that it is close to the performance peak in its class, and that it can do the job. It is a shelf item, physically small, competitively priced, offering ruggedized standard options, and it has a particularly suitable I/O system design. (Of course, other machines were certainly sensible; as an example of a reasonably close contender, the slightly faster SEL 810B met many of the criteria. However, in this case, the I/O system comparison, the Framingham location of Honeywell, and the much greater field experience with the DDP-516 were deciding factors.)

We chose the DDP-516.

APPENDIX E: HARDWARE DESIGN

1.0 THE DIRECT MULTIPLEX CONTROL UNIT

Because of its central role in providing memory accesses for the interface units, the DMC operation warrants a brief description. The DMC provides for up to 16 channels of access to the IMP memory. A pair of memory pointers for each channel control block transfers. Channels are serviced in a priority order. Each memory access requires four memory cycles in all (including pointer references).

For Input, a device requests memory access via the DMC by presenting a DIL signal on that device's line. When the DMC is free (i.e., no higher priority requests waiting), a DAL signal is sent to the device. The device uses this DAL signal to gate its data onto the common data bus to the DMC and also removes the DIL signal. The computer takes in the data word, stores it in memory according to the first pointer, and increments the pointer. It then shuts off the DAL signal and the cycle is ready to repeat. When the allocated memory buffer region is full,(i.e., on the cycle in which the first pointer matches the second — terminal — pointer), a special End of Range Signal (ERL) is sent to the device. The device then interrupts the computer program which, in turn, resets the pointers for the next buffer area and restarts the device.

For Output, a device requests a memory word by presenting a DIL signal to the DMC. The request is acknowledged by a DAL signal. A special signal (RRLIN) is provided for strobing the data from the output bus into the device's buffer. End of Range is indicated as for Input.

2.0 HOST TO IMP INTERFACING — DETAILED DESCRIPTION

Figure E-1 shows details of the logic and Figure E-2 shows the details of timing. Generally, bit flow across the boundary from the Host is controlled by two signals; "Ready for Next Host Bit" (RFNHB), a signal from the Interface to the Host, and "There's Your Host Bit" (TYHB), the answering signal. The Host typically loads an output bit to the data line and waits for RFNHB. When RFNHB comes on, TYHB is turned on. When the Interface sees TYHB, a Shift And Count (SHAC) pulse is generated which shifts in the data bit from the Host and steps C, a modulo 16 counter which decides when a word is ready to go to the IMP memory. After a data bit has been shifted in (DelA), RFNHB is turned off — in response to which the Host drops TYHB and moves up the next bit. After Del B* has allowed time for the Host to note that RFNHB went off and for the new value in C to set up, the count is checked. If the counter has not reached zero, RFNHB is turned on again to request the next bit. When the counter reaches zero, a full 16 bit word is assembled and ready for shipment to the IMP memory. At that point, instead of turning on RFNHB, the DIL flip flop goes on, presenting a memory access request to the DMC. When the DMC is available (i.e., completes servicing of the current or any higher priority requests), the DAL line for this channel will come on. That action gates the data word onto the bus to the DMC and drops the DIL request. After the word has been taken into the memory system, DAL goes off, which normally turns on RFNHB and the whole process repeats. At some point the Host will have to

*Del B is also used for Del A recovery (and vice versa) where they form a loop to fill in zeros at the end of the Host message — see below.

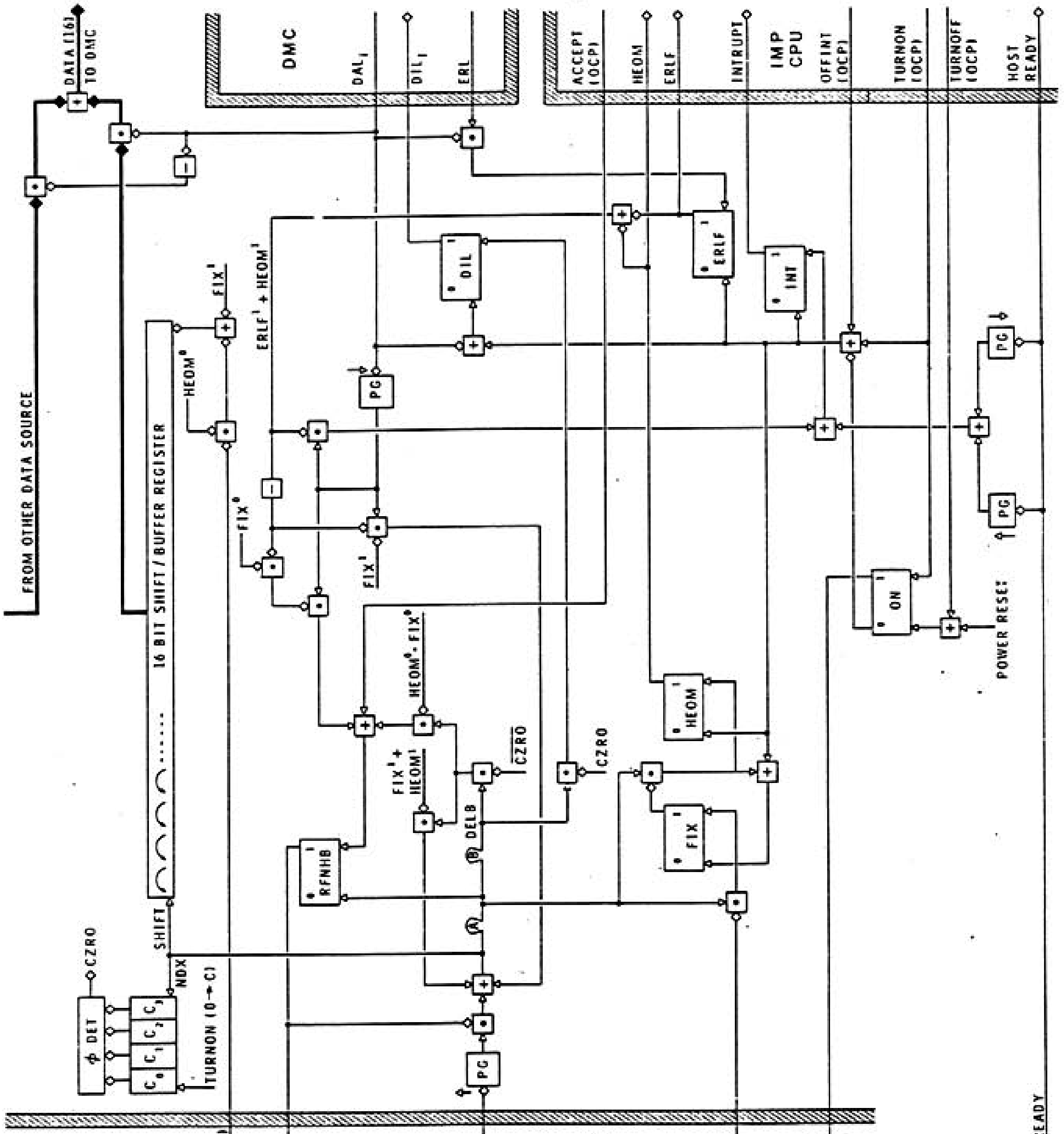


FIG. E-1 STANDARD HOST/IMP INTERFACE-HOST TO IMP SECTION

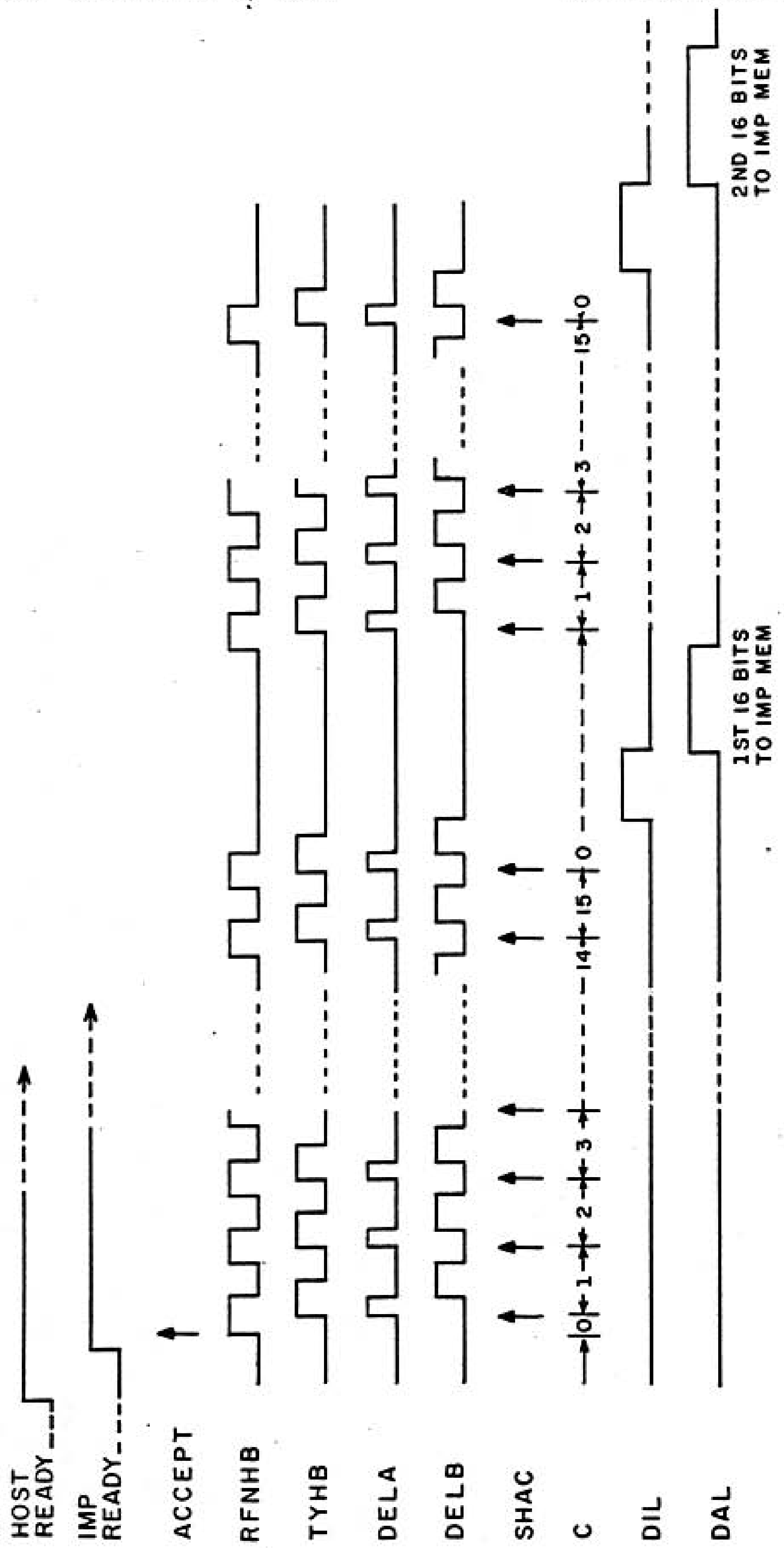


FIG. E-2 TIMING OF HOST-TO-IMP TRANSFER.

access its memory for a new word, but the Interface has no cognizance of that event. (It will probably manifest itself in a somewhat lengthened delay between turn-on of RFNHB and the answering TYHB.)

Words thus come out of the Host memory, pass serially across the interface, and are formed into 16 bit words which go into the IMP memory. A more gross boundary is reached when the pointers used by the DMC indicate that the buffer in the IMP, specified by the program, has been filled. When that happens, an ERL signal is returned to the Interface together with DAL. This sets the ERLF flip flop and when DAL goes off instead of turning RFNHB on, an interrupt request is presented to the IMP; i.e., the INT flip flop will be turned on. The interrupt routine can sense the ERLF flip flop and turn it off, together with INT, via an OFFINT command. As stated above, the interrupt routine will normally reset the buffer pointers and issue a new ACCEPT command, which continues the flow of information into the new buffer location in the IMP memory.

