BOLT BERANEK AND NEWMAN INC

CONSULTING · DEVELOPMENT · RESEARCH

TECHNICAL INFORMATION REPORT NO. 89

THE INTERFACE MESSAGE PROCESSOR PROGRAM

Developed by Bolt Beranek and Newman Inc.
under Contracts No. DAHC15-69-C-0179 and F08606-73-C-0027,
sponsored by the Advanced Research Projects Agency

CAMBRIDGE NEW YORK CHICAGO LOS ANGELES SAN FRANCISCO

BOLT BERANEK AND NEWMAN INC.

THE INTERFACE MESSAGE PROCESSOR PROGRAM

February 1973

Update Edition of November 1973

BOLT BERANEK AND NEWMAN INC.

THE INTERFACE MESSAGE PROCESSOR PROGRAM

February 1973

Update Edition of November 1973

TABLE OF CONTENTS

Figures

1. Introduction

The ARPA Network has been in operation for about four years and has become a national facility. The network has grown to over thirty sites spread across the United States, including the recent connection via satellite to Hawaii, and is steadily growing; over forty independent computer systems of varying manufacture are interconnected. Provision has been made for terminal access to the network from sites which do not enjoy the ownership of an independent computer system. A map of the ARPA Network as of September 1973 is shown in Figure 1-1.

Implementation of the IMPs required the development of a sophisticated computer program. This program has been previously described in [1,5]. As stated then, the principal function of the IMP program is the processing of packets, including the following: segmentation of Host messages into packets; receiving, routing, and transmitting store-and-forward packets; retransmitting unacknowledged packets; reassembling packets into messages for transmission into a Host; and generating RFNMs and other control messages. The program also monitors network status, gathers statistics, and performs on-line testing.

ARPA NETWORK, GEOGRAPHIC MAP
SEPTEMBER 1973

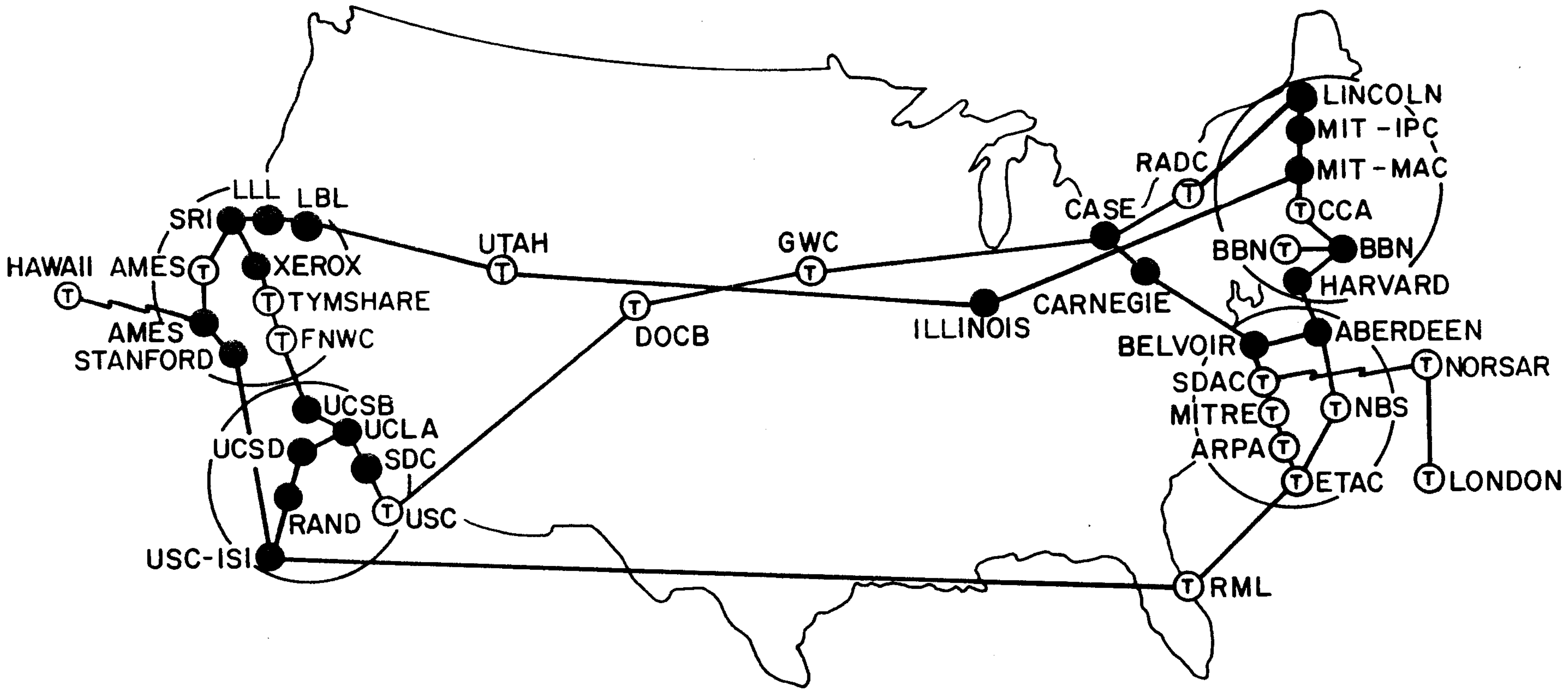
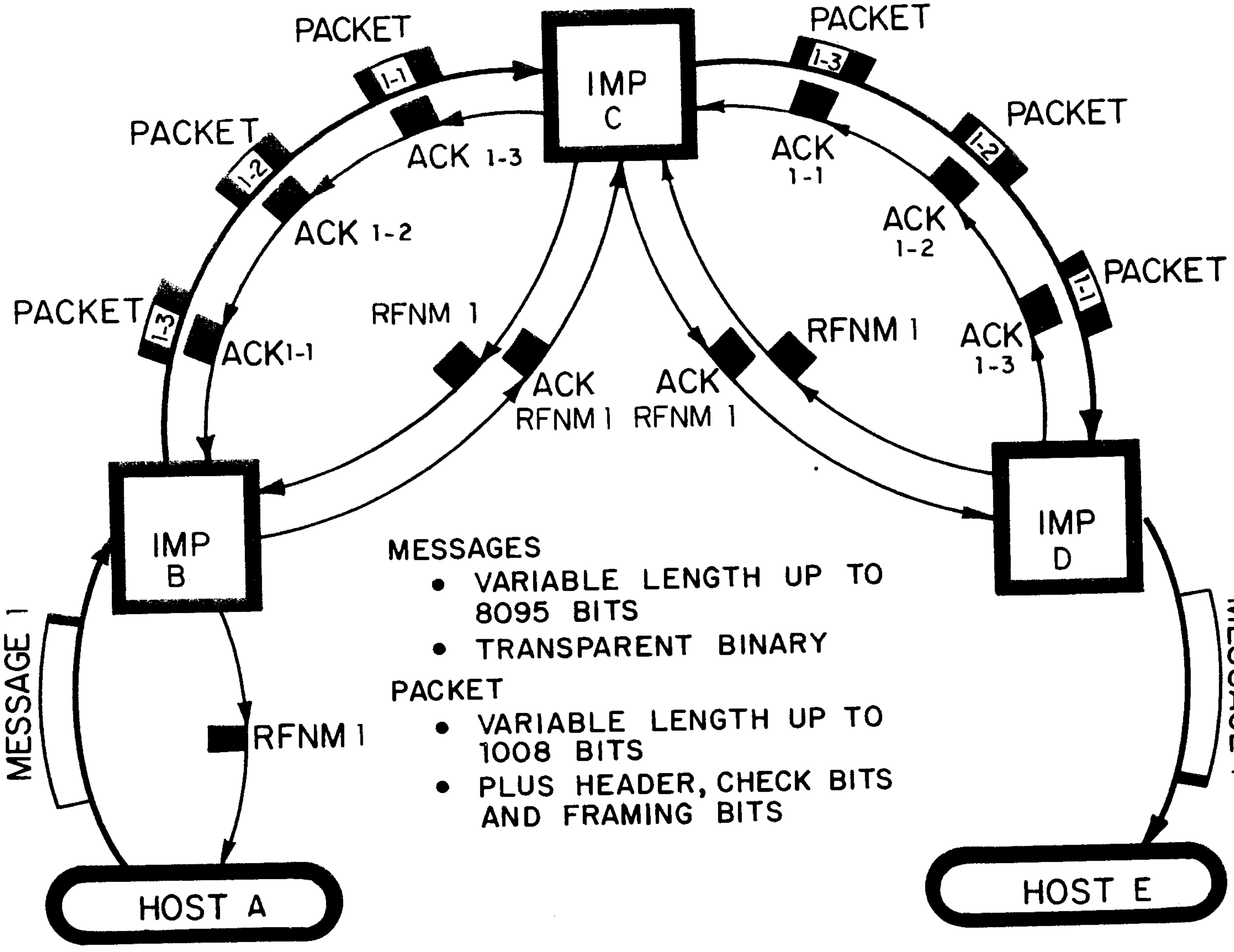


Figure 1-1

2. IMP Processes

This section considers the algorithms that the IMP uses in performing its functions as a message-switching center and interface between Host computers. Figure 2-1 helps summarize some of the terms we will be using. The Host sends the IMP a _message_ up to 8095 bits long. The message has a _leader_ specifying its destination. The source IMP accepts the message in _packets_ up to 1008 bits long. Each packet has a _header_ to allow for the transmission from IMP to IMP. Figure 2-1 demonstrates how message 1 is transferred from IMP to IMP in these packets, numbered 1-1, 1-2, and 1-3. When a packet is successfully received at each IMP, an _acknowledge_ or _ack_ is sent back to the previous IMP. Inter-IMP acks are shown returning for each packet. Finally the message arrives at the destination IMP where it is _reassembled_: that is, the packets are recombined into the original message. The message is sent to the destination Host and when it has been accepted, a _Ready For Next Message_ (RFNM) is sent back to the source Host. A RFNM is a unique, one-packet message and it is acknowledged. Several points are worth noting. First, acks are not actually separate transmissions, but are piggy-backed in returning packets to cut down on overhead. Next, packets on the inter-IMP lines are checksummed in the modem interface hardware and the IMP employs a positive acknowledgment retransmission scheme. That is, if a packet is in error, it is not acknowledged. Then it is retransmitted until an acknowledge is received. Further, because of dynamic routing, an IMP may send the several packets of a



MESSAGES
- VARIABLE LENGTH UP TO 8095 BITS
- TRANSPARENT BINARY

PACKET
- VARIABLE LENGTH UP TO 1008 BITS
- PLUS HEADER, CHECK BITS AND FRAMING BITS

Figure 2-1. Message Protocol

message out on different lines. For both of these reasons, the packets of a message may arrive at the destination IMP out of order and must be reassembled into the correct order for transmission to the destination Host.

2. IMP-Host Protocols

2.1.1 Messages and RFNMs

A major hazard in a message-switched network is congestion, which can arise either from system failures or from peak traffic flow. Congestion typically occurs when a destination IMP becomes flooded with incoming messages for its Host. If the flow of messages to this destination is not regulated, the congestion will back up into the network, affecting other IMPs and degrading or even completely clogging the communication service. To solve this problem a quenching scheme was developed that limits the flow of messages to a given destination before congestion begins to occur.

This quenching scheme consists of practices which allocate buffer space before a message may enter the system. If buffering is provided in the source IMP, one can optimize for low delay transmissions. If the buffering is provided at the destination IMP, one can optimize for high bandwidth transmissions. To be consistent with the goal of a balanced communications system, an approach has been developed which utilizes some buffer storage at both the source and the destination; the solution also utilizes a request mechanism from source IMP to destination IMP.

Specifically, no multi-packet message is allowed to enter the network until storage for the message has been allocated at the destination IMP. As soon as the source IMP takes in the first packet of a multi-packet message, it sends a small control message to the destination IMP requesting that reassembly storage be reserved at the destination for this message. It does not take in further packets from the Host until it receives an allocation message in reply. The destination IMP queues the request and sends the allocation message to the source IMP when enough reassembly storage is free; at this point the source IMP sends the message to the destination.