When the end of the Host's message is reached, a Last Host Bit Indicator (LHBI) level is presented with TYHB to the Interface Unit. A FIX flip flop is then turned on as RFNHB goes off. With FIX (or HEOM, see below) on, a loop is closed through DEL A and DEL B which artificially shifts the buffer register. FIX is on for only one bit time and forces a "1" into the end of the buffer following the last data bit. Then HEOM (Host End of Message) is turned on and any necessary trailing zeros come in to fill up the word.* Note that the "one" data input to the shift register is (Host Bit^1 and HEOM^0) + (FIX^1). Note also that the forced one is

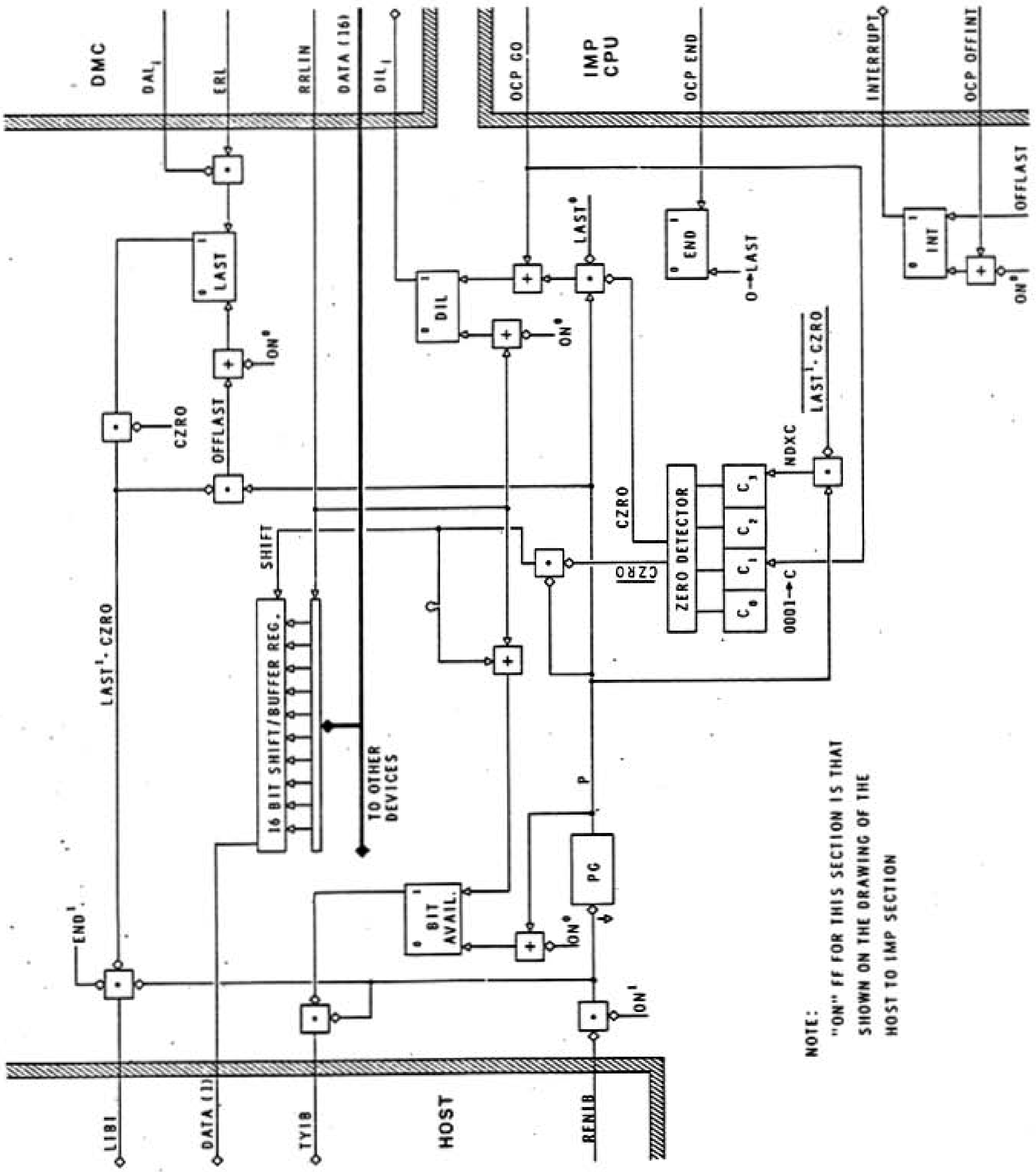
*See Chapter III which describes padding requirements and techniques.

guaranteed even if the last bit of the Host message just filled up a 16 bit word. When DAL returns, if FIX is on, it starts the delay loop that will manufacture a word with a "1" at the left end and zeros to the right. Finally, the word is passed to the DMC in the usual fashion. When the DAL line drops, the INT flip flop will be turned on, thereby interrupting the IMP which can sense HEOM and turn it off similarly to ERLF.

3.0 IMP TO HOST INTERFACING — DETAILED DESCRIPTION

Figure E-3 shows details of this section of the Host/IMP Interface Unit. So long as the Unit is not ON, all control flip flops are held in the zero state; the one incoming control line from the Host side (RFNIB—Ready for Next IMP Bit) is ignored. After TURNON of the Interface, RFNIB is recognized if the Host turns it on. RFNIB, in turn, enables return of TYIB (There's Your IMP Bit) and LIBI (Last IMP Bit Indicator) to the Host. Figure E-4 shows the timing of a transfer.

When the IMP program has a packet ready to go, (pointers set, etc.) it notifies the Interface Unit to start taking the information by executing an OCP GO. This presets counter C to 0001 and turns on the DIL flip flop. A memory request, therefore, goes to the DMC which then loads the appropriate word onto the bus and gives back DAL and RRLIN. RRLIN strobes the data into the Interface output buffer register, turns off the DIL flip flop and turns on the Bit Available flip flop. If RFNIB is on, or when it comes on, TYIB will be presented to the Host. After the bit has been taken in by the Host, RFNIB will go off, in turn shutting off TYIB and producing pulse P. P shuts off Bit Available and tests the value of



NOTE:
"ON" FF FOR THIS SECTION IS THAT SHOWN ON THE DRAWING OF THE HOST TO IMP SECTION

FIG. E-3 STANDARD HOST/IMP INTERFACE-IMP TO HOST SECTION

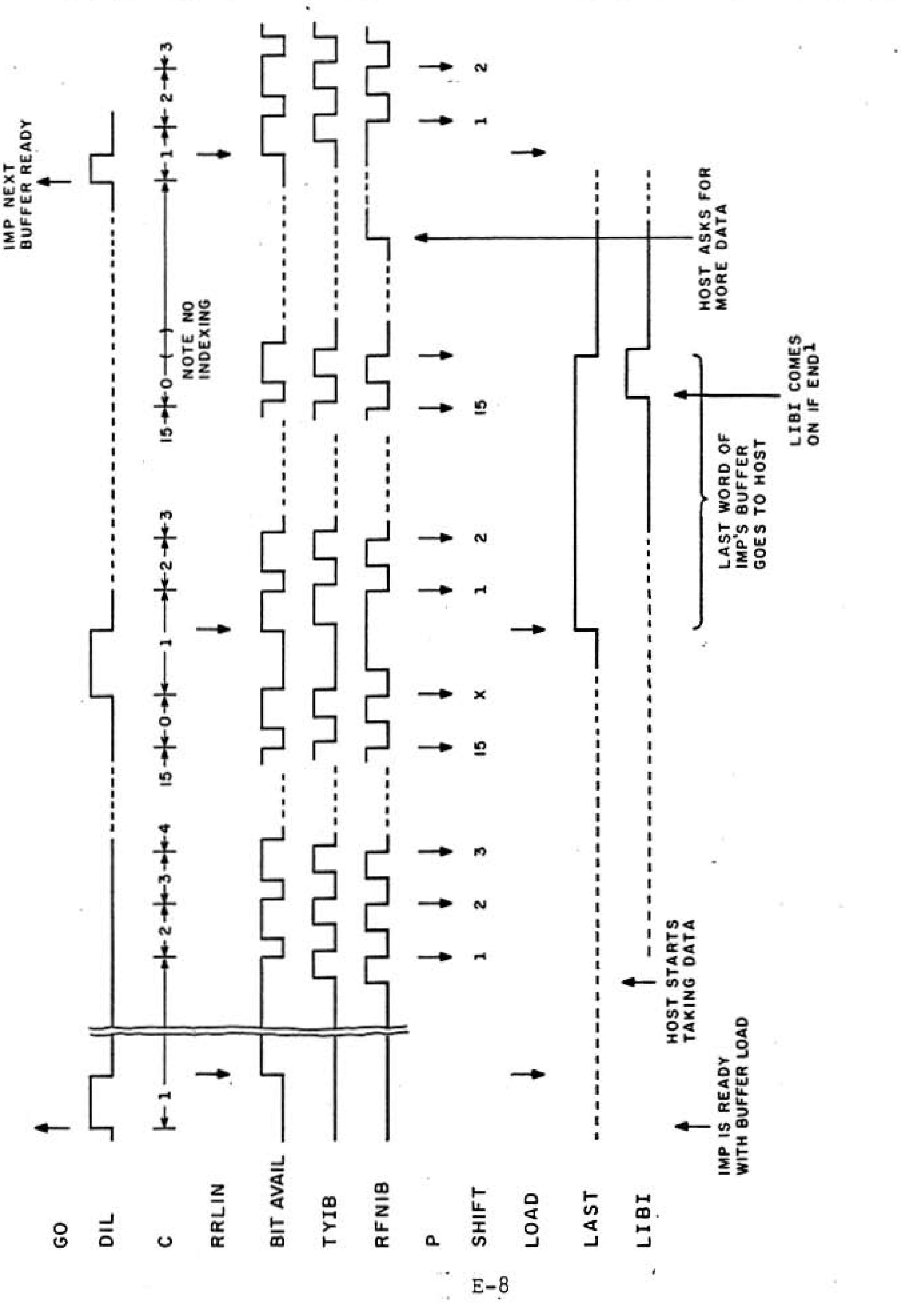


FIG. E-4   TIMING OF IMP-TO-HOST TRANSFER.

the count in C, normally indexing it at the same time. If C does not contain zero (i.e., if 15 shifts have not yet occurred), then a shift takes place and, after a short delay (to allow the shift to settle), Bit Available is turned on again.

After 15 shifts, the rightmost bit of the word will be fed to the Host and the counter C will contain the value 0000. When that bit has been accepted, (P pulse), the counter indexes as usual and Bit Available goes off. However, no shift pulse occurs and instead the DIL flip flop is turned on, thereby requesting another word from the IMP memory. When the word is loaded, RRLIN turns DIL off and Bit Available on, and the major cycle repeats.

Occasionally, the Host will be somewhat slower than usual about taking the next bit as it takes time out to put a word away in its memory. It may even be that a Host Memory Buffer area fills up, requiring a still longer pause while the Host program performs re-cycle activities. All such delays are invisible to the Interface, which patiently awaits the RFNIB signal.

At some point, the IMP Buffer will be emptied, as indicated by the appearance of the ERL signal with return of a word from the DMC. This signal sets the LAST flip flop. C will contain the value 0001 when this happens, but LAST has no effect until C counts to 0000, thus permitting the last word to be shifted out to the Host in the usual way. When C reaches the value 0000, the fact that LAST is on prevents C from counting further and inhibits turnon of the DIL flip flop. If the END flip flop is on at that point, the LAST flip flop is gated to the Host as LIBI (Last IMP Bit Indicator). The IMP program turns the END flip flop on when it sets up the packet pointers for its *last* buffer load of a message to the Host.

The P pulse that steps the counter from 17_8 to 0000 shifts the last bit to the left end of the Buffer and turns on Bit Available. Thus on the last Bit, TYIB and LIBI, are sent to the Host together. Note, therefore, that while the IMP is interrupted after *each* of its buffer loads have gone to the Host, the Host gets LIBI only on the last bit of the *final* packet of the message. The Host's special interface unit must pad with trailing zeros to fill any remaining gap in its final word (see Chapter III above). When the Host turns off RFNIB for the last bit, a final P pulse turns off Bit Available (so that, even if a new RFNIB comes along, no TYIB will be returned). No shift or count takes place and DIL is not turned on. Instead, the LAST flip flop is shut off and the INT flip flop is turned on, thereby presenting an Interrupt request to the IMP. The IMP, in honoring the Interrupt, shuts off the INT flip flop with a OCP OFFINT, and when pointers are reset and a new buffer is ready to go, it issues a new OCP GO. The Host may or may not be waiting with RFNIB. When it is on, the new transfer will start.

4.0 IMP TO MODEM INTERFACING — DETAILED DESCRIPTION

Figure E-5 shows details of the Output logic. So long as the ON FF remains in the zero state, all control is clamped off and nothing happens. The modem's regular requests for bits will elicit no response and the data line to the modem will be held at logical zero.

At some point, the IMP program issues a TURNON command which sets the ON FF. This permits modem bit requests to make GMAB (Give Me A Bit) pulses. GMABs count C and when C reaches 000, a GØ pulse is produced which loads an 8 bit shift register, R_2, with a SYN character from a set of fixed input gates. Succeeding GMAB pulses shift

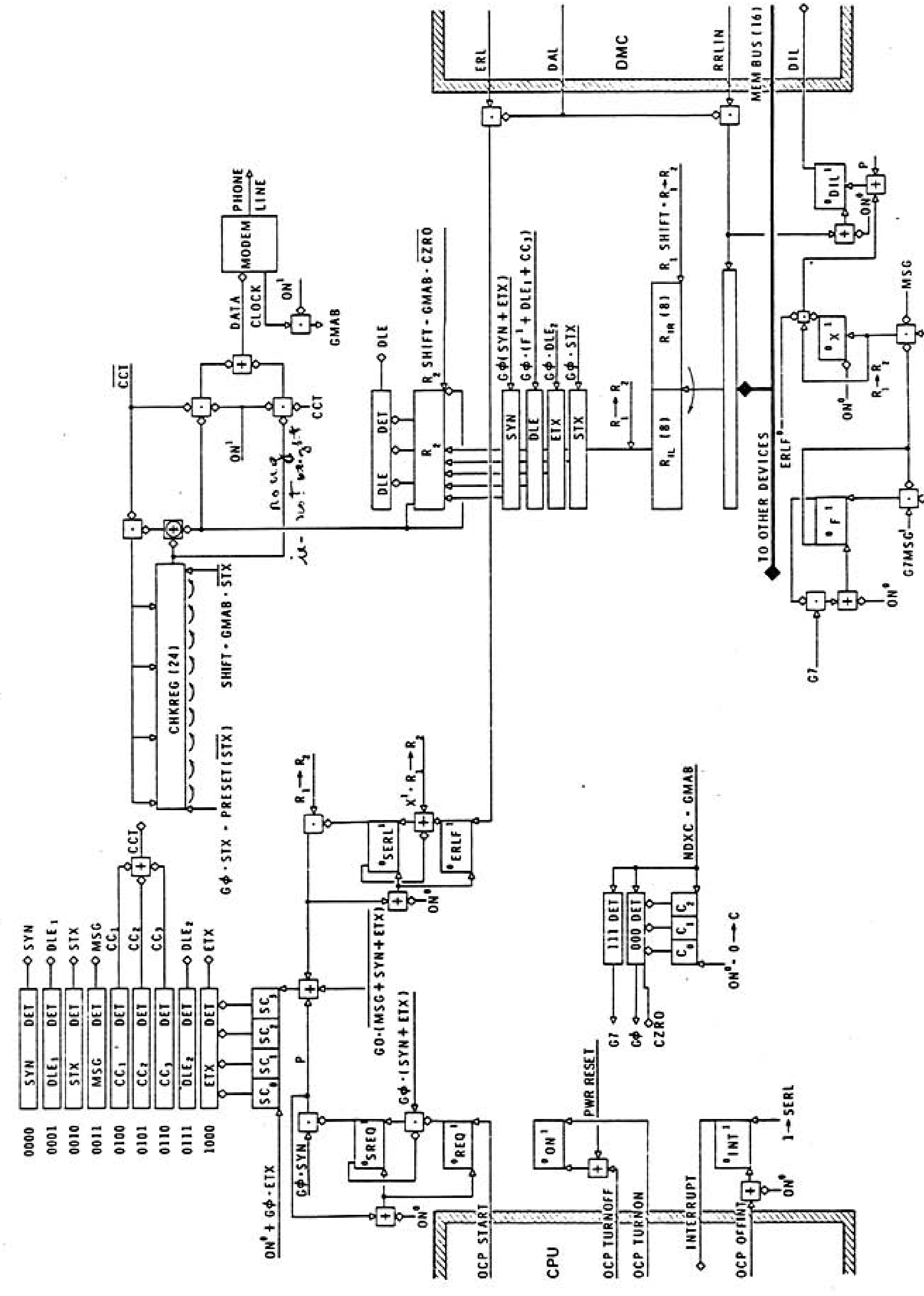


FIG. E-5 IMP/MODEM INTERFACE OUTPUT SECTION.

the SYN character from R_2 to the modem, meanwhile continuing the counting of C. Every 8 counts, the shift of R_2 is replaced by the load of a new SYN character. This constitutes the normal running mode of the Output section; i.e., an endless stream of SYN characters are fed onto the line. All of this takes place while a 4 bit state counter (SC) indicates the SYN state.

At some arbitrary and asynchronous point in this process, the IMP program executes a START command which indicates that a buffer load is ready for transmission and that the pointers for the DMC have been set. This START command sets a REQ (Request) flip flop.

The next GØ pulse sets SREQ (Synchronized Request) and the next GØ steps the state counter to the DLE state, clears the synchronizer, and sets the DIL flip flop which requests a word from the DMC. The answering RRLIN pulse from the DMC loads the contents of the memory output bus into a 16 bit buffer register, R_1. In the DLE state a DLE character is gated into R_2, from which it is shifted to the modem. The state counter steps to the STX state, an STX character is gated into R_2 for transmission and the state counter steps to the MSG state. At this point the check register is preset to a value such that shifting in the STX character will clear it. In the MSG state, an X flip flop alternates on successive characters to select the character time at which a new DMC request is to be made. When X is "zero", the left half of R_1 is gated to R_2 and is simultaneously replaced by the right half of R_1. When X is "one", the second character of the word thus goes to R_1 and the DIL flip flop is set to request another word from the DMC. The DMC has 160 μs to respond to the request (while the second character of the previous word shifts out of R_2 to the modem).

As this flow proceeds, a check sum is built up in a 24 bit check register which shifts in bits from the left end of R_2. At some point, when the IMP buffer reaches the last word, the DMC will return an ERL signal which sets the ERLF flip flop. As the second character of the last word goes into R_2, a SERL (Synchronized ERL) flip flop is set and an interrupt is presented to the computer. After the last bit has been shifted to the modem, the state counter steps to the CC_1 state. The contents of the Check Register are now gated to the modem in place of R_2 and this continues as the state counter advances through states CC_2 and CC_3. As the last bit of the check digits is shifted into the modem, gating is returned to R_2 which is simultaneously loaded with a DLE character. The state counter now reads DLE_2. After the DLE has been transmitted, the state counter advances to ETX and an ETX character is gated into R_2 and shifted to the modem. Finally, the state counter returns to the SYN state (i.e., is cleared) and the transmission of SYN characters resumes.

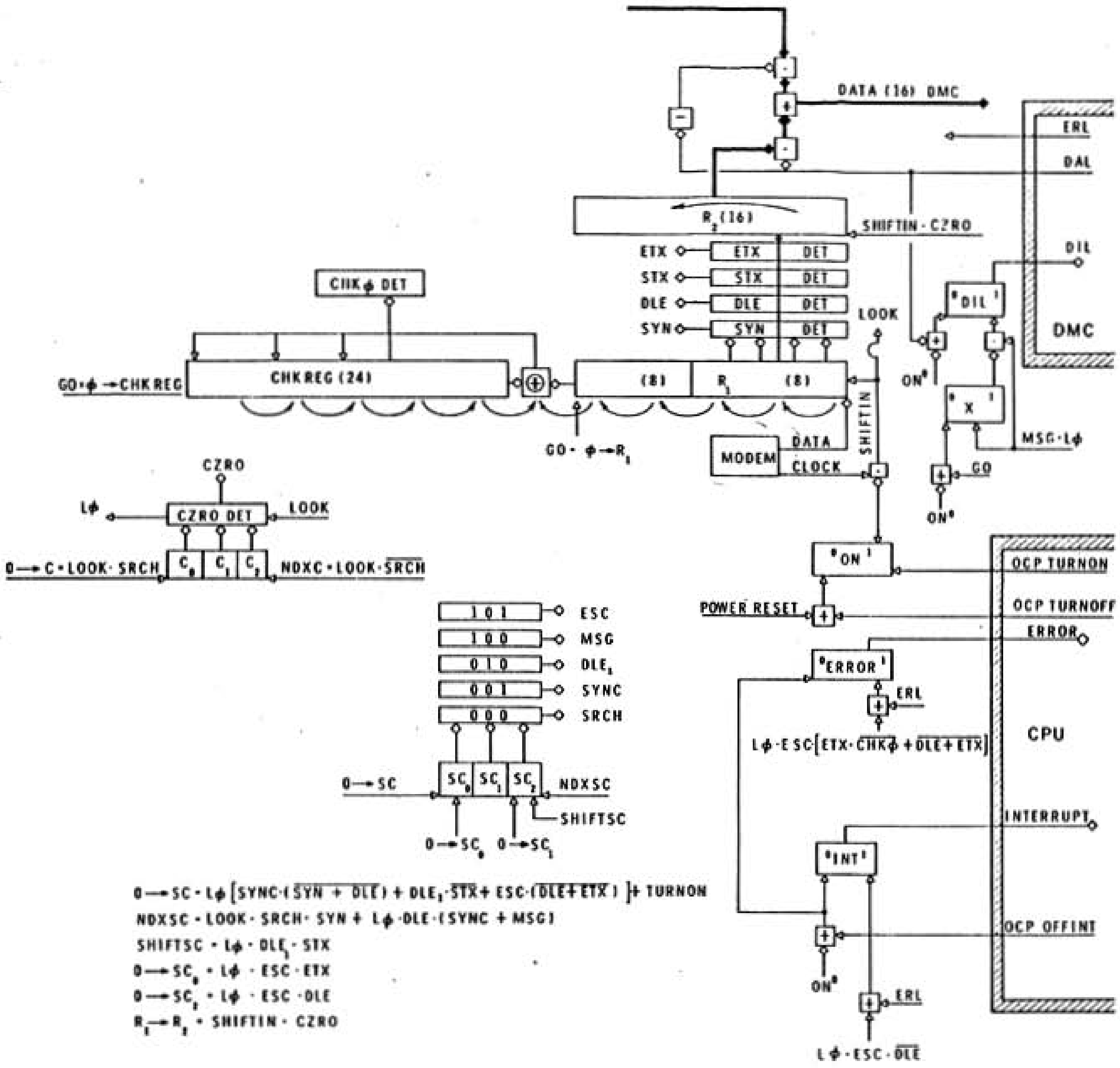
This process is complicated by the necessity of recognizing DLE characters while in the MSG state, i.e., in the interval between the STX and the ERL indication. The bits in R_2 not only are shifted to the modem and Check Register, but also cycle within R_2 itself. This allows a check to be made to determine if the character just transmitted was a DLE. This check is made one shift before the DLE returns to its original position within R_2. If a DLE is noted, an F flip flop is turned on. This interrupts the normal sequence of events while a second DLE is inserted in the data stream after the one just sensed. Sensing is, of course, inhibited on this inserted character itself. Then message flow resumes. Complementing of the X flip flop, which controls the procurement of words from memory, is inhibited on the inserted character, which means that the accessing of the next word from memory is effectively pushed back by a character.

Note that this procedure lengthens a packet containing DLE characters but since the hardware at the receiving IMP performs the inverse operation, subtracting out the extra DLE's, the effect appears only on the phone line.

5.0 MODEM TO IMP INTERFACING — DETAILED DESCRIPTION

Figure E-6 shows the details of the Input section. The IMP turns the input section on via a TURNON command which sets the state counter (SC) into the SRCH (Search) state. As the modem pumps in bits, they produce SHIFTIN pulses which shift a 16 bit shift register, R_1, together with a 24 bit check register.* After a small delay, a LOOK pulse is produced from each SHIFTIN. In the SRCH state, LOOK pulses hold a 3 bit counter, C, in the zero state. When a LOOK finds a SYN character in R_1, the state counter advanced to the SYNC state to acknowledge character sync. and C commences to count. Every eighth count, when C reads zero, an LØ pulse is produced from LOOK. In the SYNC state, anything other than a SYN or a DLE character appearing at LØ time will return the state counter to the SRCH state. Normally, the next event of interest is the arrival of a DLE at the front of a packet. This advances the state counter to the DLE_1 state whence anything other than a succeeding STX will act like a loss of sync. If the STX character appears correctly, the state counter shifts left one place to reach the MSG state and a GO pulse is produced which clears the Check Register and R_1. GO also initially clears the X flip flop which serves the same alternating function as the X flip flop in the output section. Once in the MSG state, when a character has finished shifting into R_1, it is moved into the right

*R_1, R_2, and the Check Register of the Input Section are not the same registers as those in the Output Section.



$0 \rightarrow SC = L\phi\left[SYNC\cdot(\overline{SYN + DLE}) + DLE_1\cdot\overline{STX} + ESC\cdot(\overline{DLE + ETX})\right] + TURNON$

$NDXSC = LOOK\cdot SRCH\cdot SYN + L\phi\cdot DLE\cdot(SYNC + MSG)$

$SHIFTSC = L\phi\cdot DLE_1\cdot STX$

$0 \rightarrow SC_0 = L\phi\cdot ESC\cdot ETX$

$0 \rightarrow SC_1 = L\phi\cdot ESC\cdot DLE$

$R_1 \rightarrow R_2 = SHIFTIN\cdot CZRO$

FIG. E-6 IMP/MODEM INTERFACE INPUT SECTION.

half of R_2 and the right half of R_2 simultaneously moves to the left half. Every two characters, the X flip flop reads "one" and sets DIL which presents a request for memory access to the DMC. Here again, the DMC has 160 μs to respond while the next character is assembled in R_1.