Effective bandwidth is maximized for sequences of long messages by permitting all but the first message to bypass the request mechanism. When the message itself arrives at the destination, and the destination IMP is about to return the RFNM, the destination IMP waits until it has room for an additional multi-packet message. It then piggybacks a storage allocation on the RFNM. If the source Host is prompt in answering the RFNM with its next message, an allocation is ready and the message can be transmitted at once. If the source Host delays too long, or if the data transfer is complete, the source IMP returns the unused allocation to the destination. With this mechanism, the inter-message delay has been minimized and the Hosts can obtain the full bandwidth of the network.

The delay for a short message has been minimized by transmitting it to the destination immediately while keeping a copy in the source IMP. If there is space at the destination, it is accepted and passed on to a Host and a RFNM is returned; the source IMP discards the message when it receives the RFNM. If not, the message is discarded, a request for allocation is queued and, when space becomes available, the source IMP is notified that the message may now be retransmitted. Thus, no setup delay is incurred when storage is available at the destination.

These mechanisms make the IMP network fairly insensitive to unresponsive Hosts, since the source Host is effectively held to a transmission rate equal to the reception rate of the destination Host. Further, reassembly lockup is prevented because the destination IMP will never have to turn away a multi-packet message destined for one of its Hosts; reassembly storage has been allocated for each such message in the network.

2.1.2 Host-IMP Interfacing

Each IMP will service up to four Hosts whose cable distances from the IMP are less than 2000 feet. For distances greater than that, a modem channel must be used. This latter type of Host connection is termed a Very Distant Host (VDH). Procedures used for VDH connections are discussed in reference [4] and section 2.6 of this report.

Connecting an IMP to a wide variety of different local Hosts, however, requires a hardware interface, some part of which must be custom tailored to each Host. It was decided, therefore, to partition the interface such that a standard portion would be built into the IMP, and would be identical for all Hosts, while a special portion of the interface would be unique to each Host. The interface is designed to allow messages to flow in both directions at once. A bit-serial interface was designed partly because it required fewer lines for electrical interfacing and was, therefore, less expensive, and partly to accommodate conveniently the variety of word lengths in the different Host computers. The bit rate requirement on the Host line is sufficiently low that parellel transfers are not necessary.

The Host interface operates asynchronously, each data bit being passed across the interface via a Ready for Next Bit/There's Your Bit handshake procedure. This technique permits the bit rate to adjust to the rate of the slower member of the pair and allows necessary interruptions, when words must be stored into or retrieved from memory. The IMP introduces a preadjusted delay between bits that limits the maximum data rate; at present, this delay is set to 10 μsec. Any delay introduced by the Host in the handshake procedure further slows the rate below this 100 Kbs maximum.

2.2 IMP-IMP Message Protocols

To insure that messages arrive at a destination Host in proper order, a sequence control mechanism was developed based on a single logical "pipe" between each source and destination IMP. Each IMP maintains an independent message number sequence for each pipe. A message number is assigned to each message at the source IMP and this message number is checked at the destination IMP. All Hosts at the source and destination IMPs share this message space. Out of an eight-bit message number space both the source and destination keep a small window of currently valid message numbers, which allows several messages to be in the pipe simultaneously. Messages arriving at a destination IMP with out-of-range message numbers are duplicates to be discarded. The window presently encompasses four numbers. The message number concept serves two purposes: it orders the four messages that can be in the "logical" pipe, and it allows detection of duplicates. The message number is internal to the IMP subnetwork and is invisible to the Hosts.

A sequence control system based on a single source/destination pipe, however, does not permit priority traffic to go ahead of other traffic. This problem was solved by permitting two pipes between each source and destination, a priority (or low delay) pipe and a non-priority (or high bandwidth) pipe. To avoid having each IMP maintain two eight-bit message number sequences for every other IMP in the network, the low delay and high bandwidth pipes were coupled so that duplicate detection can be performed in common, thus requiring only one eleven-bit message number sequence for each IMP.

The eleven-bit number consists of a one-bit priority/ non-priority flag, two bits to order priority messages, and eight bits to order all messages. For example, if we use the letters A, B, C, and D to denote the two-bit order numbers for priority messages and the absence of a letter to indicate a non-priority message, we can describe a typical situation as follows: the source IMP sends out non-priority message 100, then priority messages 101A and 102B, and then non-priority message 103. Suppose the destination IMP receives these messages in the order 102B, 101A, 103, 100. It passes these messages to the Host in the order 101A, 102B, 100, 103. Message number 100 could have been sent to the destination Host first if it had arrived at the destination first, but the priority messages are allowed to "leapfrog" ahead of message number 100 since it was delayed in the network. The IMP holds 102B until 101A arrives, as the Host must receive priority message A before it receives priority message B. Likewise, message 100 must be passed to the Host before message 103.

Hosts may, if they choose, have several messages outstanding simultaneously to a given destination but, since priority messages can "leapfrog" ahead, and the last message in a sequence of long messages may be short, priority can not be assigned strictly on the basis of message length. Therefore, Hosts must explicitly indicate to the IMP whether a message has priority or not.

Since message numbers and reserved storage are so critical in the system, very stringent and careful procedures were developed to account for a lost message. The source IMP keeps track of all messages for which a RFNM has not yet been received. When the RFNM is not received for too long (presently about 30 seconds), the source IMP sends a control message to the destination inquiring about the possibility of an incomplete transmission. The destination responds to this message by indicating whether the message in question was previously received or not. The source IMP continues inquiring until it receives a response. This technique guarantees that the source and destination IMPs keep their message number sequences synchronized and that any allocated space will be released in the rare case that a message is lost in the subnetwork because of a machine failure.

2.3 IMP-to-IMP Channel Protocol

2.3.1 Logical Channel Protocol

A technique has been adopted for IMP-to-IMP transmission control which improves efficiency by 10-20% over the original separate acknowledge/timeout/retransmission approach described in [1]. In the new scheme, which is also used for the Very Distant Host [4], each physical inter-IMP circuit is broken into a number of logical channels, currently eight in each direction. Acknowledgments are returned piggybacked on normal network traffic in a set of eight acknowledgment bits, one bit per channel, contained in every packet, thus requiring less bandwidth than the original method of sending each acknowledge in its own packet. In addition, the period between retransmissions is dependent upon the volume of new traffic. Under light loads the network has minimal retransmission delays, and the network automatically adjusts to minimize the interference of retransmissions with new traffic.

Each packet is assigned to an outgoing logical channel and carries the odd/even bit for its channel (which is used to detect duplicate packet transmissions), its channel number, and eight acknowledge bits - one for each channel in the reverse direction.

The transmitting IMP continually cycles through its used channels (those with packets associated with them), transmitting the packets along with the channel number and the associated odd/even bit. At the receiving IMP, if the odd/even bit of the received packet does not match the odd/even bit associated with the appropriate receive channel, the packet is accepted and the receive odd/even bit is complemented; otherwise the packet is a duplicate and is discarded.

Every packet arriving over a line contains acknowledges for all eight channels. The ack bits are set up at the distant IMP when it copies its receive odd/even bits into the positions reserved for the eight acknowledge bits in the control portion of every packet transmitted. In the absence of other traffic, the acknowledges are returned in null packets in which only the acknowledge bits contain relevant information (i.e., the channel number and odd/even bit are meaningless; null packets are not acknowledged). When an IMP receives a packet, it compares (bit by bit) the acknowledge bits against the transmit odd/even bits. For each match found, the corresponding channel is marked unused, the corresponding waiting packet buffer is discarded, and the transmit odd/even bit is complemented.

In view of the large number of channels, and the delay that is encountered on long lines, some packets may have to wait an inordinately long time for transmission. A one-character packet should not have to wait for several thousand-bit packets to be transmitted, multiplying by 10 or more the effective delay seen by the source. Therefore, the following transmission ordering scheme has been instituted: priority packets which have never been transmitted are sent first; next sent are any regular packets which have never been transmitted; finally, if there are no new packets to send, previously transmitted packets which are unacknowledged are sent. Of course, unacknowledged packets are periodically retransmitted even when there is a continuous stream of new traffic.

2.3.2 Physical Circuit Protocol

Each packet is individually routed from IMP to IMP through the network toward the destination. At each IMP along the way, the transmitting hardware generates initial and terminal framing characters and checksum digits that are shipped with the packet and are used for error detection by the receiving hardware of the next IMP. The format of a packet on an inter-IMP channel is shown in Figure 2-2.

Errors in transmission can affect a packet by destroying the framing and/or by modifying the data content. If the framing is disturbed in any way, the packet either will not be recognized or will be rejected by the receiver. In addition, the check digits provide protection against errors that affect only the data. The check digits can detect all patterns of four or fewer errors occurring within a packet, and any single error burst of a length less than twenty-four bits. An overwhelming majority of all other possible errors (all but about one in 2^{24}) is also detected. Thus, the mean time between undetected errors in the subnet should be on the order of years.

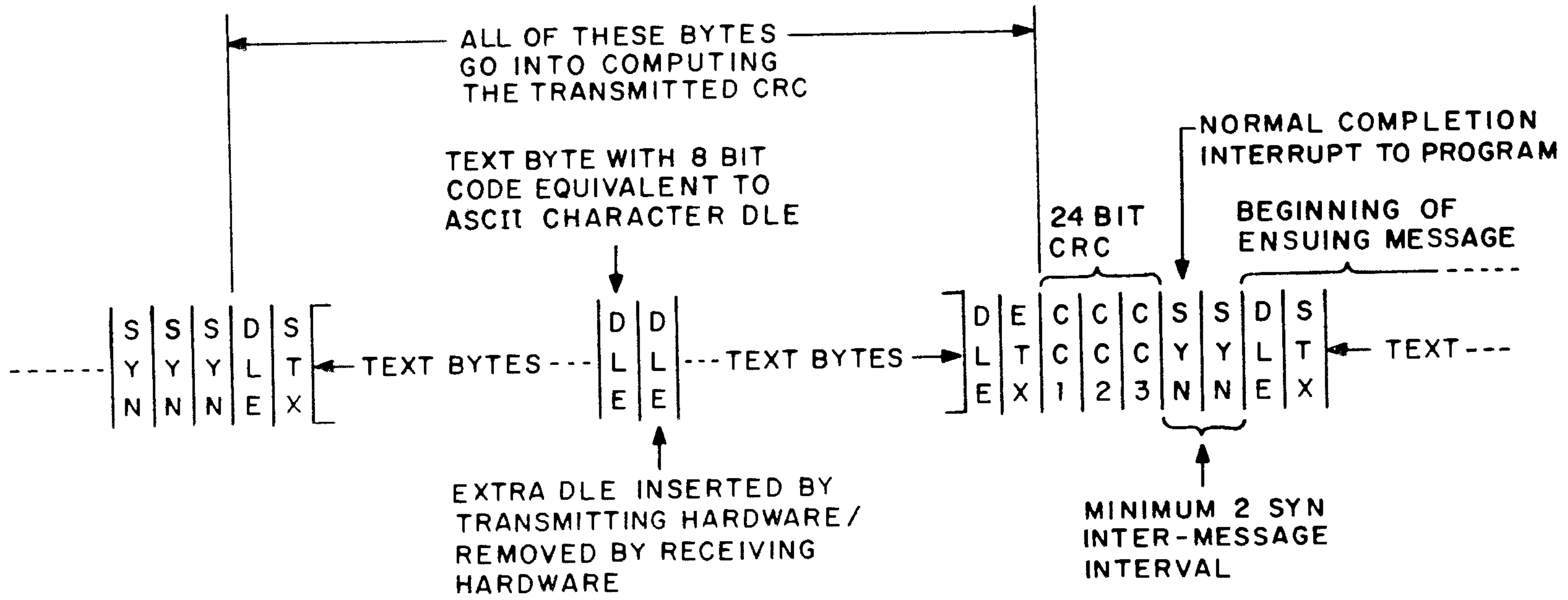

ALL OF THESE BYTES GO INTO COMPUTING THE TRANSMITTED CRC
TEXT BYTE WITH 8 BIT CODE EQUIVALENT TO ASCII CHARACTER DLE
NORMAL COMPLETION INTERRUPT TO PROGRAM
24 BIT CRC
BEGINNING OF ENSUING MESSAGE
SYN SYN SYN DLE STX
TEXT BYTES
DLE DLE
TEXT BYTES
DLE ETX CC1 CC2 CC3 SYN SYN DLE STX
TEXT
EXTRA DLE INSERTED BY TRANSMITTING HARDWARE/ REMOVED BY RECEIVING HARDWARE
MINIMUM 2 SYN INTER-MESSAGE INTERVAL

Figure 2-2. Packet Format on Line

2.4 Routing Algorithm

The routing algorithm directs each packet to its destination along a path for which the total estimated transit time is smallest. This path is not determined in advance. Instead, each IMP individually decides onto which of its output lines to transmit a packet addressed to another destination. This selection is made by a fast and simple table lookup procedure. For each possible destination, an entry in the table designates the appropriate next leg. These entries reflect line or IMP trouble, traffic congestion, and current local subnet connectivity. This routing table is updated about every two-thirds of a second, as described below.