This process normally continues until a DLE character is sensed which advances the state counter to ESC. The ensuing ETX returns the state counter to the SYNC state, causes the zero detector on the check register to be sensed (setting an error flip flop if the register is non-zero), and produces an Interrupt. The unit then keeps vigil for commencement of the next message.

Should the DLE ETX at the end of a packet fail to be recognized, an ERL signal will be returned from the DMC when the allocated IMP buffer is full and Error and Interrupt signals will be generated.

The only remaining complexity is the deletion of the extra DLE characters inserted by the transmitting IMP. The insertion process guarantees that the DLEs (other than the opening and closing ones) occur in pairs. The first DLE steps into the ESC state and, while in this state, no memory requests (DILs) will be generated. Furthermore, the regular alternation of the X flip flop is inhibited to avoid counting the extra DLE. The second DLE (i.e., any DLE occurring in the ESC state) returns SC to the MSG state. While in the ESC state, a normal opportunity for transfer of R_2 into the IMP memory was missed and thus the next character from the right half of R_1 will replace the character in the right half of R_2, consequently deleting the overwritten DLE character in R_2.

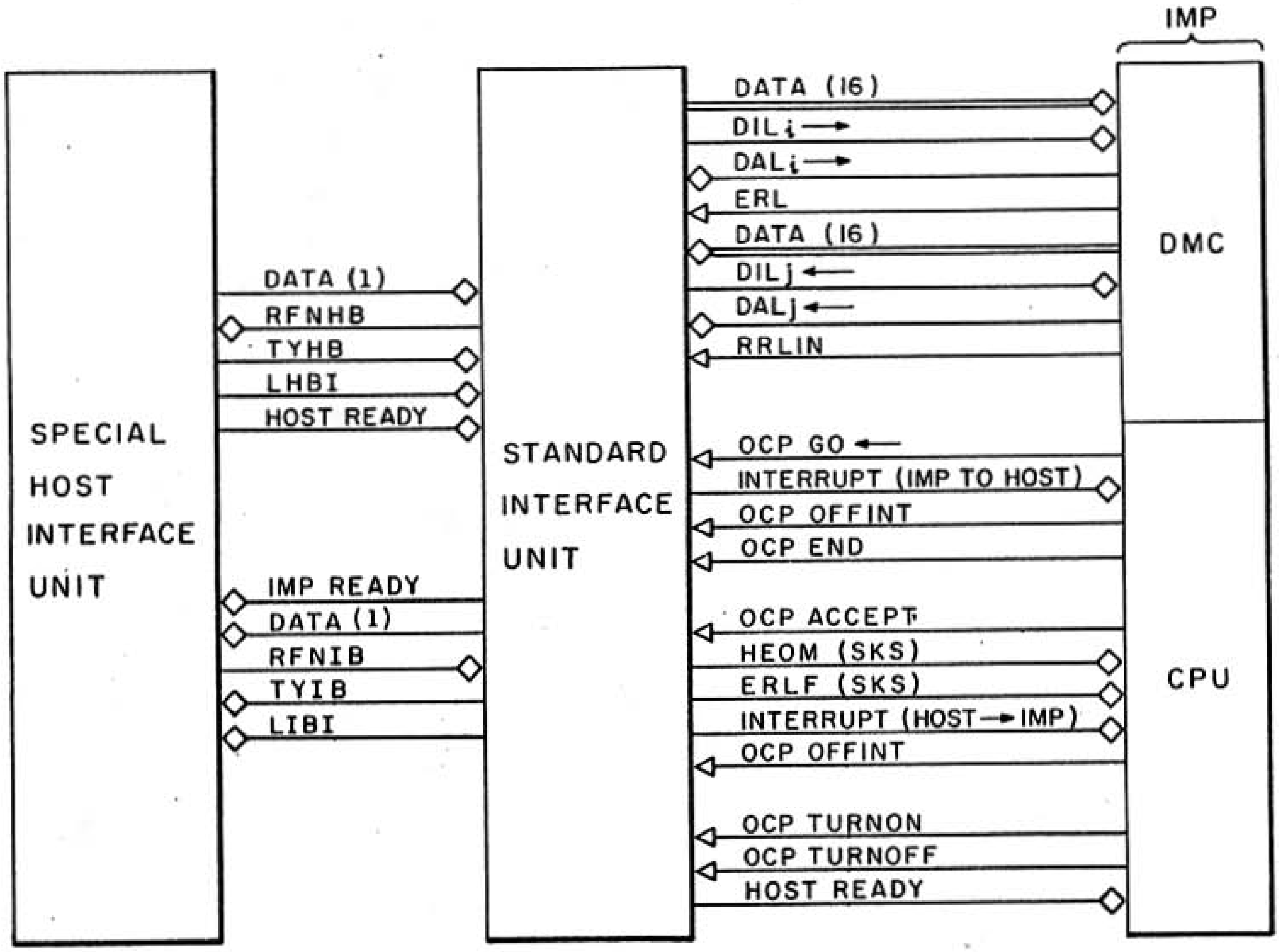


FIG. E-7 HOST/IMP INTERFACE LINES.

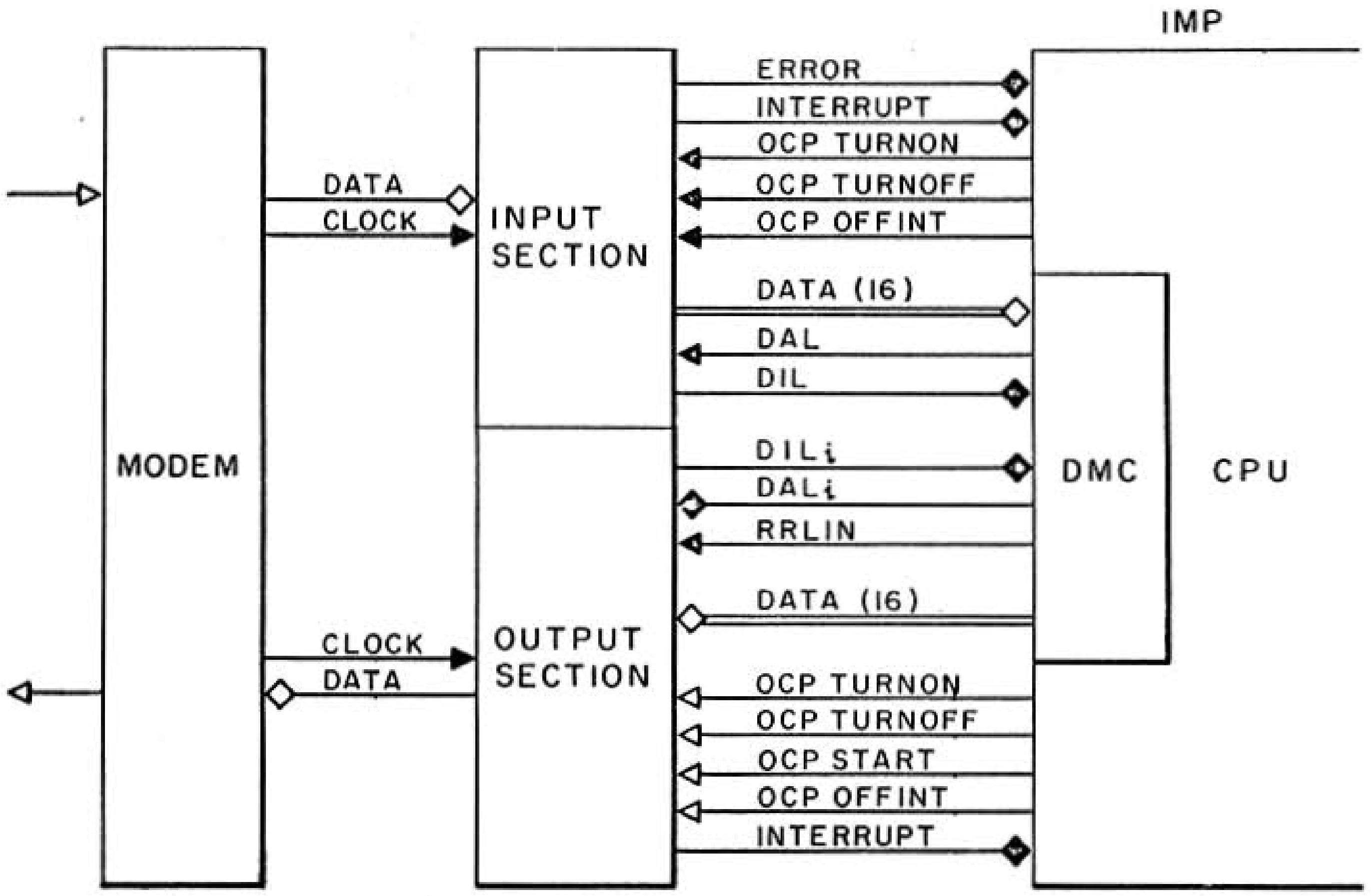


FIG. E-8 IMP/MODEM INTERFACE LINES.

6.0 ASSIGNMENT OF CHANNELS AND FUNCTIONAL MEANING OF PRIORITY INTERRUPTS —— in order of decreasing priority

Interrupt		
1	Modem 1 to IMP Interface:	packet reception completed or check sum error or failed to recognize end of packet before buffer region filled.
2	Modem 2 to IMP Interface:	packet reception completed or check sum error or failed to recognize end of packet before buffer region filled.
3	Modem 3 to IMP Interface:	packet reception completed or check sum error or failed to recognize end of packet before buffer region filled.
4	Modem 4 to IMP Interface:	packet reception completed or check sum error or failed to recognize end of packet before buffer region filled.
5	Modem 5 to IMP Interface:	packet reception completed or check sum error or failed to recognize end of packet before buffer region filled.
6	Modem 6 to IMP Interface:	packet reception completed or check sum error or failed to recognize end of packet before buffer region filled.
7	IMP to Modem 1 Interface:	packet transmission completed.
8	IMP to Modem 2 Interface:	packet transmission completed.
9	IMP to Modem 3 Interface:	packet transmission completed.
10	IMP to Modem 4 Interface:	packet transmission completed.
11	IMP to Modem 5 Interface:	packet transmission completed.

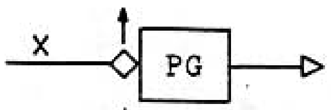
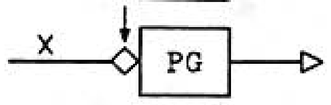
12	IMP to Modem 6 Interface:	packet transmission completed.
13	Host to IMP Interface:	specified IMP buffer region filled, or Host's Ready State changed, or Host indicated End of Message.
14	IMP to Host Interface:	Finished transmitting (packet) to Host.
15	Clock:	Self explanatory.
16	Program:	Program induced interrupt.

7.0 NOTES ON DRAWING SYMBOLOGY AND CONVENTIONS

1. Names are not always unique among drawings. Thus a line marked DAL on one drawing is a *different* DAL line from that shown on another drawing. On the other hand, as there is but one ERL line from the DMC, all lines called ERL are really the same. Each Interface Section has its own Interrupt flip flop, assigned to its own priority interrupt line, although all of these flip flops are labelled INT. Each also has its own OCP OFFINT line, etc.

2. Symbology:

$\boxed{\cdot}$	Logical AND
$\boxed{+}$	Logical OR
$\boxed{\oplus}$	Logical EXCLUSIVE OR
$\boxed{-}$	Logical NOT
⎍	Delay

	Device which generates a Pulse on the False to True Transition of X.
	Device which generates a Pulse on the True to False Transition of X.
$\overline{CZRO}$	Detector on Counter C shows contents ≠ Ø.
MSG^1	MSG Flip Flop is in the one state.

APPENDIX F: IMP PROGRAM DETAILS

1.0 FLOW CHARTS

The following flow charts detail the various functions of the IMP program routines. The flow charts are divided into five sections:

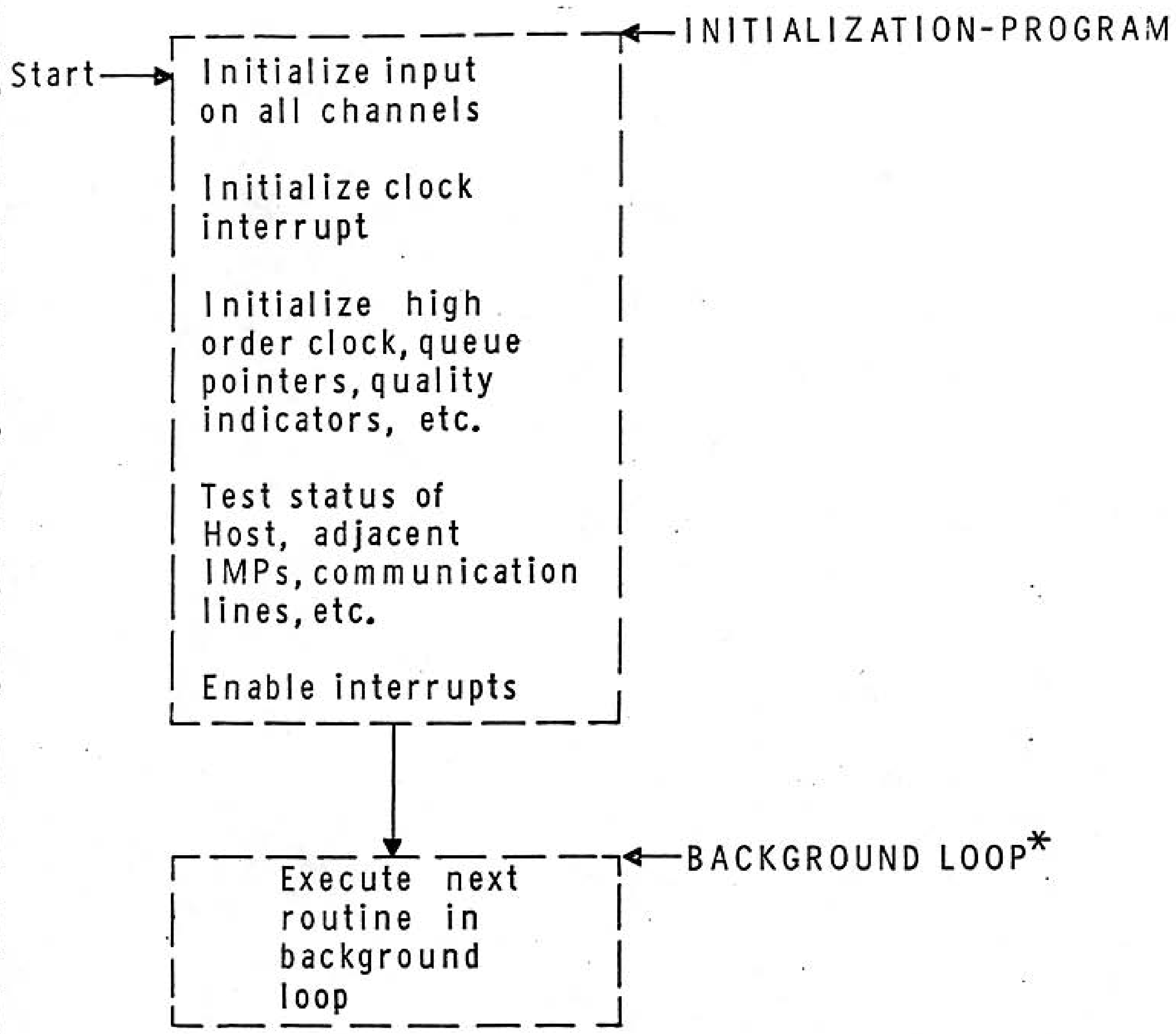
1) Initialization and background routines.
2) Interrupt routines.
3) Task routines (i.e., routines called indirectly, via the task list, by interrupt routines).
4) Input processing routines — a set of routines dispatched to by the NETWORK-INPUT task routine depending on the contents of the input packet.
5) Shared subroutines — a set of routines which are shared by various other routines. Interrupt routines which call a shared subroutine must insure that all other interrupt routines which call that shared subroutine are locked out. Locking and unlocking of interrupts for this purpose is not explicitly shown in the flow charts but it can be assumed to be included where appropriate.

The basic relations between these five sets of routines are outlined in Chapter III, and the reader should familiarize himself with that chapter before studying this section.

The following notational conventions are followed within the flow charts:

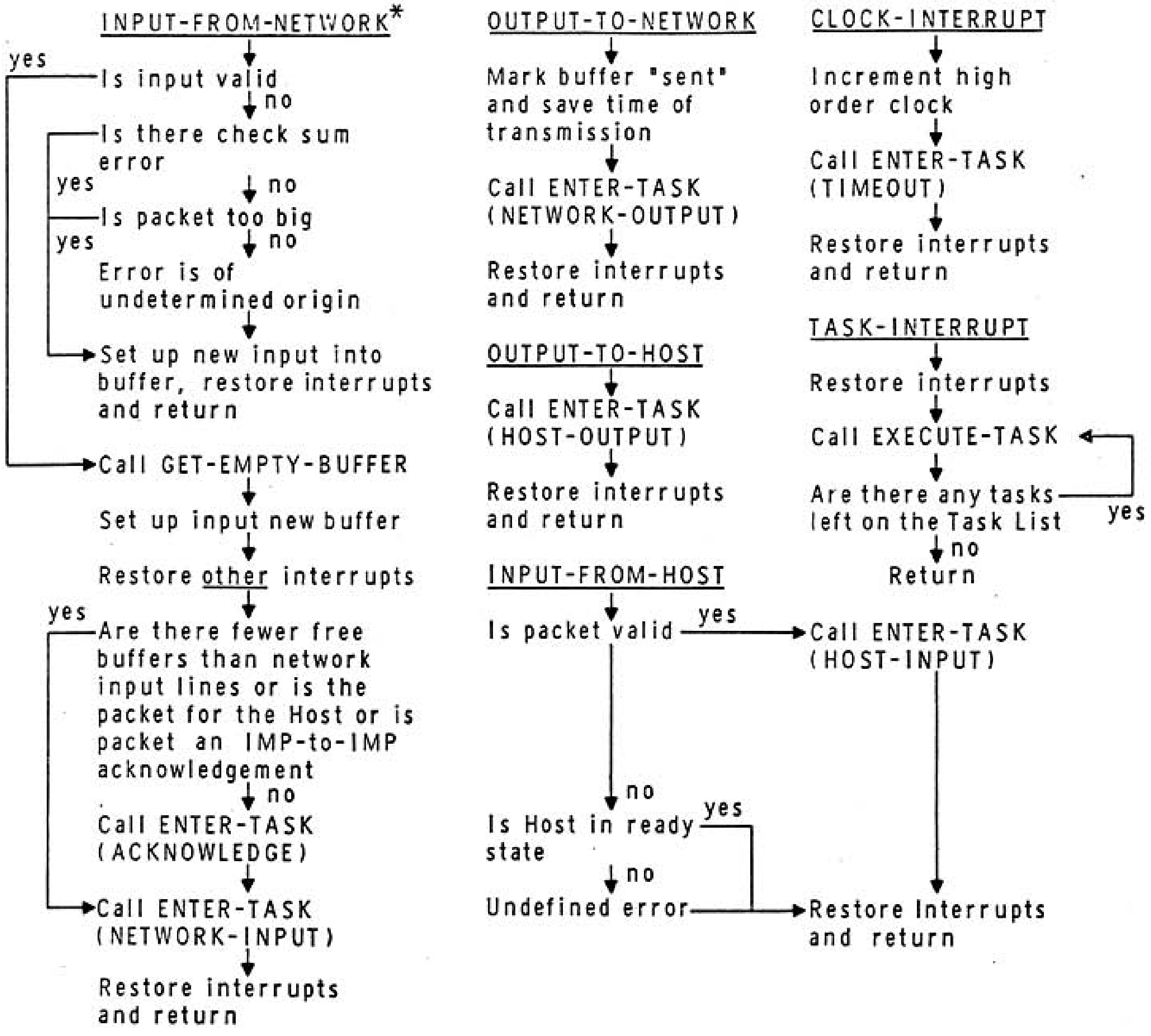
1) Subroutine calls are indicated by the word "Call" followed by the capitalized name of the called routine. Although many of

IMP PROGRAM FLOW CHARTS


INITIALIZATION-PROGRAM
Start
Initialize input on all channels
Initialize clock interrupt
Initialize high order clock, queue pointers, quality indicators, etc.
Test status of Host, adjacent IMPs, communication lines, etc.
Enable interrupts
BACKGROUND LOOP*
Execute next routine in background loop

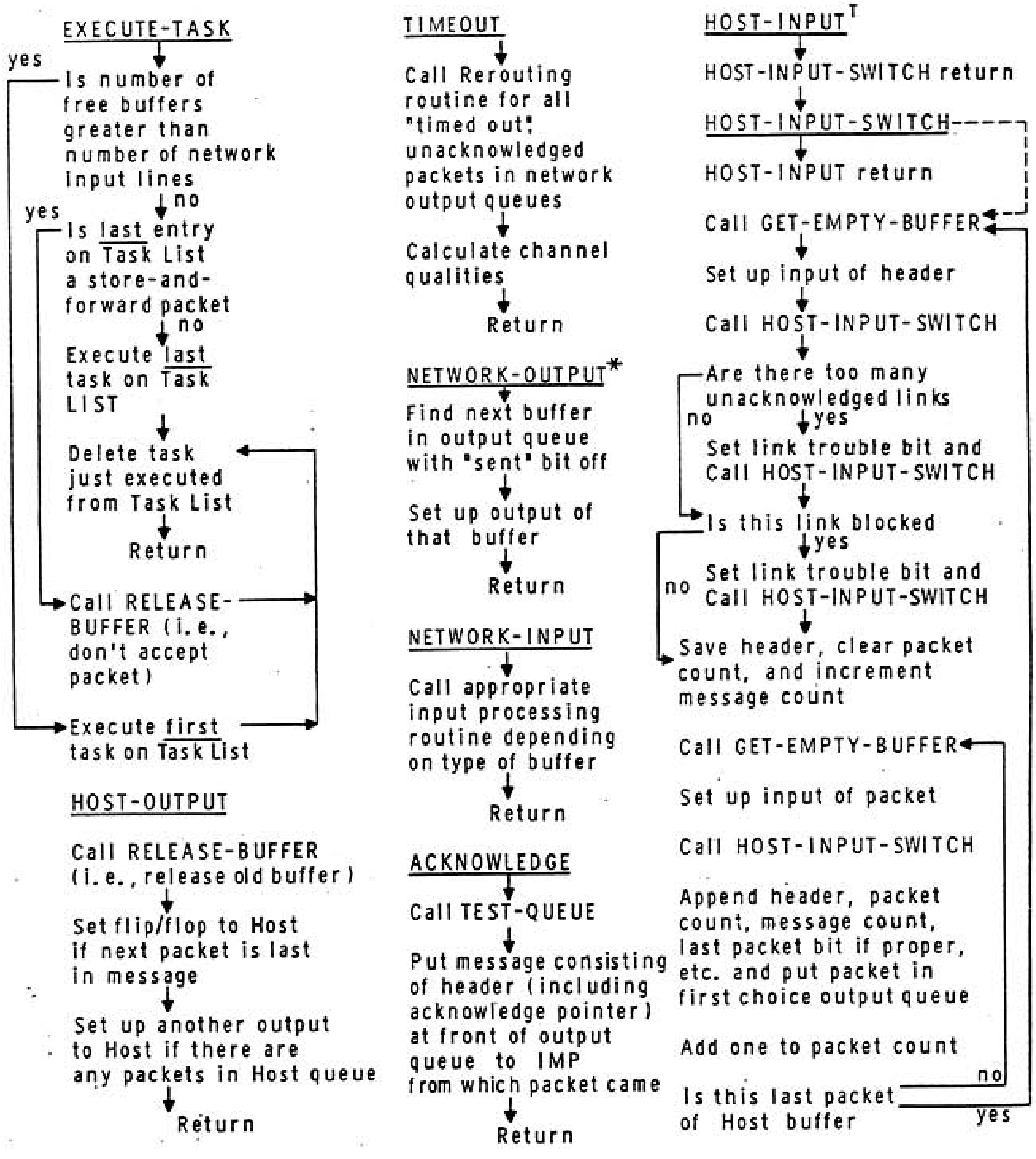
*Control is obtained from the Background Loop by way of interrupts.

FIG. F-1 INITIALIZATION PROGRAM AND BACKGROUND LOOP.


INPUT-FROM-NETWORK*
Is input valid
yes
no
Is there check sum error
yes
no
Is packet too big
yes
no
Error is of undetermined origin
Set up new input into buffer, restore interrupts and return
Call GET-EMPTY-BUFFER
Set up input new buffer
Restore other interrupts
Are there fewer free buffers than network input lines or is the packet for the Host or is packet an IMP-to-IMP acknowledgement
yes
no
Call ENTER-TASK (ACKNOWLEDGE)
Call ENTER-TASK (NETWORK-INPUT)
Restore interrupts and return
OUTPUT-TO-NETWORK
Mark buffer "sent" and save time of transmission
Call ENTER-TASK (NETWORK-OUTPUT)
Restore interrupts and return
OUTPUT-TO-HOST
Call ENTER-TASK (HOST-OUTPUT)
Restore interrupts and return
INPUT-FROM-HOST
Is packet valid
yes
no
Is Host in ready state
yes
no
Undefined error
CLOCK-INTERRUPT
Increment high order clock
Call ENTER-TASK (TIMEOUT)
Restore interrupts and return
TASK-INTERRUPT
Restore interrupts
Call EXECUTE-TASK
Are there any tasks left on the Task List
yes
no
Return
Call ENTER-TASK (HOST-INPUT)
Restore Interrupts and return

*There is a separate copy of the INPUT-FROM-NETWORK routine for each network input channel: the other input and output interrupt routines will probably be parameterized.