Each IMP estimates the delay it expects a packet to encounter in reaching every possible destination over each of its output lines. It selects the minimum delay estimate for each destination and periodically passes these estimates to its immediate neighbors. Each IMP then constructs its own routing table by combining its neighbors' estimates with its own estimates of the delay to each neighbor. The estimated delay to each neighbor is based upon both queue lengths and the recent performance of the connecting communication circuit. For each destination, the table is then made to specify that selected output line for which the sum of the estimated delay to the neighbor plus the neighbor's delay to the destination is smallest.

The routing table is periodically and dynamically updated to adjust for changing conditions in the network. The system is adaptive to the ups and downs of lines, IMPs, and congestion; it does not require the IMP to know the topology of the network.

In particular, an IMP need not even know the identity of its immediate neighbors. Thus, the leased circuits could be reconfigured to a new topology without requiring any changes to the IMPs.

The routing program has recently been modified to provide for more rapid and efficient propagation of routing messages. Through use of a technique called hold-down, the IMPs delay the route changeover process for a few seconds and in this way permit a faster and smoother cutover. When the best route is about to change, the IMPs first make sure that the neighboring IMPs know that the old route has gone bad before they attempt to change; this prevents the adjacent IMPs from slowing down the process by transmitting old information.

The IMPs also measure the bandwidth and loading of the circuits to which they are connected. The IMPs send routing proportionately more often on faster lines. In addition, they send routing proportionately more often on idle lines. Thus, the percentage of line bandwidth used for routing varies between 3% and 15%, approximately, as a function of line use.

Finally, the IMPs perform the routing computation on an incremental basis as each routing message is received. This means that the routing message being output on a given line is as up-to-date as possible. The routing messages carry serial numbers to permit the IMPs to detect that a new set of routing data has arrived which then is used, with the current data, to form the next routing message. There is a triple buffer pool for the routing message being output, the routing message being built, and the idle buffer, previously used for output. Each time a new input is received and no line is using the idle buffer for output, there is a cyclic permutation of the input, output, and free buffers.

Each of the routing messages sent and received carries a software checksum. In addition, the input, output, and timeout processes for routing all compute a checksum on the program before executing it, and a reload is initiated in the event of failure. These reliability measures are discussed in more detail in the next section.

2.5 Failure Protocols

The network is designed to be largely invulnerable to circuit or IMP failure as well as to outages for maintenance. Special status and test procedures are employed to help cope with various failures. In the normal course of events the IMP program transmits hello bits in its routing messages. The acknowledgment for a hello packet is an I-heard-you (IHY) bit in a returning null packet.

A dead line is detected by the sustained absence (approximately 3.2 sec) of IHY messages on that line. No regular packets will be routed onto a dead line, and any packets awaiting transmission will be rerouted. Routing tables in the network are adjusted automatically to reflect the loss. Receipt of thirty consecutive I-heard-you packets is required (an event which consumes at least 15 seconds) before a dead line is defined to be alive once again.

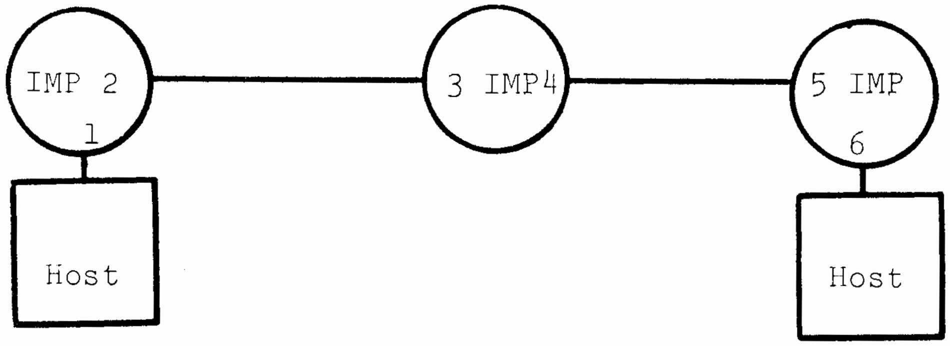
A dead line may reflect trouble either in the communication facilities or in the neighboring IMP itself. Normal line errors caused by dropouts, impulse noise, or other similar conditions should not result in a dead line, because such errors typically last only a few milliseconds, and only occasionally as long as a few tenths of a second. Therefore, it is expected that a line will be defined as dead only when serious trouble conditions occur.

If dead lines eliminate all routes between two IMPs, the IMPs are said to be disconnected and each of these IMPs will discard messages destined for the other. Disconnected IMPs cannot be rapidly detected from the delay estimates that arrive from neighboring IMPs. Consequently, additional information is transmitted between neighboring IMPs to help detect this condition. Each IMP transmits to its neighbors the length of the shortest existing path (i.e., number of IMPs) from itself to each destination. To the smallest such received number per destination, the IMP adds one. This incremented number is the length of the shortest path from that IMP to the destination. If the length ever exceeds the number of network nodes, the destination IMP is assumed to be unreachable and therefore disconnected.

Messages intended for dead Hosts (which are not the same as dead IMPs) cannot be delivered; therefore, these messages require special handling to avoid indefinite circulation in the network and spurious arrival at a later time. Such messages are purged from the network at the destination IMP. A Host computer is notified about another dead Host only when attempting to send a message to that Host.

The components of the IMP program dedicated to improving reliability have two main functions. First, the software is built to be as invulnerable as is possible in practice to hardware failures. Second, the software isolates and reports what failures it can detect to the NCC. With intermittent failures, it is important in practice to keep the IMP program running and diagnosing the problem rather than keeping the IMP down for long periods to run special hardware diagnostics.

The IMPs use the technique of software checksums on all transmissions to detect errors in packets, protecting the integrity of the data and isolating hardware failures. The end-to-end software checksum on packets, without any time gaps, works as follows:



- A checksum is computed at the source IMP for each packet as it is received from the source Host (interface 1).

- The checksum is verified at each intermediate IMP as it is received over the circuit from the previous IMP (interfaces 3 and 5).

- If the checksum is in error, the packet is discarded, and the previous IMP retransmits the packet when it does not receive an acknowledgement (interfaces 2 and 4).

- The previous IMP does not verify the checksum before the original transmission, to cut the number of checks in half. But when it must retransmit a packet it does verify the checksum. If it finds an error, it has detected an intra-IMP failure, and the packet is lost. If not, then the first transmission was lost due to an inter-IMP failure, a circuit error, or was simply refused by the adjacent IMP. The previous IMP holds a good copy of the packet, which it then retransmits (interfaces 2 and 4).

- After the packet has successfully traversed several intermediate IMPs, it arrives at the destination IMP. The checksum is verified just before the packet is sent to the Host (interface 6).

This technique provides a checksum from the source IMP to the destination IMP on each packet, with no gaps in time when the packet is unchecked. Any errors are reported to the NCC in full, with a copy of the packet in question. This method answers both requirements stated above: it makes the IMPs more reliable and fault-tolerant, and it provides a maximum of diagnostic information for use in fault isolation.

One of the major questions about such approaches is their efficiency. We have been able to include the software checksum on all packets without greatly increasing the processing overhead in the IMP. The method described above involves one checksum calculation at each IMP through which a packet travels. We developed a very fast checksum technique, which takes only 2 μsec per word. The program computes the number of words in a packet and then jumps to the appropriate entry in a chain of add instructions. This produces a simple sum of the words in the packet, to which the number of words in the packet is added to detect missing or extra words of zero. With the inclusion of this code, the effective processor bandwidth of a 516 IMP is reduced by one-eighth for full-length store-and-forward packets. This add checksum is not a very good one in terms of its error-detecting capabilities, but it is as much as the IMP can afford to do in software. Furthermore, the primary goal of this modification is to assist in the remote diagnosis of intermittent hardware failures.

A different set of reliability measures has been instituted for routing. It is clear that catastrophic effects can follow for the Network as a whole when a single IMP begins to propagate incorrect routing information. This failure may be due to a memory failure in the data area or in the program itself. A single broken instruction in the part of the IMP program that builds the routing message causes the routing messages from the IMP to be random data. The neighboring IMPs interpret these messages as routing update information, and traffic flow through the Network can be completely disrupted.

This kind of problem, the introduction of incorrect routing information into the Network, can happen in three ways:

- The routing message is changed in transmission. The inter-IMP checksum should catch this. The bad routing messages we saw in the Network had good checksums.
- The routing message is changed as it is constructed, say by a memory or processor failure, or before it is transmitted. This is what we termed above an intra-IMP failure.
- The routing program is incorrect for hardware or software reasons.

The last two kinds of problems can be fixed by extending the concept of software checksums. The routing program has been modified to build a software checksum for the routing message as it builds the message, just as if it came from a Host. It is important that this checksum refer to the intended contents of the routing message, not the actual contents. That is, the program which generates the routing message builds its own software checksum as it proceeds, not by reading what has been stored in the routing message area, but by adding up the intended contents for each entry as it computes them. The process which sends out routing messages then always verifies the checksum before transmitting them. This scheme should detect all intra-IMP failures.

Finally, the routing program itself is checksummed to detect any changes in the code. The programs which copy in received routing messages, compute new routing tables, and send out routing messages each calculate the checksum of the code before executing it. If the program finds a discrepancy in the checksum of the program it is about to run, it immediately requests a program reload from an adjacent IMP. These checksums include the checksum computation itself, the routing program and any constants referenced. This modification should prevent a hardware failure at one IMP from affecting the Network at large by stopping the IMP before it does any damage in terms of spreading bad routing.

2.6 Very Distant Host (VDH) Protocols

In instances where a Host is located more than 2000 feet from the IMP, connection is made by means of the standard modem interface hardware normally used for inter-IMP communication. Reference [4] contains a detailed description of the protocol used for this type of interface.

Briefly, the method used to assure successful IMP-Host transfers is similar to that used for the inter-IMP channels. Logical channels are used as described in section 2.3.1, although in this case only two channels are employed and the order of transmission is important. Therefore, both the Host and IMP software must be aware of packets. For example, assume packet A is transmitted from an IMP on channel Ø, and packet B is then transmitted on channel 1. If an error were detected in packet A, but not B, no ack would be returned for A. The Host would retain Packet B until A is retransmitted to it and received successfully, thus insuring delivery of the packets to its own processes in order A-B.

3. PROGRAM DESCRIPTIONS

Implementation of the IMPs required the development of a sophisticated operational computer program and the development of several auxiliary programs for hardware tests, program construction, and debugging. This section discusses the design of the operational program and describes the auxiliary software. Detailed program descriptions for the IMP software are included in Section 4.

3.1 General Descriptions

As previously mentioned, the principal function of the IMP operational program is the processing of packets. This processing includes segmentation of Host messages into packets for routing and transmission, building of headers, receiving, routing and transmitting of unacknowledged packets, reassembling of received packets into messages for transmission to the Host, and generating of RFNMs and acknowledgments. The program also monitors network status, gathers statistics, and performs on-line testing.