FIG. F-2 INTERRUPT ROUTINES.


EXECUTE-TASK
Is number of free buffers greater than number of network input lines
yes
no
Is last entry on Task List a store-and-forward packet
yes
no
Execute last task on Task LIST
Delete task just executed from Task List
Return
Call RELEASE-BUFFER (i.e., don't accept packet)
Execute first task on Task List
HOST-OUTPUT
Call RELEASE-BUFFER (i.e., release old buffer)
Set flip/flop to Host if next packet is last in message
Set up another output to Host if there are any packets in Host queue
Return
TIMEOUT
Call Rerouting routine for all "timed out" unacknowledged packets in network output queues
Calculate channel qualities
Return
NETWORK-OUTPUT*
Find next buffer in output queue with "sent" bit off
Set up output of that buffer
Return
NETWORK-INPUT
Call appropriate input processing routine depending on type of buffer
Return
ACKNOWLEDGE
Call TEST-QUEUE
Put message consisting of header (including acknowledge pointer) at front of output queue to IMP from which packet came
Return
HOST-INPUT†
HOST-INPUT-SWITCH return
HOST-INPUT-SWITCH
HOST-INPUT return
Call GET-EMPTY-BUFFER
Set up input of header
Call HOST-INPUT-SWITCH
Are there too many unacknowledged links
no
yes
Set link trouble bit and Call HOST-INPUT-SWITCH
Is this link blocked
yes
no
Set link trouble bit and Call HOST-INPUT-SWITCH
Save header, clear packet count, and increment message count
Call GET-EMPTY-BUFFER
Set up input of packet
Call HOST-INPUT-SWITCH
Append header, packet count, message count, last packet bit if proper, etc. and put packet in first choice output queue
Add one to packet count
Is this last packet of Host buffer
no
yes

*Input and Output Task Routines are parameterized (i.e., there is a single copy of the routine which is called with a parameter which indicates which channel the routine should be concerned with.

†The first four lines of HOST-INPUT are, in effect, a multi-way switch. The HOST-INPUT-SWITCH return is initialized to "return" control to the line indicated by the dotted arrow.

FIG. F-3 TASK ROUTINES

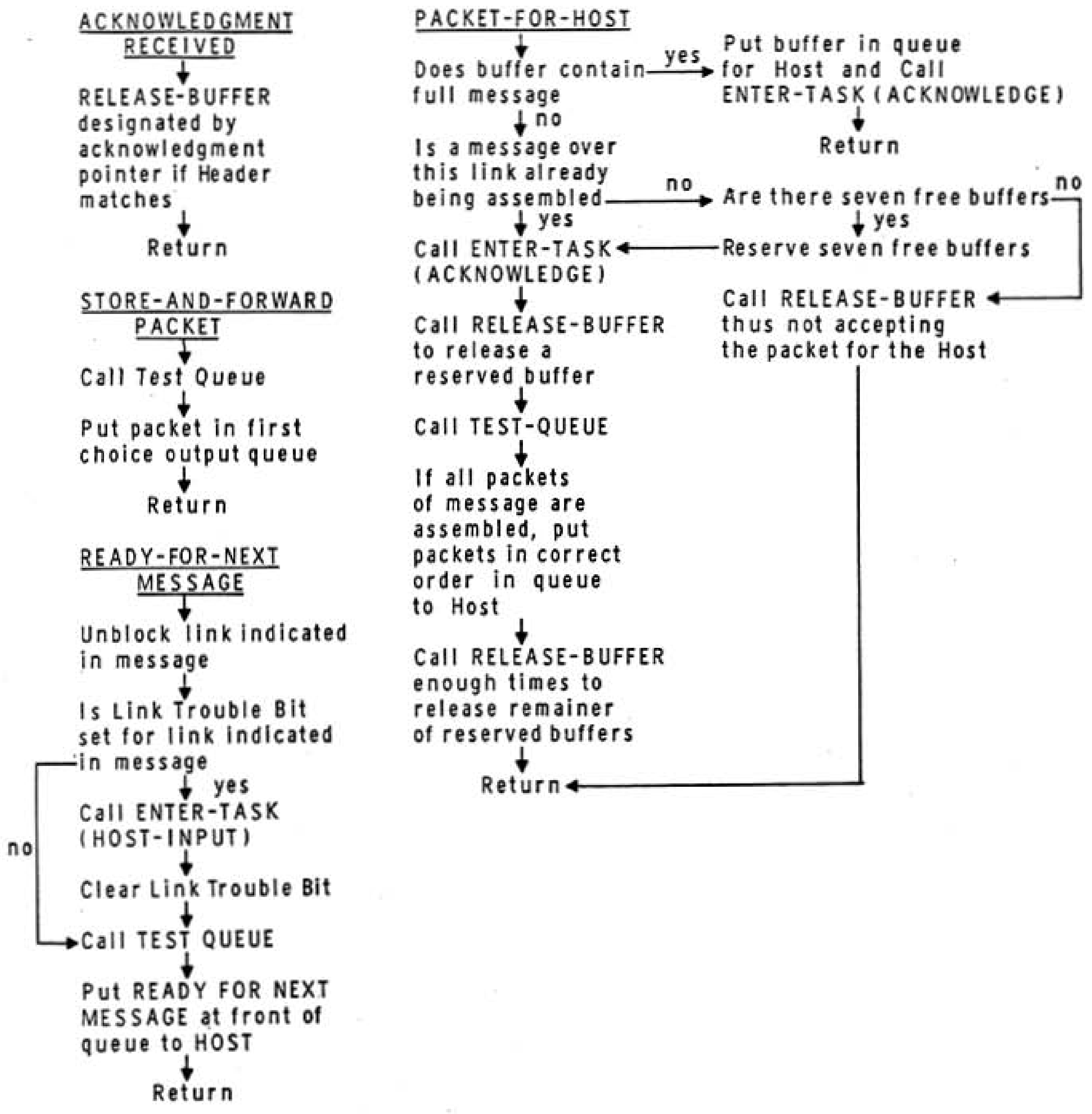
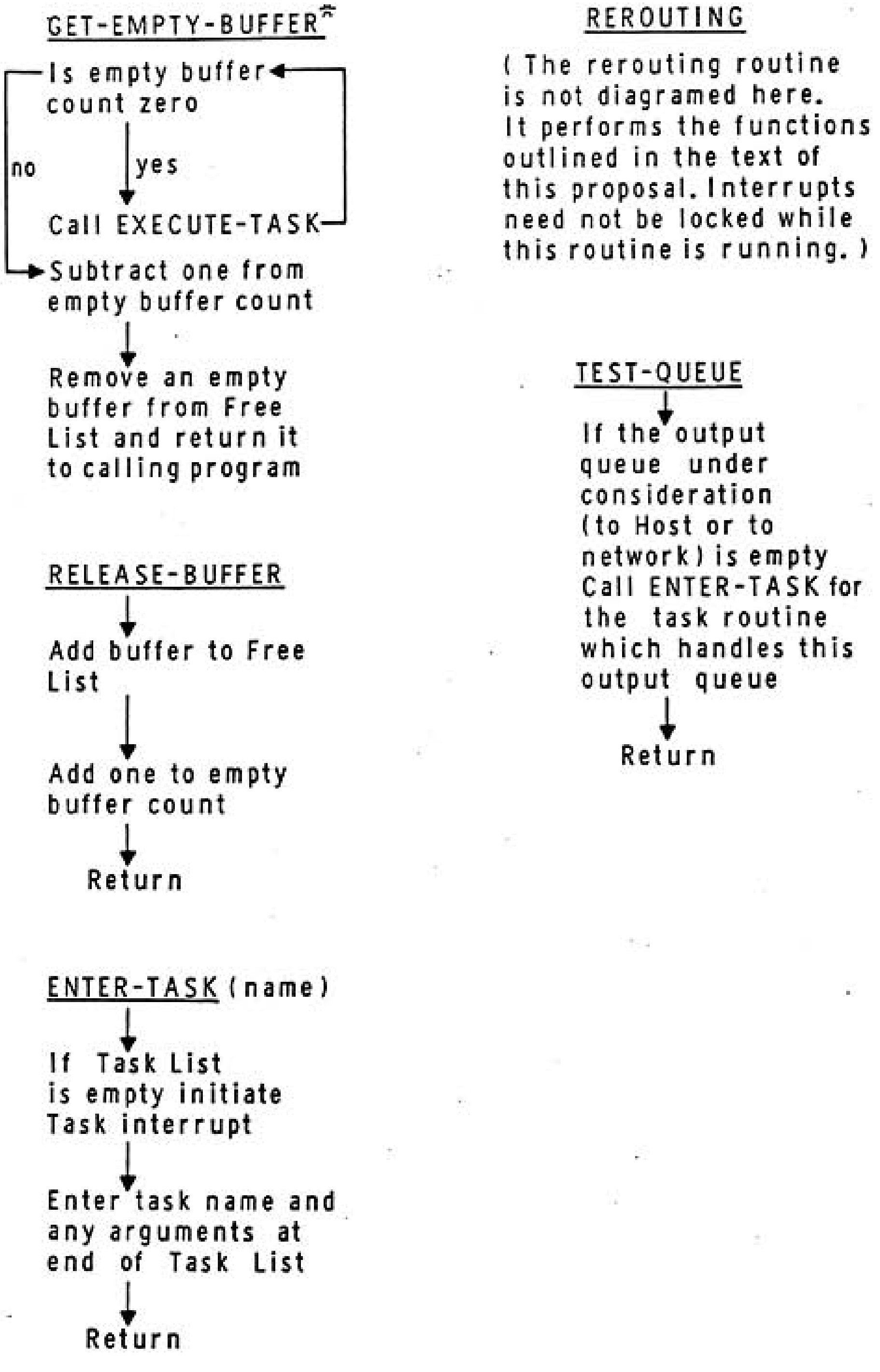

ACKNOWLEDGMENT RECEIVED
RELEASE-BUFFER designated by acknowledgment pointer if Header matches
Return
STORE-AND-FORWARD PACKET
Call Test Queue
Put packet in first choice output queue
Return
READY-FOR-NEXT MESSAGE
Unblock link indicated in message
Is Link Trouble Bit set for link indicated in message
yes
no
Call ENTER-TASK (HOST-INPUT)
Clear Link Trouble Bit
Call TEST QUEUE
Put READY FOR NEXT MESSAGE at front of queue to HOST
Return
PACKET-FOR-HOST
Does buffer contain full message
yes
Put buffer in queue for Host and Call ENTER-TASK (ACKNOWLEDGE)
Return
no
Is a message over this link already being assembled
no
Are there seven free buffers
no
yes
Reserve seven free buffers
yes
Call ENTER-TASK (ACKNOWLEDGE)
Call RELEASE-BUFFER thus not accepting the packet for the Host
Call RELEASE-BUFFER to release a reserved buffer
Call TEST-QUEUE
If all packets of message are assembled, put packets in correct order in queue to Host
Call RELEASE-BUFFER enough times to release remainer of reserved buffers
Return

FIG. F-4 INPUT PROCESSING ROUTINES


GET-EMPTY-BUFFER*
Is empty buffer count zero
no
yes
Call EXECUTE-TASK
Subtract one from empty buffer count
Remove an empty buffer from Free List and return it to calling program
RELEASE-BUFFER
Add buffer to Free List
Add one to empty buffer count
Return
ENTER-TASK (name)
If Task List is empty initiate Task interrupt
Enter task name and any arguments at end of Task List
Return
REROUTING
(The rerouting routine is not diagramed here. It performs the functions outlined in the text of this proposal. Interrupts need not be locked while this routine is running.)
TEST-QUEUE
If the output queue under consideration (to Host or to network) is empty Call ENTER-TASK for the task routine which handles this output queue
Return

*Any further interrupts that might cause these routine to be called must be locked out when these routines are called

FIG. F-5 SHARED SUBROUTINES

the routines have arguments, these are not explicitly indicated in the flow charts. An exception is the routine "ENTER-TASK," one of whose arguments, the name of the task to be put on the Task List, is explicitly shown in parentheses.

2) The work "network" when used in the flow charts, refers to the IMP-modem interface, while the word HOST refers to the IMP-Host interface.

3) The start of a subroutine is indicated by the Capitalized and underlined name of the subroutine.