The entire program is composed of fifteen functionally distinct routines; each piece occupies no more than two or three pages of core (512 words per page). These routines communicate primarily through common registers residing in page zero of the machine which are directly addressable from all pages of memory.

A map of core storage is shown in Figure 3-1. Seven of the fifteen programs are directly involved in the flow of packets through the IMP: the task program performs the major portion of the packet processing, including the reassembly of Host messages; the modem programs (IMP-to-Modem and Modem-to-IMP) handle interrupts and the resetting of buffers for the Modem channels; the Host programs (IMP-to-Host and Host-to-IMP) handle interrupts and resetting of buffers for the Host channels, build packet headers during input, and construct allocation requests sent to the destination IMPs; the timeout program maintains a software clock, times out unused buffer allocations, reinitiates programs which have paused, and initiates routing computations and other relatively infrequent events. A background loop contains the remaining major programs and deals with initialization, debugging, testing, statistics gathering, and tracing. Background programs also initiate RFNM allocation and other sequencing and control messages. After a brief description of data structures, we will discuss packet processing in some detail.

3.1.1 Data Structures

The major system data structures consist of buffers, queues and tables.

Buffer Storage. The buffer storage space consists of about 80 fixed length buffers, each of which is used for storing a single packet. An unused buffer is chained onto a free buffer list and is removed from this list when it is needed to store an incoming packet. A packet, once stored in a buffer, is never moved. After a packet has been successfully passed along to its Host or to another IMP, its buffer is returned to the free list. The buffer space is partitioned in such a way that each process

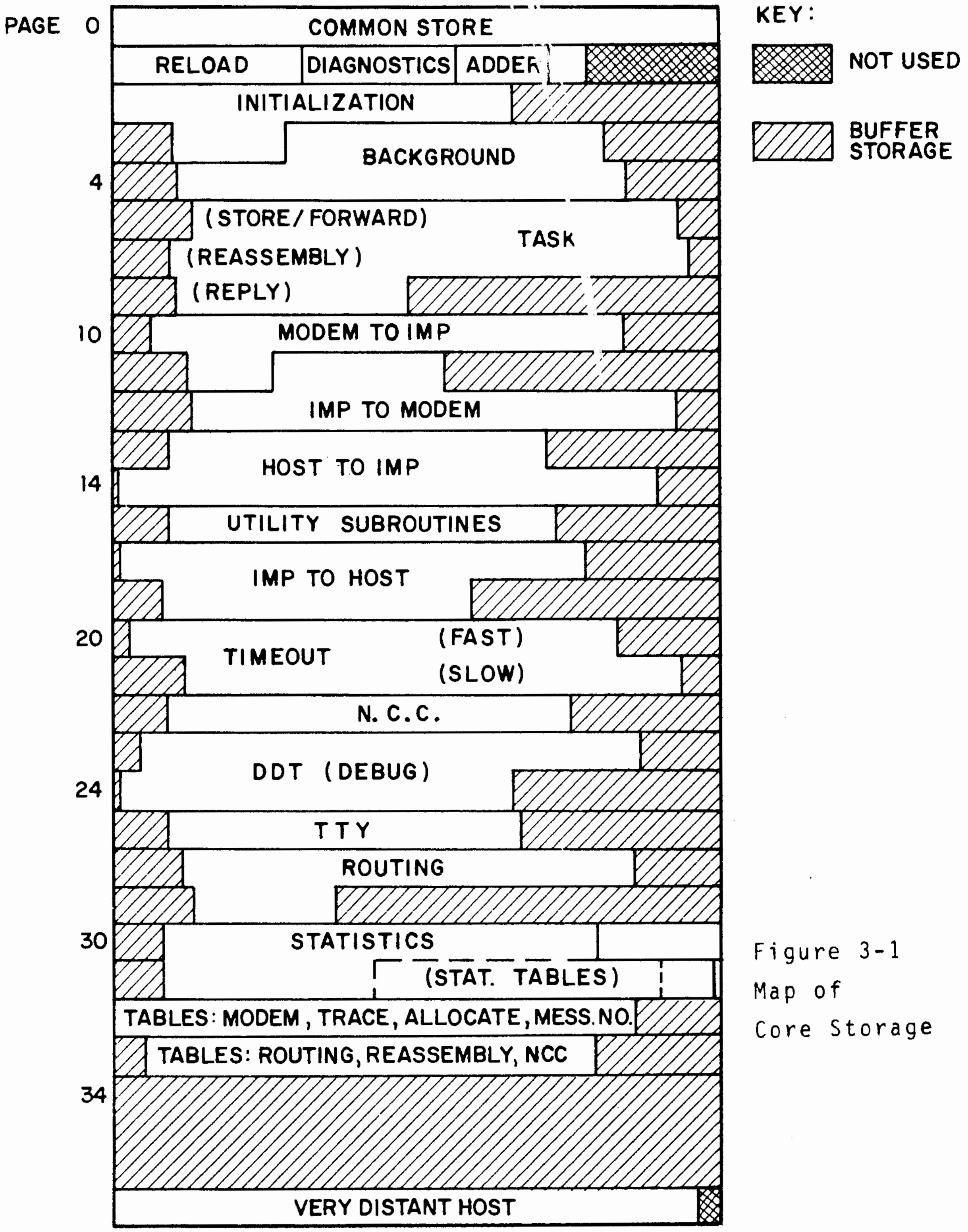


Figure 3-1
Map of
Core Storage

IMP 3050

(store and forward traffic, Host traffic, etc.) is always guaranteed some buffers. For the sake of program speed and simplicity, no attempt is made to retrieve the space wasted by partially filled buffers.

In handling store and forward traffic, all processing is on a per-packet basis. Further, although traffic to and from Hosts is composed of messages, the IMP converts to dealing with packets; the Host transmits a message as a single unit but the IMP takes it one buffer at a time. As each buffer is filled, the program selects another buffer for input until the entire message has been provided for. These successive buffers will, in general, be scattered throughout the IMP's memory. An equivalent inverse process occurs on output to the Host after all packets of the message have arrived at the destination IMP. No attempt is made to collect the packets of a message into a contiguous portion of IMP memory. The allocation of buffer space in core storage is shown in Figure 4, as mentioned previously. Notice that code is generally centered within a page, and there is code on every page of core.

The IMP program uses the following set of rules to allocate the approximately 80 available buffers to the various tasks requiring them:

- Each line must be able to get its share of buffers for input and output. In particular, one buffer is always allocated for output on each line, guaranteeing that output is always possible for each line. Double buffering is provided for input on each line, which permits all input traffic to be examined by the program. Thus, acknowledgments can always be processed, which frees buffers.

- An attempt is made to provide enough store-and-forward buffers so that all lines may operate at full capacity. The number of buffers needed depends directly on line distance and line speed. The current limit for each line is eight or fewer buffers, and a pool is provided for all lines. A maximum of twenty buffers is available in the store-and-forward pool.
- Ten buffers are always allocated to reassembly storage, allowing allocations for one multi-packet message and two single-packet messages. Additional buffers may be claimed for reassembly, up to a maximum of seventy-four.

Buffers currently in use are either dedicated to an incoming or outgoing packet, chained on a queue awaiting processing by the program, or being processed. Occasionally, a buffer may be simultaneously found on two queues; this situation can occur when a packet is waiting on one queue to be forwarded and on another to be acknowledged.

Queues. There are three principal types of queues:

- Task: All routing packets, all packets from the modems and all packets received on Host channels are placed on the task queue.
- Output: A separate output queue is constructed for each inter-IMP modem circuit and each Host. Each modem output queue is subdivided into a priority queue and a regular message queue, which are serviced in that order. Each Host output queue is subdivided into a control message queue, a priority queue, and a regular message queue, which are also serviced in the indicated order.
- Reassembly: The reassembly queue contains those packets being reassembled into messages for the Host.

Tables. Tables in core are used for the storage of queue pointers, for trace blocks, for reassembly information, for messages waiting for allocation, for routing, allocation, sequence control, and for statistics and trace data. Figure 5 summarizes the IMP table storage. All IMPs have identical tables. The program has twelve words of tables for each of the sixty-four IMPs now possible in the network. The program has ninety-one words of tables for each of the eight Hosts (four real and four fake) that can be connected; additionally, twelve words of code are replicated for each real Host that can be connected. The program has fifty-five words of tables for each of the five lines that can be connected; additionally, thirty-seven words of code are replicated for each line that can be connected.

The size of the initialization code and the associated tables deserves mention. This was originally quite small. However, as the network has grown and the IMP's capabilities have been expanded, the amount of memory dedicated to initialization has steadily grown. This is mainly due to the fact that the IMPs are no longer identically configured. An IMP may be required to handle a Very Distant Host, or TIP hardware, or five lines and two Hosts, or four Hosts and three lines, or a very high speed line, or a satellite link. As the physical permutations of the IMP have continued to increase, the criterion followed has been that the program should be identical in all IMPs, allowing an IMP to reload its program from a neighboring IMP and providing other considerable advantages. However, maintaining only one version of the program means that the program must rebuild itself during initialization to be the proper program to handle the particular physical configuration of the IMP. Furthermore, it must be able to turn itself back into its nominal form when it is reloaded into a neighbor. All of this takes tables and code. Unfortunately,

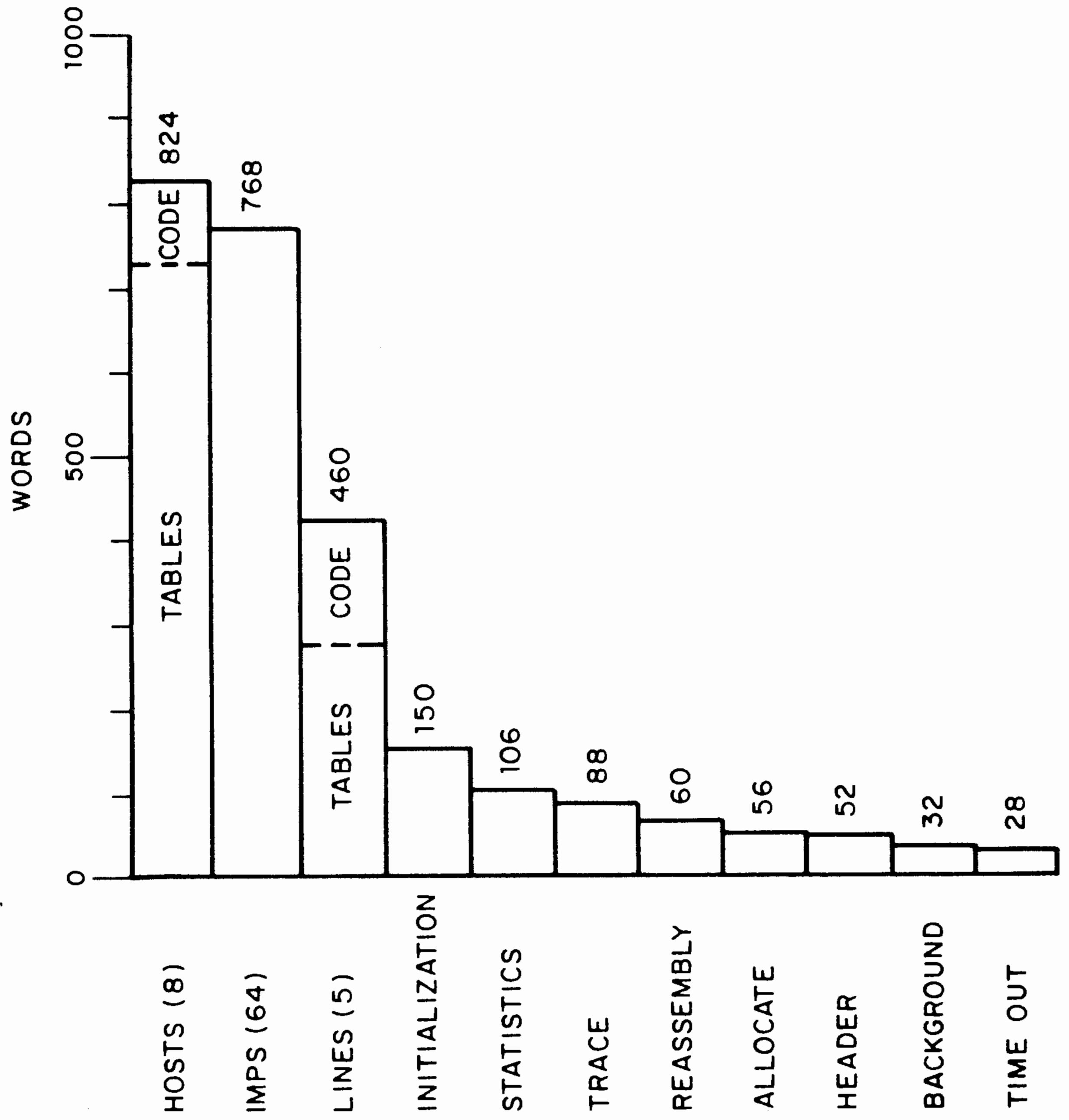


Figure 3-2. Allocation of IMP Table Storage

the proliferation of IMP configurations which has taken place was not foreseen; therefore, the program differences currently cannot be conveniently computed from a simple configuration key. Instead, the configuration irregularities must be explicitly tabled.

3.1.2 Packet Flow Through Major IMP Routines

Figure 3-3 is a schematic drawing of packet processing. The processing programs are described below. Packet flow may be followed by referring to Figure 3-3.

The *Host-to-IMP* routine (H-I) handles messages being transmitted into the IMP from a local Host. The routine first accepts the leader to construct a header that is prefixed to each packet of the message. It then accepts the first packet and, if no allocation of space exists for the destination IMP, constructs a request for buffer allocation, which it places on the task queue. Single-packet messages are placed directly on the task queue regardless of allocation status and are held via the pending packet table until either a RFNM or allocation is returned. A returned RFNM releases the packet. A returned allocation for the single-packet message will cause retransmission from the background loop. Requests for multipacket allocation are sent without actual message data. The request is recorded at the destination IMP and an allocation message is returned via the background loop when space is available. A returned allocation causes H-I to release the first packet with header to the task queue via the programmable task interrupt. Subsequent input is then accepted from the Host until end of message (EOM) occurs. The routine also tests a hardware trouble indicator and verifies the message format. The routine is serially reentrant and services all Hosts connected to the IMP.

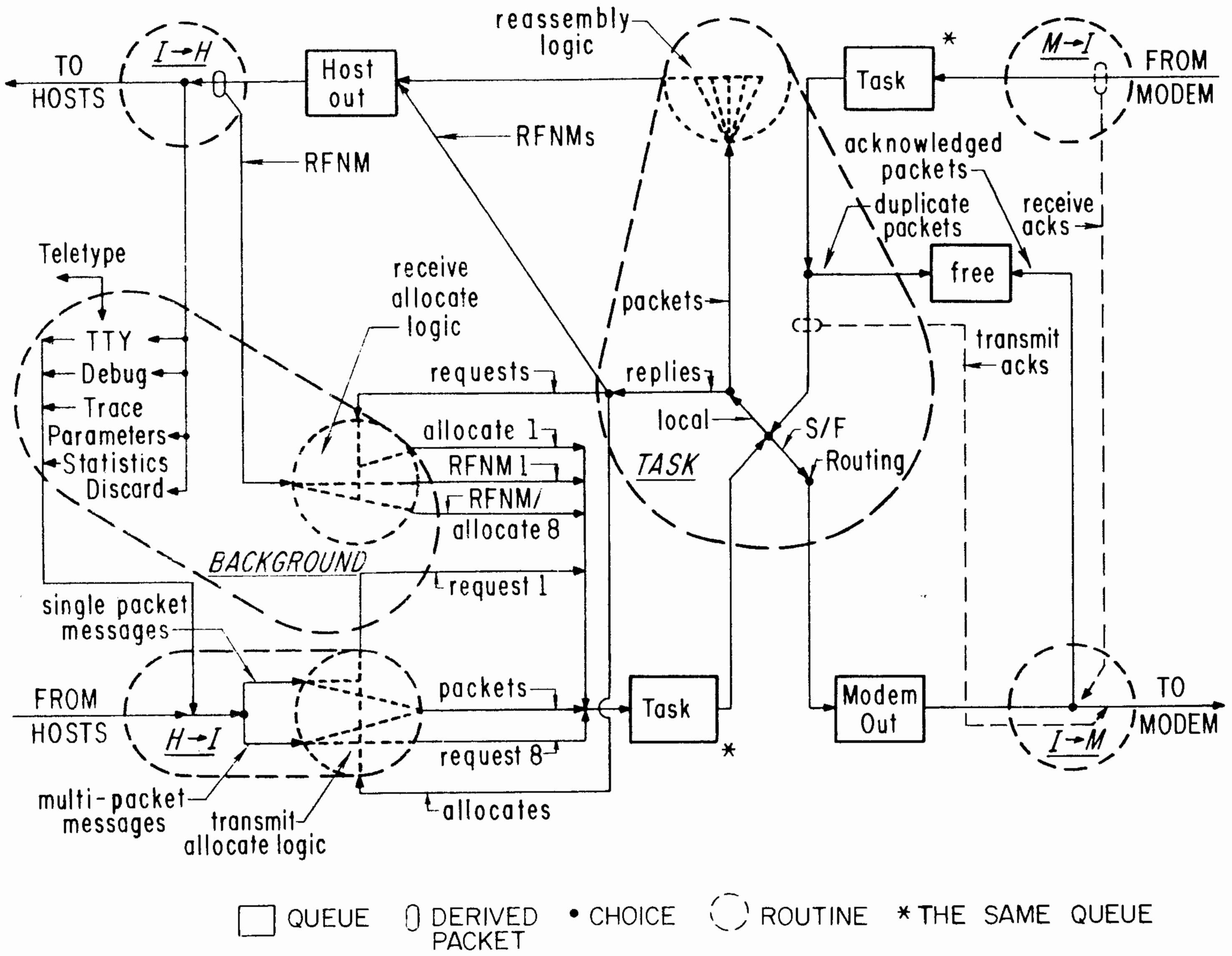


Figure 3-3. Packet Flow and Processing

The *Modem-to-IMP* routine (M-I) handles inputs from the modems. This routine consists of several identical routines, each devoted to a modem channel. Such duplication is useful to obtain higher speed. This routine first sets up a new input buffer, normally obtained from the free list. If a buffer cannot be obtained, the received buffer is not acknowledged and is reused immediately. The discarded packet will be retransmitted by the distant IMP. The routine processes returning acknowledgments for previously transmitted packets and either releases the packets to the free list or signals their subsequent release to the IMP-to-Modem routine. The M-I routine then places the buffer on the end of the task queue and triggers the programmable task interrupt.

The *TASK* routine uses the header information to direct packets to their proper destination. The routine is driven by the task interrupt, which is set whenever a packet is put on the task queue. The routine routes packets from the task queue onto an output modem or Host queue determined from the routing algorithm. If the packet is for non-local delivery, the routine determines whether sufficient store and forward buffer space is available. If not, buffers from modem lines are flushed and no subsequent acknowledgment is returned by the IMP-to-Modem routine. Normally, an acknowledgment is returned in the next outgoing packet over that modem line. Packets from Hosts which cannot get store and forward space are removed from the queue and replaced at a later time by the H-I routine.

If a packet from a modem line is addressed for local delivery, its message number is checked to see whether a duplicate packet has been received. As mentioned previously, each IMP maintains a window of contiguous numbers which it will accept from a source IMP. Packets with out-of-range numbers are considered duplicate and are discarded. The message window is four numbers wide. Thus, eight messages can be in progress between any two IMPs (four in each direction). The receipt of a RFNM or other control message by the origin Host permits the window to be moved up by one number.

Replies such as RFNMs or Dead Host messages are placed on the Host one-word queue. TASK then pokes to IMP-to-Host routine to initiate output to the Host.

Message packets for local delivery are linked together with other packets of the same message number on the reassembly queue. When a message is completely reassembled, the leading packet is linked to the appropriate Host output queue for processing by the IMP-to-Host.

Incoming routing messages are processed by the task routine by storing the neighbor's route data in temporary tables. These tables are consolidated and entered into the IMP's own route table by the timeout routine, approximately every 640 ms. The task routine also responds to I-heard-you bits in routing messages, which indicate the neighbor received the routing table last sent.

IMP-to-Modem (I-M). This routine transmits successive packets from the modem output queues and sends piggybacked acknowledgments for packets correctly received by the Modem-to-IMP routine and accepted by the task routine.

IMP-to-Host (I-H). This routine passes messages to local Hosts and informs the background routine when a RFNM should be returned to the source Host.

Initialization and Background Loop. The IMP program starts in an initialization section that builds the initial data structures, prepares for inputs from modem and Host channels, and resets all program switches to their nominal state. The program then falls into the background loop, which is an endlessly repeated series of low-priority subroutines that are interrupted to handle normal traffic.

The programs in the IMP background loop perform a variety of functions: TTY is used to handle the IMP Teletype traffic; DEBUG, to inspect or change IMP core memory; TRACE, to transmit collected information about traced packets; STATISTICS, to take and transmit network and IMP statistics; PARAMETER-CHANGE, to alter the values of selected IMP parameters; and DISCARD, to throw away packets. Selected Hosts and IMPs, particularly the Network Control Center, will find it necessary or useful to communicate with one or more of these background loop programs. So that these programs may send and receive messages from the network, they are treated as "fake Hosts." Rather than duplicating portions of the large IMP-to-Host and Host-to-IMP routines, the background loop programs are treated as if they were Hosts, and they can thereby utilize existing programs. The "For IMP" bit or the "From IMP" bit in the leader indicates that a given message is for or from a fake Host program in the IMP. Almost all of the background loop is devoted to running these programs.

The TTY program assembles characters from the Teletype into network messages and decodes network messages into characters for the Teletype. TTY's normal message destination is the DEBUG program at its own IMP; however, TTY can be made to communicate with any other IMP Teletype, any other IMP DEBUG program or any Host program with compatible format.

The DEBUG program permits the operational program to be inspected and changed. Although its normal message source is the TTY program at its own IMP, DEBUG will respond to a message of the correct format from any source. This program is normally inhibited from changing the operational IMP program; local operator intervention is required to activate the program's full power.

The STATISTICS program collects measurements about network operation and periodically transmits them to a designated Host. This program sends but does not receive messages. STATISTICS has a mechanism for collecting measurements over 10-second intervals and for taking half-second snapshots of IMP queue lengths and routing tables. It can also generate artificial traffic to load the network.

Other programs in the background loop drive local status lights and operate the parameter change routine. A thirty-two word parameter table controls the operation of the TRACE and STATISTICS programs and includes spares for expansion; the PARAMETER-CHANGE program accepts messages that change these parameters.

Other routines, which send incomplete transmission messages, send allocations, return givebacks, and send RFNMs, also reside in the background program. These routines are called Back Hosts. However, these programs run in a slightly different manner than the fake Hosts in that they do not simulate the Host/IMP channel hardware. They do not go through the Host/IMP code at all, but rather put their messages directly on the task queue. Nonetheless, the principle is the same.

Timeout. The timeout routine is started every 25.6 ms (called a fast-tick timeout period) by a clock interrupt. The routine has two sections: the fast timeout routine which "wakes up" any Host or modem interrupt routine that has languished (for example, when the Host input routine could not immediately start a new input because of a shortage in buffer space); and the slow timeout routine which marks lines as alive or dead, updates the routing tables and does long term garbage collection of queues and other data structures. (For example, it protects the system from the cumulative effect of such failures as a lost packet of a multiple packet message, where buffers are tied up in message reassembly).