4) Every subroutine is assumed to store and restore any active registers which it uses, including the state of the interrupt priorities. These functions are not explicitly shown.

2.0 QUEUES

It is stated in Chapter III that output queues are doubly-linked circular lists with a list pointer denoting the "head" of the list. We have made all system lists of the doubly-linked cyclic variety, since the space and simplicity advantages gained by having only one set of list handling machinery are significant. Also we gain the versatility of being able to insert or delete an entry of any list rapidly, and to make insertions easily at either the head or tail of any list.

The following diagram of the buffer storage area of the IMP computer shows the structure of the task list, a sample output queue, and the free buffer list. Each buffer has two words reserved at the front for forward and backward pointers as well as (probably) two additional words indicating the current status of the buffer

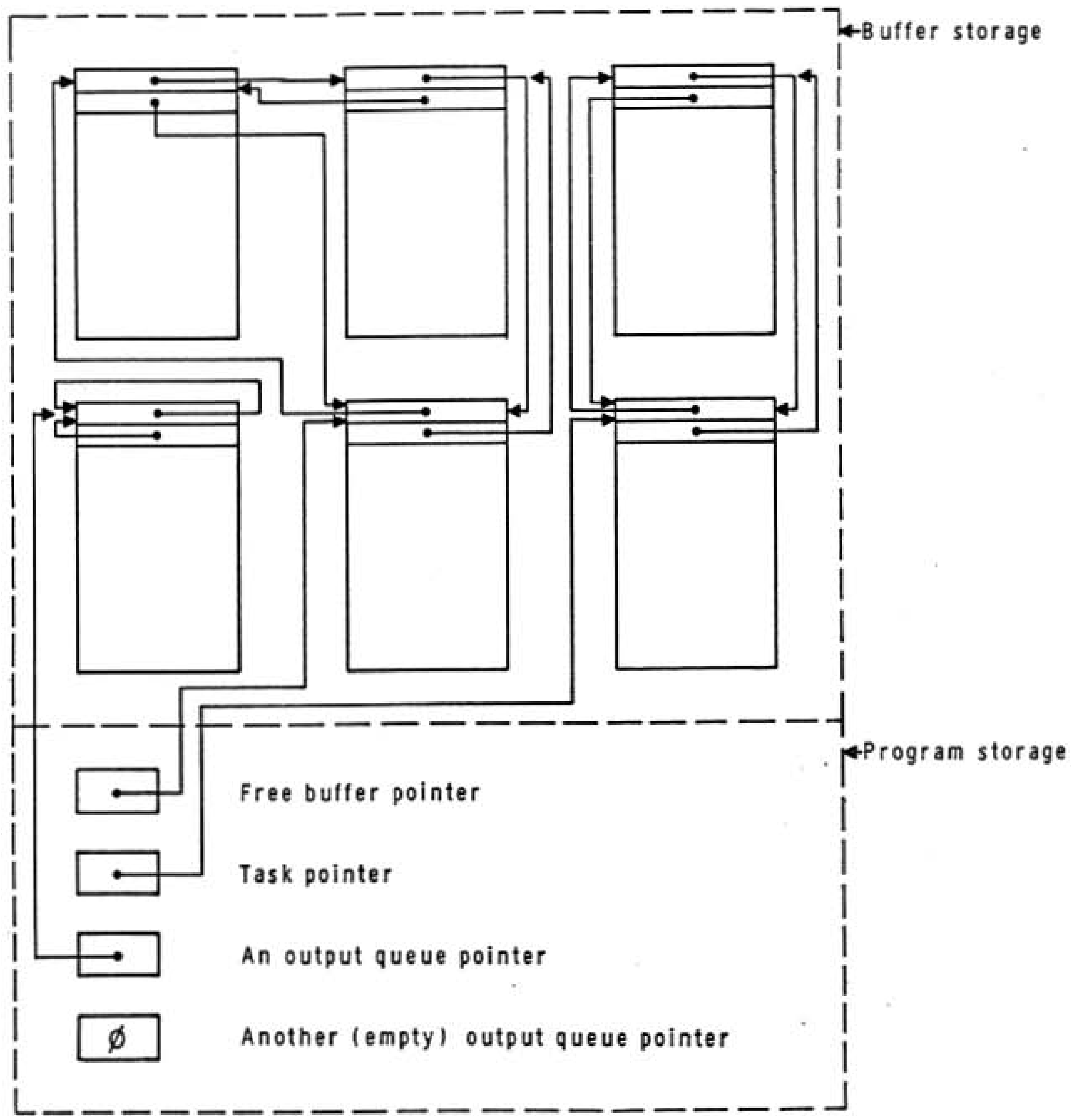


FIG. F-6 STRUCTURE OF BUFFER STORAGE.

(e.g., "sent," time of transmission). All buffers are on one and only one list at a time. Buffers which are on no other list are on the free buffer list. Since the task list is also chained through the buffer storage (one of the status words will indicate which task is to be performed on the buffer), the buffer storage may be completely shared by all variable length lists and there is no need to reserve in advance a fixed amount of storage for each list.

3.0 EXAMPLE

To conclude this appendix, we trace through the program logic which handles a store and forward packet coming in on channel A and destined for some other Host N. We start by assuming that the program is cycling in the background loop and that an input buffer is assigned on each input channel.

Upon receipt of an input interrupt from channel A, the channel A INPUT-FROM-NETWORK interrupt routine (Fig. F-2) is called. A new input buffer is assigned and ACKNOWLEDGE and NETWORK-INPUT tasks are put on the task list (i.e., an IMP-to-IMP acknowledge must be sent and the packet must be processed). The first call to ENTER-TASK (for ACKNOWLEDGE) finds the task list is empty and initiates the task interrupt. When the INPUT-FROM-NETWORK routine returns to the background loop and reenables interrupts, the task interrupt is serviced and the TASK-INTERRUPT routine (Fig. F-2) is called. TASK-INTERRUPT calls EXECUTE-TASK which executes the first task on the task list. Thus the ACKNOWLEDGE task is called which calls TEST-QUEUE (Fig. F-5) with reference to output channel A. TEST-QUEUE finds that the channel A output queue is empty and therefore calls ENTER-TASK with the NETWORK-OUTPUT routine for channel A as a task. This initiates the output and arms the output interrupt. TEST-QUEUE returns to ACKNOWLEDGE which puts an IMP-to-IMP acknowledgment on the channel A output queue and returns to EXECUTE-TASK. EXECUTE-TASK deletes the ACKNOWLEDGE task from the Task List and returns to TASK-INTERRUPT. Since there are more tasks on the Task List, TASK-INTERRUPT again calls EXECUTE-TASK which in turn calls NETWORK-INPUT (Fig. F-3). NETWORK-INPUT notes that the packet which came in on channel A is a store and forward packet and calls the STORE-AND-FORWARD input processing routine (Fig. F-4).

STORE-AND-FORWARD calls TEST-QUEUE with reference to the first choice output queue to destination N, thus putting a NETWORK-OUTPUT task for that queue on the task list. STORE-AND-FORWARD returns to INPUT-FROM-NETWORK and thus to EXECUTE-TASK since there are still two tasks on the task list. The first task on the list is the NETWORK-OUTPUT (Fig. F-3) task on channel A. This sets up an output of the IMP-to-IMP acknowledge. The second call to EXECUTE-TASK finds another NETWORK-OUTPUT task, this time for the first choice queue to destination N. Since there are no further tasks on the task list, TASK-INTERRUPT will return control to the background loop. As each of the above outputs are completed, the OUTPUT-TO-NETWORK interrupt routine is called. Each of these calls will result in a NETWORK-OUTPUT task being entered on the task list. When NETWORK-OUTPUT is run, no unsent buffers will be found in either output queues, so the output interrupt sequence for these queues will be broken until restarted, as before, by TEST-QUEUE.

Eventually an IMP-to-IMP acknowledge should return on the first choice channel to destination N. The INPUT-FROM-NETWORK interrupt routine will ENTER-TASK(NETWORK-INPUT) when this happens, which will call ACKNOWLEDGMENT-RECEIVED. ACKNOWLEDGMENT-RECEIVED will delete the acknowledged buffer from the output queues. Should an acknowledgment not be received within some time out period the CLOCK-INTERRUPT (Fig. F-2) routine will be called. It will set up the TIME-OUT task which will call the REROUTING (Fig. F-5) routine for the unacknowledged packet.

APPENDIX G: DETAILS ON TESTING

Insofar as testing is concerned, IMP development can be divided into four phases. During the first or prototype phase, test equipment and programs will be developed, as previously untried equipment is shaken down, to the point where designs of equipment and programs usable for testing of subsequent units result. During the second or production phase, the emphasis is on realizing production units for the field installation phase. Production units will be tested using a prototype unit as a test set. In the third, or field installation phase, units will operate in a self-test mode by providing their own test signals for proving operability. In the fourth or network mode, units will operate with real Host and line inputs and outputs, and will be capable of detecting operational malfunctions and reverting selectively to loop test modes for diagnosis of trouble. Thus, each phase builds on tests developed in the previous phase.

Describing the tests and test sequences requires describing sources and receivers of signals in long chains of equipment and distinguishing between chains as the testing complexity builds up. To simplify and clarify test descriptions, the following notation will be used. Capital letters with numerical subscript will denote system elements to be interconnected. Examples are: P_1, the first prototype IMP; I_1, first production IMP; M_1, first modem; L_1, first phone line; H_1, first Host. No subscript denotes a general element; lower case letters with subscript i or o denote interface equipment for input or output, respectively, to their associated system element. Examples are: c_i, input interface to IMP from carrier; h_i, input interface to IMP from Host; n_i, input interface to Host from IMP. Test equipment such as a scope will be written out: scope. Passage of signals from one element to another in a test chain will be

IMP, and its own retransmission IMP as follows:

$$\begin{tikzcd} h_i \arrow[r] & I_1 \arrow[r] & c_o \arrow[r] & c_i \arrow[r] & I_1 \arrow[r] \arrow[lll, bend right, "\text{retransmission}"'] & h_o \arrow[lllll, bend left] \end{tikzcd}$$

Receiving Host
Transmitting IMP

Transmitting Host
Receiving IMP

M looped on itself or M $\rightarrow$ L looped on itself may be added to the IMP self-loop for testing carrier operation.

1.1 Production Phase

During the production phase, production IMPs will be accepted into the BBN test facility with the P_1 prototype generating test signals. Each production IMP will be run through the following sequence of tests bringing it up to the IMP self-test level ready for shipment.

Single Interface Tests

$P_1 \rightarrow h_o \rightarrow h_i \rightarrow I$, $I \rightarrow h_o \rightarrow h_i \rightarrow P_1$, $P_1 \rightarrow c_o \rightarrow c_i \rightarrow I$, $I \rightarrow c_o \rightarrow c_i \rightarrow P_1$.

Modem Interface Tests

$P_1 \rightarrow c_o \rightarrow M \rightarrow c_i \rightarrow I$, $I \rightarrow c_o \rightarrow M \rightarrow c_i \rightarrow P_1$.

Operational Test

$P_1 \rightarrow h_o \rightarrow h_i \rightarrow I \rightarrow c_o \rightarrow M \rightarrow c_i \rightarrow P_1 \rightarrow c_o \rightarrow c_i \rightarrow I \rightarrow h_o \rightarrow h_i \rightarrow P_1$.

IMP Self-Test

$$\begin{tikzcd} h_i \arrow[r] & I \arrow[r] & c_o \arrow[r] & M \arrow[r] & c_i \arrow[r] & I \arrow[r] & h_o \arrow[llllll, bend left] \end{tikzcd}$$

1.2 Installation Phase

Prior to shipment of an IMP to a Host site, the Host computer will have been provided with a network interface that will interface the standard IMP-Host interfaces. Testing of the network interfaces of the Host will be similar to prototype testing, namely, initial output interface test, Host loop-test, and Host self-test. Symbolically the sequence is $H \rightarrow n_o \rightarrow$ scope, $H \rightarrow n_o \rightarrow n_i \rightarrow H$ (test program), $H \rightarrow n_o \rightarrow n_i \rightarrow H$ (self-test). The IMP will be shipped when the Host has completed interface testing using the test program only, because developing the complete Host self-test capability is more easily accomplished after initial operation with IMP when the Host operational program has been checked out in operation. Host self-test is necessary for network operation to permit fault isolation between the IMP and the Host. When the IMP has been installed, it will be tested in IMP self-test mode without modems, then with modems, then with the Host using Host and IMP test messages, and finally with the Host using Host operational program in a loop. The sequence of tests is:

$$\begin{tikzcd} h_i \arrow[r] & I \arrow[r] & c_o \arrow[r] & c_i \arrow[r] & I \arrow[r] \arrow[lll, bend right] & h_o \arrow[lllll, bend left] \end{tikzcd},\quad \begin{tikzcd} h_i \arrow[r] & I \arrow[r] & c_o \arrow[r] & M \arrow[r] & c_i \arrow[r] & I \arrow[r] \arrow[llll, bend right] & h_o \arrow[llllll, bend left] \end{tikzcd},$$

$H \rightarrow n_o \rightarrow h_i \rightarrow I \rightarrow h_o \rightarrow n_i \rightarrow H$, and

$$\begin{tikzcd} H \arrow[r] & n_o \arrow[r] & h_i \arrow[r] & I \arrow[r] & c_o \arrow[r] & M \arrow[r] & c_i \arrow[r] & I \arrow[r] \arrow[llll, bend right] & h_o \arrow[r] & n_i \arrow[r] & H \end{tikzcd}.$$

APPENDIX H: SIMULATION

In this Appendix we describe a LISP program written to simulate and display the operation of the ARPA network. This network consists of a collection of nodes linked to various neighbors.