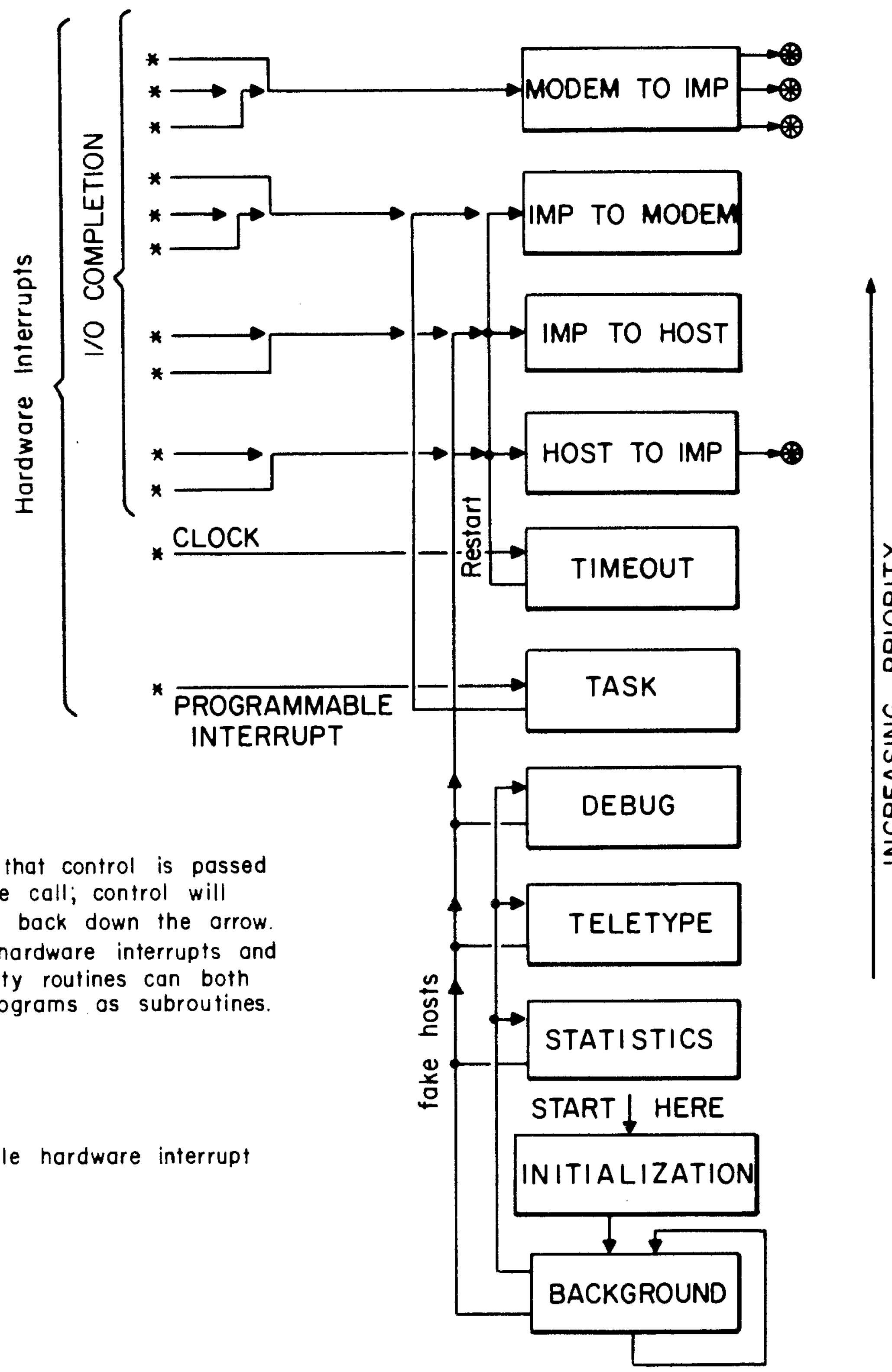
These two routines, Fast and Slow, are executed so that fast timeout runs every clock tick (25.6 ms) and the slow timeout runs every 25th clock tick (640 ms). Although they run off a common interrupt, they are constructed to allow fast timeout to interrupt slow timeout should slow timeout not complete execution before the next timeout period. During garbage collection, every table, most queues, and many states of the program are timed out. Thus, if an entry remains in a table abnormally long or if a routine remains in a particular state for abnormally long, this entry or state is garbage-collected and the table or routine is returned to its initial or nominal state. In this way, abnormal conditions are not allowed to hang up the system indefinitely.

Garbage collection may be explained by the following example. Every entry in a sixty-four entry table must be looked at every now and then. Timeout could wait the proper interval and then look at every entry in the table on one pass. However, this would cause a severe transient in the timing of the IMP program as a whole. Instead, one entry is looked at each time through the timeout routine. This takes a little more total time but is much less disturbing to the program as a whole. In particular, worst case timing problems (for instance, the processing time between the end of one modem input and the beginning of the next) are significantly reduced by this technique.

In addition to timing out various states of the program, the timeout routine is used to awaken routines which have put themselves to sleep for a specified period. Typically these routines are waiting for some resource to become available, and are written as co-routines with the timeout routine. When they are restarted by Timeout the test is made for the availability of the resource, followed by another delay if the resource is not yet available.

3.1.3 Control Organization

It is characteristic of the IMP system that many of the main programs are entered both as subroutine calls from other programs and as interrupt calls from the hardware. The resulting control structure is shown in Figure 3-4. The programs are arranged in a priority order; control passes upward in the chain whenever a hardware interrupt occurs or the current program decides that the time has come to run a higher priority program, and control passes downward only when the higher priority programs are finished. No program may execute either itself or a lower priority program; however, a program may freely execute a higher priority program. This rule is similar to the usual rules concerning priority interrupt routines.



Arrows indicate that control is passed with a subroutine call; control will eventually return back down the arrow. Note that the hardware interrupts and the lower priority routines can both call the same programs as subroutines.

⊛
Set programmable hardware interrupt

Figure 3-4. Program Control Structure

In one important case, however, control must pass from a higher priority program to a lower priority program - namely, from the several input routines to the task routine. For this special case, the computer hardware was modified to include a low-priority hardware interrupt that can be set by the program. When this interrupt has been honored (i.e., when all other interrupts have been serviced), the task routine is executed. Thus, control is directed where needed without violating the priority rules.

The practical implementation of priority control involves the setting of interrupt masks and enabling or inhibiting interrupts. Masks are built during initialization. In general, when a routine is entered, either by hardware- or software-initiated interrupt, the entering mask registers and keys are saved. A mask for the new routine is set into the mask register and the routine controls interrupts by executing INH or ENB commands. Therefore, H-I may inhibit interrupts by M-I for short periods of time during critical functions by using the INH. When the ENB command is executed, however, the mask bits for M-I will permit hardware interrupts transferring control from H-I. Interrupt control is obviously extremely critical and its use constitutes the most complex area of program operation.

Some routines must occasionally wait for long intervals of time, for example when the Host-to-IMP routine must wait for an allocation from the destination IMP. Stopping the whole system would be intolerable. Therefore, should the need arise, such a routine is dismissed, and the timeout routine will later transfer control to the waiting routine.

The control structure and the partition of responsibility among various programs achieve the following timing goals:

- No program stops or delays the system while waiting for an event.
- The program gracefully adjusts to the situation where the machine becomes compute-bound.
- The Modem-to-IMP routine can deliver its current packet to the task routine before the next packet arrives and can always prepare for successive packet inputs on each line. This timing is critical because a slight delay here might require retransmission of the entire packet.
- The program will almost always deliver packets waiting to be sent as fast as they can be accepted by the phone line.
- Necessary periodic processes (in the timeout routine) are always permitted to run, and do not interfere with input-output processes.

3.1.4 Support Software

Designing a real-time program for a small computer with many high rate I/O channels is a specialized kind of software problem. The operational program required not only unusual techniques but also extra software tools; often the importance of such extra tools is not recognized. Further, even when these issues are recognized, the effort needed to construct such tools may be seriously underestimated. The development of the IMP system has resulted in the following kinds of supporting software:

- Programs to test the hardware
- Tools to help debug the system
- A Host simulator
- An efficient assembly process

So far, three hardware test programs have been developed. The first and largest is a complete program for testing all the special hardware features in the IMP. This program permits running of any or all of the modem interfaces in a crosspatched mode; it even permits operating several IMPs together in a test mode. The second hardware test program runs a detailed phone line test that provides statistics on phone line errors. The final program simulates the modem interface check register whose complex behavior is otherwise difficult to predict.

The software debugging tools exist in two forms. Initially a simple stand-alone debugging program was designed with the capability to do little more than examine and change individual core registers from the console Teletype. Subsequently, a version of the stand-alone debugging program (DDT) was embedded into the operational program. This operational debugging program not only provides debugging assistance at a single location but may also be used in network testing and network debugging in a real-time environment.

The initial implementation of the IMP software took place without connecting to a true Host. To permit checkout of the Host-related portions of the operational program, a "Host simulator" was built that takes input from the console Teletype and feeds the Host routines exactly as though the input had originated in a real Host. Similarly, output messages for a destination Host are received by the simulator and typed out on the console Teletype. A program was also developed to automatically generate messages at a predefined rate and having predefined characteristics. The message generator routine is turned on, along with message rate, length and destination, via the parameter change fake Host program. The generator itself is embedded in the Statistics fake Host and is run during the background loop.

Without recourse to expensive additional peripherals, the assembly facilities on the DDP-516 are inadequate for a large program. (For example, a listing of the IMP program would require approximately 20 hours of Teletype output). Therefore, BBN uses other locally available facilities to assist in the assembly process. Specifically, a PDP-1D text editor is used to compose and edit the programs, which are stored on a large random access file. The file is then used as a source for a PDP-1D assembly program which assembles the IMP system, producing both object machine code and program listings.

The PDP-1D as a Host on the network provides two additional debugging aids. Core images of the current IMP system are created and maintained on the PDP-1D mass storage directly from assembled object code. When requested by the IMPVER program, an IMP's debug program dumps some or all of its core to the PDP-1D, where it is verified against the core image and discrepancies are typed out. If diagnostic or corrective patches are required in a part or all of the network, it is made up as a small IMP assembly, and the PDP-1D program BRODCAST can quickly send the patch to the required IMP(s).

Additionally, when an IMP running a very distant Host requires a fresh copy of its very distant Host module (which is not present, for example, when an IMP reloads from a neighbor), the PDP-1D's VDHLOAD program extracts this module from the IMP system binary files, broadcasts it to the IMP in question, and starts the module.

4.Ø Detailed Program Descriptions

A concise, systematic approach has been taken in presenting some details of the IMP programs in the following pages. The approach is reflected by the headings of the outline used in describing the programs:

1. *Function* - each function is numbered for reference in subsequent sections. The list of functions contains those which are fundamental to major IMP operations.
2. *Control Structure* - A general description of the coding structure and its interrupt level.
   a. Entry points - locations and modes by which the program is entered.
   b. External calls - the names of subroutine or coroutine calls which program makes.
   c. Initialization - important settings made during initialization process.
   d. Cleanup - actions taken before exiting or during unusual situations.
3. *Data Structures*.
   a. Local data - variables and constants which are used only by the program.
   b. Shared - tables, variables and constants which are used by other programs as well. Care must be taken to use interrupt locks wisely so as to insure consistency in shared data.

4. *I/O Performed*.

5. *Running Time* - an estimate of machine cycles for each function, where practical.

6. *Size* - an approximate octal number of locations used in accomplishing a function.

PROGRAM DESCRIPTIONS

X00p

PROGRAM DESCRIPTION 4.1

NAME

Modem to IMP

FUNCTION

1) Process input interrupts and initiate new modem inputs.
2) Verify packet checksums.
3) Pass packets to TASK.
4) Free acknowledged packets.
4a) Trace acknowledged packets if necessary. Copy send and acknowledge times from packet to trace block. Mark the output channel and complete indicators.

CONTROL STRUCTURE

Function 1 is performed by straight line code which is duplicated for each modem. Function 2 is performed by the shared add chain. Functions 3 and 4 are performed by shared code. Function 4a is performed by the subroutine TRCDUN which is also called by IMP to Host. Only one modem runs at a time. The entire routine runs with interrupts disabled.

ENTRY POINTS

Modem to IMP hardware interrupts come to M2In where n is modem number 1-5. These are the only entrances. No software calls are made.

EXTERNAL CALLS

There are no direct calls. The TASK interrupt is forced by the OCP TASK instructiion.

INITIALIZATION

The first input on each modem after initialization is discarded to avoid devoting a buffer to input on an unused modem.

CLEANUP

When a line goes down, input is turned off for a specified time by KILLIN and reinstated in Timeout by JSRT.

DATA STRUCTURES

LOCAL DATA

TA, TX, TK, and TAR are the save registers for the A, X registers, the keys, and the add chain return. The priority interrupt mask is not changed. The active modem number is saved in MP.

SHARED DATA

FREE- Accessed to obtain a buffer for the next input, and also to free acknowledged packets.
STQS, ETQS- Packets placed on the queue for TASK processing.
I2MTAB- Acknowledged packets freed.
I2MNXT- The packet marked as being the last one sent out a particular line is not free immediately when acknowledged, but a flag is set for modem output to perform this function.
TSEX,CHFREE- Acknowledgement bits expected on channels in use.

I/O PERFORMED

Input of header and data into a packet buffer, which is later identified as a packet, routing message, or null packet.

RUNNING TIME

1) 50 cycles
2) 2 cycles/word
3) 43 cycles
4) 13 cycles + (60 cycles * number of acks)
4a) (35 cycles)
Total for 1 ack = 166 cycles

SIZE (OCTAL)

1) 40 words * 5 modems = 240 words
2) 10 words
3) 120 words
4) 200 words
4a) (20 words)
Other 40 words
Total 620 words

PROGRAM DESCRIPTION 4.2

NAME
IMP to Modem

FUNCTION
1) Process output interrupts.
2) Free the packet last sent if acknowledged while being sent.
3) Send routing messages (including core loads).
4) Retransmit packets unacknowledged for 125 ms. verifying their checksums.
5) Send new priority packets.
6) Send new regular packets.
7) Send null packets of acknowledgements only.

CONTROL STRUCTURE
The routine performs functions 1 and 2 and then attempts to perform functions 3 through 7 in that order, as required. If none are required, the modem output is marked inactive for that channel. Only modem input interrupts are enabled. Checksum verification is performed using the shared add chain.

ENTRY POINTS
IMP to Modem hardware interrupts come to I2Mn where n is the modem number 1-5. Software interrupts come to I2MSB whenever any traffic is generated for an inactive modem. A periodic wakeup of IMP to Modem from Timeout is also necessary to perform retransmissions in the absence of other traffic.

EXTERNAL CALLS
None.

INITIALIZATION
Each modem is initialized to be down (see next section).

CLEANUP
When a line goes down, output is turned off for a specified time, then all queues and tables are garbage-collected and any packets marked for retransmission on other lines. Then output is reinstated. There is also a timeout on outputs which are not completed within 30 seconds.

DATA STRUCTURES

LOCAL DATA

TATA, TXTX, TKTK, TMTM, TARTAR are the save registers for the A register, X register, keys, interrupt mask, and add chain. The active modem number is saved in OCHN. A pointer is kept into I2MTAB for retransmissions.

SHARED DATA

1) NONE - Busy flag =0 idle, <0 busy.
2) I2MNXT - Pointer to last packet output, flagged if acknowledged while being output.
3) SLT - Flag to send a line test (routing message).
4) I2MTAB - Table of NACH slots per line, containing packet pointers.
5) SMPQ, EMPQ - Priority modem output queue.
6) SMQ, EMQ - Regular modem output queue.
7) SNULL - Flag to send a null packet of acks only.

I/O PERFORMED

3) Output of routing table from fixed core area, common to all modems.
4,5,6) Output of header and data from packet buffer.
7) Output of null packet from fixed core area, one per modem.

RUNNING TIME

1) 45 cycles
2) 30 cycles
3) 30 cycles
4) 80 cycles
5, 6) 85 cycles
7) 65 cycles

SIZE (OCTAL)

1) 50 words
2) 10 words
3) 40 words
4) 30 words
5, 6) 60 words
7) 10 words
Total 300 words

PROGRAM DESCRIPTION 4.3

NAME

IMP to Host

FUNCTION

1) Process output interrupts.
2) Mark to send RFNM and allocate if just transmitted first packet.
3) Free the packet just sent.
4) Trace packet just sent if necessary.
5) Verify checksum of and send next packet in message.
6) Send next two-word message.
7) Send next priority message.
8) Send next regular message.

CONTROL STRUCTURE

The routine performs function 1, checks to see if 2 is required, and performs function 3. Then functions 4-8 are performed if necessary. If none are, no action is taken. Modem input and output interrupts are enabled. Checksum verification is performed by the shared add chain.

ENTRY POINTS

IMP to Host hardware interrupts come to IHnE, where n is the host number (0-3). Software interrupts come to IHSB whenever traffic is generated for an inactive host, whenever Timeout has ticked over a host's 'alarm clock', or whenever a fake host has finished with a packet. The TIP's software calls IMP to Host via its associated hardware entry (currently IH2E).

EXTERNAL CALLS

2) RALLYP - Enters allocate and RFNM into RALLY table for use by Background.
4) TRCDUN - Traces a packet - also called by Modem to IMP.
5) HTPPF - Counts a packet of throughput for trouble reports.
7)-8) HTPMF - Counts a message of throughput.

INITIALIZATION

All real hosts are initialized to be held down for a specific delay (30 seconds), to empty all their queues, and to send 3 NOPs. This action is begun before Background runs.

CLEANUP

All two-word entries are cleared and all packets on a host's priority and regular queues are freed, the host's ready line is flapped, and an 'Interface Reset' message returned to the host, if:

a) A RFNM is in transmission for more than a specified time, or

b) Any message is on a host queue for more than that specified time.

That maximum is currently set to 30 seconds.

DATA STRUCTURES

LOCAL DATA

The X register, A register, Mask, add chain return, and keys are stored in IHX, IHA, IHMS, IHAR, and IHK. IHP contains the currently running host number. Tables indexed by host include:
IHLO - Saved co-routine reentry points.
IHSP - Saved buffer pointers.
IHWQ - Saved host queue pointers.
IHLSTP - Last-packet flag.

SHARED DATA

1) IHTT - Timeout alarm clock, used to wake up IMP to Host if a packet is too long in transmission.
2) RALLY - Table of receive allocates.
3) FREE - Queue of free buffers.
5) HTPMFL, HTPMFN - Message throughput counters.
6) SHWQ, SHBQ - Host two-word message queues and their buffers' queues.
7) SHPQ - Host priority queue.
8) SHQ - Host regular message queue.
7)-8) HTPPFL, HTPPFN - Packet throughput counters.
Also, cleanup and initialization use HINWAT to inform Host to IMP that the host is to be held off.

I/O PERFORMED

5)-8) OCP to initiate a middle-output or a final-output is issued.
Also, cleanup and initialization enable and disable the host interface.

RUNNING TIME

4) (35 cycles)

SIZE (OCTAL)

4) (20 words)

PROGRAM DESCRIPTION 4.4

NAME

Host to IMP

FUNCTION

1) Process input interrupt.
2) For input of control message, take appropriate action, and initiate input.
3) For input of regular message leader, begin processing of message, and initiate input of first packet of message.
4) For input of first packet, if destination has not allocated space, initiate request.
5) For input of first packet, if destination has allocated space, process packet and initiate input of subsequent packet.
6) For input of subsequent packet, process the packet and initiate input of subsequent packet.

CONTROL STRUCTURE

Modem input and output and host output interrupts are enabled. Function 1 is performed, then one of the remaining functions, with processing resuming from the last co-routine exit. Processing of packets and requests for allocation includes generation of checksum using the shared add chain.

ENTRY POINTS

Host input hardware interrupts come to HInE, where n is the host number (0-3). Software interrupts come to HISB whenever Timeout has ticked over a host's 'alarm clock', when Task has processed a received allocate, or when a fake host has a packet to send. The TIP's software calls Host to IMP via its associated hardware entry (currently HI2E).

EXTERNAL CALLS

FLUSH - Returns a packet to the free list if it was on only one queue.
OWP - One Word Put - Sends control messages to host.
MESGET - Assigns a message number.
PPTPUT - Mak5s entry in Pending Packet Table.
PLTPUT - Makes entry in Pending Leader Table.
TALLYG - Retrieves transmit allocate from TALLY if any.
HTPMT - Counts a message of throughput for trouble report.
HTPPT - Counts a packet of throughput.

INITIALIZATION

Initialization is accomplished by setting HILO, the co-routine entry points and EMFH, the end test instructions. Real hosts start by discarding the initial input. Fake hosts start by expecting input of a leader. Initial input is blocked according to HINWAT, which is controlled by IMP to Host and Timeout.

CLEANUP

A host is initialized as above when an error is detected, when a host has been down, or when an input takes too long (currently when over 15 seconds elapses between leader and last bit).

DATA STRUCTURES

LOCAL DATA

HIA, HIX, HIK, HIAR, HIMS are used to save the A and X registers, keys, add chain return, and mask. HIP saves the number of the calling host. Tables indexed by host include:
HISP - Saved buffer pointers.
HILO - Saved co-routine entry points.
EMFH - Host's end of message test instruction.
HIDEST - Saved destination of current message.
HILINK - Pointer to saved link or sub-code.
HIH0, HIH1, HIH2, HIH3 - Used for building headers.

SHARED DATA

HITT - Timeout alarm clock.
HINWAT - Flag from IMP to Host to block input.
FREE - Queue of free buffers.
DHT - Dead Host Table.
RUT - Route Use Table, used to see if an IMP is up.
STQ - Task queue.
PPT - Pending Packet Table.
PLT - Pending Leader Table.
TALLY - Table of transmit allocates outstanding.
HTPMTL, HTPMTN, HTPPTL, HTPPTN - Message and packet throughput counters.

I/O PERFORMED

OCP's to initiate input; SKS's to test for errors and end of message; and input from the Real time clock.

PROGRAM DESCRIPTION 4.5

NAME

Timeout

FUNCTION

1) Service the 25.6 ms clock interrupt and count time in 25.6 ms units.
2) Call JOBF, Fast Timeout, 24 out of 25 ticks.
3) Call JOBS, Slow Timeout, 1 out of 25 ticks.

CONTROL STRUCTURE

The Slow Timeout routine enables clock interrupts. That is, it is constructed to be able to interrupt itself. It keeps a table of saved registers for the Slow and Fast Timeout routines, and keeps track of which routine to run at the next clock tick. In this way, Fast Timeout may interrupt Slow Timeout. All Modem and Host input and output interrupts are also enabled The Slow Timeout routine falls into Fast Timeout, which thus in effect runs every tick.

ENTRY POINTS

Hardware 25.6 ms clock interrupt only.

EXTERNAL CALLS

None.

INITIALIZATION

The first Timeout is initialized to be a Slow Timeout tick. All Timeout programs run before background runs for the first time.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

TOA, TOCK, TOX, TOC, TOMK, TOB are the saved A register, keys, X register, return address, interrupt mask, and next-time-to-run timer for each of the Timeout routines.

SHARED DATA

None.

I/O PERFORMED

None.

RUNNING TIME

1) 85 cycles
2) 450 cycles

SIZE (OCTAL)

1) 200 words
2) 210 words
3) 1000 words
Total 1410 words

PROGRAM DESCRIPTION 4.5.1

NAME

Fast Timeout

FUNCTION

1) RSTOUT - Send routing, determine line status.
2) IMTC - Periodic wakeup of IMP to Modem.
3) HITC - Periodic wakeup of Host to IMP.
4) SWCH - Monitor flags and switches which generate immediate NCC reports.
5) TALLYB - Time out unused allocates for 8 buffers in 125 ms.

CONTROL STRUCTURE

A series of subroutine calls, one for each function.

ENTRY POINTS

Called directly by Timeout 4 of every 5 ticks and by Slow Timeout 1 of every 5, or each 25.6 ms.

EXTERNAL CALLS

1) DEDL - Timing and routing on a dead line.
   KILLIN (from DEDL) - Kill a line.
   RSTINP (from DEDL) - Process a (dummy) routing input.
   JSRT (from DEDL) - Bring up a dead line.
2) I2MSB - Software interrupt of IMP to Modem.
3) HISB - Software interrupt of Host to IMP.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

1) RSTDT - Temporary index register.
RMCLKS - Table of clock counters for line speeds.
RMBIT - Table of routing flag counters, one per line.
LTR - Line going down counter.
SENR - JSRT Temporary.
JSRTQ - JSRT Temporary.