In operation, a user requests that a message be sent from one node to another, and the program then displays the progress of this message from its creation to the final acknowledgment of its receipt. Since more than one message can be sent at a time, and each node has a limited capacity, the network can become clogged. The program is designed to serve as a tool for studying the ways in which networks become congested, for studying queueing problems, and for exploring routing algorithms (and heuristics) for handling these situations.

1.0 Nodes, Lines, and Messages

There are three distinct entities in the program's data structure: nodes, lines, and messages. Each of these consists of an array containing relevant information. For example, associated with each node is its position, its capacity, its neighbors, and its outstanding messages. Associated with each line is its length, and an indication of whether it is in use. Associated with each message is its origin, its destination, its current position, and an identifying number. Currently, each node has 14 entries, each message 11, and each line, 6. The organization of the program makes it easy to add new information for more sophisticated applications. The program is thus definitely not *frozen*.

Each message consists of a number of smaller units called packets. Every packet must reach its final destination before the message is considered received, although each packet can go a different route and arrive at a different time. Since only one packet can be sent on a given line (in a given direction) at a time, there are usually a number of packets waiting at a node to go out on one or more of its lines.

Packets of a single message are especially likely to bunch up because they all follow the same route, namely, the shortest one. The user does have the option to reroute packets, and thus reduce their waiting time at a given node, but this is done at the expense of making the packets travel a greater distance, and hence takes a longer time. We plan to experiment with different algorithms for deciding when to reroute packets automatically, depending on congestion at a node and the extra distance required by alternate routes.

When a packet arrives at a node, it is placed in the appropriate queue to go out on the line to the next node along its route. When a line clears, the next packet is sent. The length of time to travel the line is proportional to its length. (The user can alter this constant of proportionality effectively to speed up or slow down the changing display.) When the packet arrives at the next node, the line becomes clear, and another packet may be sent.

Even though a packet arrives at its ultimate destination, the rest of the packets in the message must arrive before the message is considered to be received. When all packets have arrived, they "enter the node" and disappear from the network. In addition, an identifying message is typed on the teletype, and an acknowledging message sent from the node to the originator of the message. This message consists of a single packet which travels at a much faster rate than regular packets, and immediately goes to the front of the line at each node.

Each node has a limited capacity which is the total number of packets it can accommodate (i.e., those waiting in line to go out) plus those waiting to enter the node. When this number is reached, the node refuses to accept any more packets. In terms of the program, this means that packets sent to that node will not arrive and will not be taken off of the queue of the node sending them. They will thus be sent continually until, ultimately, they are accepted. The capacity of a node can be varied dynamically by the user.

The user also has the option of disconnecting a given line, i.e., making it permanently busy. This feature is included in the program primarily to provide a way of quickly congesting nodes to then study how they may become uncongested.

2.0 The Display

The display generated and maintained by the program shows the current status of all nodes and lines in the network, and the position of those packets that are actually in transit. More detailed information is available in response to specific user queries — see Section 3.0 below.

Each node is displayed as a small circle with its name immediately above it. Above the name are two numbers: the number of packets holding at the node (waiting to enter the Host), and the number of packets in queues at the node (waiting to go out on one of its lines). The first is displayed slightly to the left of the second.

Each line on which a packet is currently being transmitted is displayed as a sequence of short dashed lines much brighter than the rest of the display. The message number, and packet number, is also displayed beside the line. When this packet arrives at the next node, the line is turned off and displayed as a much dimmer dotted line. If the packet is accepted (i.e., the capacity of the next node *not* reached), the number in queues of the sending node is decremented and either the number in queues or the number in holding of the receiving node is incremented. These changes may not occur simultaneously because of the order in which the program does things.

Message numbers begin with 1 and increase by 1. Packet numbers are really letters and begin with A and progress through the alphabet. Thus, if the third message sent consisted of ten packets, they would be labeled 3A through 3J. The acknowledging message packet has the message number and the symbol *. Thus, when all of the packets of message 3 reach their destination, a packet labelled 3* will be sent to the origin of the message, and the number in holding at the destination node will be decreased by the number of packets in the message.

If a node is at capacity, and a packet is sent to it, the line along which the packet is being sent will be brightened, and the packet and message number displayed. After the time normally required to send the packet over this line has elapsed, the line will be returned to its off status. However, no changes in the numbers of the nodes will be made, and the same packet will be sent out along this line again at the next cycle of the program.

Note that a node may be at capacity when a packet first starts out on a line toward it, and still have room for the packet by the time it arrives.

If a line is disconnected, the packet currently being sent on the line, if any, is unaffected. However, when the packet reaches its destination, the line will be displayed in its off state, and no more messages will be sent out on that line.

A picture of the network display is included at the end of this appendix.

3.0 Use of the Program

The program is called by the function ARPA() of no arguments. ARPA will go through several stages of initialization, during which the following will be typed:

```
0
1
INITIALIZING...
FINDING ROUTES...
STORING DISPLAY...
```

at which point the display appears on the scope. (There may be one or more garbage collections in the process.)

ARPA recognizes the following commands:

STOP	freezes the display at the end of the current cycle; i.e., no messages are sent or received. Any requests will be immediately executed.
OK	reverses the effect of STOP.
(node1 node2 N)	sends a message consisting of N packets from <u>node1</u> to <u>node2</u>, i.e., starts a message.
(node1 node2 N T)	sends a message consisting of N packets from <u>node1</u> to <u>node2</u> with the N packets being placed at the front queue in <u>node1</u>.
(node1 node2)	sends a message consisting of NUMPACK packets from <u>node1</u> to <u>node2</u>. NUMPACK is currently set to 8, but can be changed.
(CHANGE SPEED TO N)	changes speed factor to N. The number of cycles of the program required for a message packet to travel a line of length L (in scope units) is L/N. Speed is currently set to 400.
(CHANGE CAPACITY OF Node TO N)	changes capacity of Node to N. Capacities are initially set to value of variable CAPACITY, which is currently set to 50.
(DISCONNECT node1 TO node2)	disconnects line from <u>node1</u> to <u>node2</u>.

(RECONNECT node1 AND node2)	reconnects line from node1 to node2.
(REROUTE n AT node1 VIA node2)	all packets of message number n at node1 are placed in the queue waiting to go to node2.
(REROUTE n AT node1 VIA node2 T)	same as above except packets are placed at front of queue.
? node	prints a complete description of the status of node, in terms of which packets are in queues and which are holding.
? FROM node	prints location of all packets originated with node.
?TO node	prints location of all packets with node as their destination.
? TO1 node	prints all packets that are one node away from node.

ARPA responds to requests only once each cycle, which may be anywhere from 5 to 15 seconds, real time, depending on load of system state of network; i.e., more packets means more work for the program. The user should normally type STOP and wait for ARPA to respond with a : before typing any requests.

No errors made while typing requests are harmful. If the user makes an impossible request, such as disconnecting a non-existent line or querying a non-existent queue, the program will simply print a ? and await more input.

Whenever a message is started, its number is printed followed by its origin, destination and number of packets. Whenever a message is received, its number is printed, followed by its destination, origin, and the length of time, in number of cycles, required for transmission. Whenever an acknowledgment of a message is received back at its origin, the message number and the message RECEIPT ACKNOWLEDGED is typed.

4.0 Changing the Network

The current network may be found on the variable ARPA. The form of the network is a list of lists each of the form (name X Y name1... namen); e.g., (BBN 450 450 CMU HARV DART). name is the name of the node; X and Y, its scope coordinates. The scope is 1024 points by 1024 points with (0, 0) at the center. Thus BBN would be at the upper right hand corner. Following the coordinates are a list of the names of the nodes to which BBN is directly connected. Redundancy is allowed but not necessary; i.e., if BBN is specified as connected to HARV, it is not necessary to also specify that HARV is connected to BBN; this is assumed.

The program which initializes the network is called SETUP. The function ARPA calls SETUP with ARPA as its single argument the very first time ARPA is called. If the user wishes to change the network, he should leave the function ARPA by typing control-C, stop the display by typing HPSTOP(), and then perform E (SETUP network), where network is his new network. When this is completed, he can then call ARPA again by typing ARPA(). ARPA will type STORING DISPLAY... and will regenerate the display. Once this is done, the user can proceed as before with his new network.

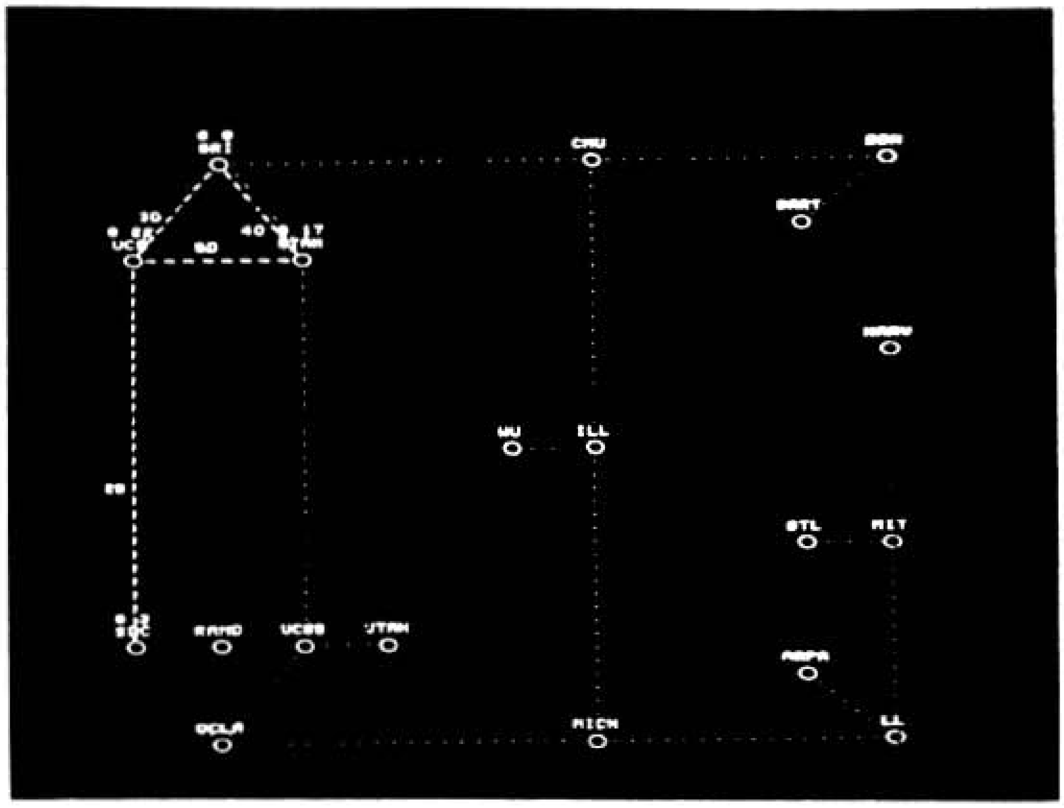


PLATE 2. PHOTOGRAPH OF THE DISPLAY.

1) BACK-UP AND SUPPORT PERSONNEL

2) WEST COAST PERSONNEL

APPENDIX J: RESUMES OF BBN PERSONNEL

Frank E. Heart
Hawley K. Rising
Severo M. Ornstein
William Crowther
David C. Walden
Robert Kahn
~~Robert V. Jacobson~~
~~John C. Henry~~
~~Ronald F. Gagne~~
~~Alexander A. McKenzie~~
Bernard Cosell
~~Frederick N. Webb~~
~~Michael S. Fein~~

Back-Up and Support Personnel

Jerome I. Elkind
Daniel G. Bobrow
Theodore R. Strollo
John Barnaby
Ralph Alter
Joseph Markowitz
Sheldon Boilen
Daniel L. Murphy

West Coast Personnel

~~Roland F. Bryan~~
~~Ralph P. Graeber~~
~~Ralph E. Athearn~~
~~Ronald L. Freeman~~