2) IMTK - Loop counter to attempt to wake up each modem output in numerical order.

3) HITK - Loop counter to attempt to wake up each host input in random order.

SHARED DATA

1) SLT - Line test/dead flag.
E123 - Line error counter.
LAC - Line alive counter.
LINE - Line alive dead status.
NEIGHB - Line's neighbor IMP number.
LEND - LOW NUMBERED END BIT.
RTRCVD - LINE TESTS RECEIVED.
RTSSNT - LINE TESTS SENT.
I2MTAB - Table of modem output pointer slots.
ERQ - Retransmit queue.
TSEX, RSEX - Acknowledge bits.

2) NONE - An alarm clock for IMP to Modem to indicate whether each modem is waiting for hardware interrupts (<0 and a timer) or idle (=0) and thus requires software wakeup.

3) HITT - A similar alarm clock for Host to IMP. If Host to IMP is waiting for a resource or timing out an input, it uses HITT as its alarm clock.

4) Flags which are monitored for immediate NCC reports-
RSFNCC - A restart/reload/wdt/power fail indicator.
HLTLOC - The address of the last pseudo-halt.
HLNM - The number of the host interface under test.
TRON, SNON, SON, MGON - the status of various statistics programs.
Also, the state of memory protect and sense switches.

5) TALLYI, TALLYC - Pointer and timer for allocates.

I/O PERFORMED

None.

RUNNING TIME

1) 100 cycles
3) 250 cycles
4) 75 cycles
5) 10 cycles
Total 450 cycles

SIZE (OCTAL)

1)	300 words
2)	30 words
3)	30 words
4)	60 words
5)	10 words
Other	30 words
Total	510 words

PROGRAM DESCRIPTION 4.5.2

NAME

Slow Timeout

FUNCTION

1) IHTC - Periodic wakeup of IMP to Host.
2) DEDH - Host alive/dead status check.
3) JED - Compute effective delay.
4) RUTCLK - Periodic routing functions.
5) RUTOUT - Compute new routing tables.
6) HTEST - Perform interface loop/unloop/clear.
7) HPOKE - Test host interfaces with data.
8) DEDI - Garbage-collect after dead IMPs.
9) JUQC - Update queue counters.
10) MESSTO - Mark messages unanswered in 30 seconds.
11) Verify that add chain is intact.
12) VDH.TO - VDH Timeout call.

CONTROL STRUCTURE

A series of subroutine calls, one for each function, except (11), which is in line.

ENTRY POINTS

Called by Timeout 1 out of every 25 clock ticks, or every 640 ms.

EXTERNAL CALLS

1) IHSB - Periodic wakeup of IMP to Host.
2) IHST - IMP to Host is restarted if the Host is detected to be hardware-down.
8) PPTGET - Find pending single packet message.
   PLTGET - Find pending multipacket message.
   OWP - Put one word message.
   TALLYG - Get pending allocates.
   upmess - Increment message table entry.
   MESINI - Initialize message table entries.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

1) IHTK - Loop counter.
2) DHC - Loop counter.
3) JDEC - Loop counter.
   JEDP - Pointer to modem slot tables.
6) HTOLD - Saved HTPAR flag.
   HTINTF - Interface under test.
8) DEDIMP - Dead IMP number.
   DMSTAB - Pointer to message table entry.
   DTEND - Temporary queue pointer.
   IMPDLH - Local source host.
10) MESST - Timeout clock for message table.

SHARED DATA

1) IHTT - Alarm clock for IMP to Host.
3) CED - Table of computed effective delays.
7) HTPAR - Function and interface test command.
8) TMESS - Transmit message table.
   PPT - Pending packet table.
   PLT - Pending leader table.
9) All queue counters.
10) TMESS - Transmit message table.

I/O PERFORMED

None.

RUNNING TIME

1) 150 cycles
2) 200 cycles
3) 125 cycles
4) 300 cycles per IMP = 20000 cycles
5) cycles
6) 40 cycles
7) 40 cycles
8) 25 cycles per live IMP = 1600 cycles
   60 cycles per dead IMP plus needed cleanup
9) 350 cycles
10) 30 cycles per IMP = 2000 cycles (every 15 seconds)
11) 825 cycles
12) 30 cycles

Total 5000 cycles.

SIZE (OCTAL)

1) 30 words
2) 60 words
3) 20 words
4) 120 words
5) 120 words
6) 60 words
7) 20 words

8) 220 words
9) 30 words
10) 30 words
11) 20 words
12) 30 words
Total 1040 words.

PROGRAM DESCRIPTION 4.6

NAME

TASK

FUNCTION

1) Process interrupts and service the TASK queue.
2) Perform duplicate detection if modem input.
3) For routing input, copy information and monitor line up/down status.
4) For packet input, branch to either TASK Store-and-forward or TASK For Us.
5) Return an ACK or a NACK.

CONTROL STRUCTURE

TASK has the logical structure of a subroutine called by Modem or Host input, which provides two returns- input accepted and input rejected. It is implemented as an interrupt routine, called by a settable interrupt. It processes all inputs on the TASK queue and then dismisses. All interrupts except TASK are enabled.

ENTRY POINTS

Called by Modem to IMP, Host to IMP, and Background, by programmed interrupt.

EXTERNAL CALLS

5) I2MSB - Software interrupt of IMP to Modem if necessary to return acknowledgement.
HISB - Software interrupt of Host to IMP to communicate acceptance or rejection of the input.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

TA, TX, TK are the saved A register, X register, and keys. The interrupt mask is known to be all ones.
THIS - Pointer to the packet TASK is processing.
2) ACKP, ACKBIT - Pointer to RSEX and bit pointer corresponding to input channel.

SHARED DATA

1) STQ,ETQ - TASK queue.
2) RSEX - Receive odd/even bits.
3) NRT, NEIGHB, LAC, LEND, E123 -
The routing temp tables, and line up/down flags.
4) RUT - Route Use Table, =0 for us, <0 IMP dead, otherwise branch to TASK Store and Forward.
5) NONE, RSEX, TSKFLG - communication registers for TASK and input routines.

I/O PERFORMED

None.

RUNNING TIME

1) 40 cycles
2) 20 cycles
3) in flux
4) see 6.1, 6.2
5) 20 cycles

SIZE (OCTAL)

1) 40 words
2) 20 words
3) 100 words
4) see 6.1, 6.2
5) 60 words

PROGRAM DESCRIPTION 4.6.1

NAME

TASK Store and Forward

FUNCTION

1) Select output line for the packet.
2) Search for an output slot on that line.
3) Check and update the storage utilization counts.
4) Assign the packet to an output slot.
4a) Trace the packet if necessary.
    Acquire a trace block from the free trace list.
    Put a pointer to the block in the packet.
    Copy the packet header and input time into the block.
5) Put the packet on an output queue.
6) Call IMP to Modem if it is idle.

CONTROL STRUCTURE

TASK Store and Forward is designed to run in the minimum time necessary to perform functions 1-6. The routine is almost completely straight line code. Function 2 has the from of a loop. Function 4a is performed by the subroutine TSUB which is also called by TASK Reassembly. Functions 3,5, and 6 run with interrupts locked, otherwise interrupts are enabled.

ENTRY POINTS

Function 1 is actually a test to determine whether the packet is destined for this IMP , in which case the code branches to FORUS, or whether it is traffic for some other IMP.

EXTERNAL CALLS

6) Software interrupt of IMP to Modem if it is idle.

INITIALIZATION

None.

CLEANUP

The modem output slots and queues are garbage-collected in JSRT when a line goes down (see 5.2).

DATA STRUCTURES

LOCAL DATA

1) OURR - Our route to send the packet out.
2) I2MSLT - Output slot to use for the packet.
4) ACKCH, I2MBIT - Channel corresponding to slot, and bit pointer for associated odd/even bit.
4a) SPB, IMB - Packet pointers, standard and post-indexed.
    STB, ITB - Trace block pointers, standard and post-indexed.

SHARED DATA

1) RUT - Route Use Table.
2) I2MTAB, I2MEND - Output slot pointers.
   CHFREE - Bits indicate slots in use.
3) NSFA, NSFS, MAXSI - Store and Forward storage utilization counts and limit.
   NFA, NFS, NALA, NALS, MINF - Free storage utilization counts and limit.
4) TSEX, LEND - Transmit odd/even bits and line high end/low end flag.
4a) TTF, STRQ - Free and active trace blocks.
5) EMPQ, EMQ - End of modem priority and regular queues.
6) NONE - Modem output busy flag.

I/O PERFORMED

None.

RUNNING TIME

1) 10 cycles
2) 10 cycles + 10 cycles/slot
3) 20 cycles
4) 30 cycles
4a) 5 cycles (+ 100 cycles)
5) 20 cycles
6) 5 cycles
Total 125 cycles

SIZE (OCTAL)

1) 10 words
2) 10 words
3) 20 words
4) 30 words
4a) 5 (+ 50 words)
5) 20 words
6) 5 words
Total 120 words

PROGRAM DESCRIPTION 4.6.2

NAME

TASK For Us

FUNCTION

1) Check message number, discard out-of-range and duplicate messages except incomplete transmissions which receive replies.
For transmissions:
2) Dispatch on destination host up/down, single or multi-packet, and request for allocation or not.
3) If the destination host is up, process single packet requests as messages if storage present, otherwise save a request for allocate. If the host is down, return a destination dead message.
4) For multi-packet requests for allocation, save the request whether the host is up or down.
5) If the destination host is up, find the multi-packet message in reassembly storage or start a new reassembly block for it. If the host is down, return a destination dead message.
6) If the multi-packet message is complete, or if it is a single packet, mark the receive message number as complete. Search the message stack in a loop finding the next message to give to host output from this source IMP.
7) If the next message is found, remove it from the stack and put it on tht host queue, acquiring a trace block for it if necessary. Remove multi-packet messages from their reassembly blocks and free the blocks. Call host output and increment the message number.
For replies to single-packet messages:
8) Find the PPT entry.
9) If an allocate, mark the entry for retransmission.
10) If not, format a leader and call OWP, clear the PPT entry and mark the message number completed.
For replies to multi-packet messages:
11) Find the PLT entry.
12) If it is an allocate, put an entry in the TALLY table.
13) If it is not just an allocate, format a leader and call OWP, clear the PLT entry and mark the message number completed.

CONTROL STRUCTURE

TASK For Us is a tree-structured group of code, most of which is straight line with some subroutine calls. Many sections run with interrupts locked, some because of shared data, others because of shared code.

ENTRY POINTS

In TASK, when the destination of the packet is determined to be this IMP, the code branches to TASK For Us.

EXTERNAL CALLS

3,4) RALLYP - Put an entry in RALLY.

INITIALIZATION

1) Message numbers are initialized in MESINI.
2) Hosts are initialized to be down.

CLEANUP

1) Message numbers are timed out in MESSTO and incomplete transmissions are sent back by BACK1.

DATA STRUCTURES

LOCAL DATA

MESTAB - Pointer to message table.
MESNUM- Message number for this packet.
MESBIT - Bit corresponding to this packet.
PKTN - Packet number for this packet.
MESSID - Message number and source IMP for this packet.
ORB - Our reassembly block.
ORS - Our reassembly slot.
TEND - Temporary queue end pointer.
READY - Pointer to next packet to give to host.
READYE - Queue end pointer corresponding to READY.
NPKTS - Packet counter for reassembly.
SOURCE - IMP number of source of this packet.
PPTASK - Pointer to PPT or PLT for this reply.
TWDPA - First word of leader to give to host.
LOCHNO - Number of host to give message.
LOCHST - Number of host to give message.

SHARED DATA

1) TMESS, RMESS - Transmit and receive message numbers.
SRQ, ERQ - Retransmit queue.
2) HIHD - Host up/down indicator.
3) NFA,NFS,NREA,NRES,NALA,NALS - Storage utilization counters.

I/O PERFORMED

None.

DATA STRUCTURES

LOCAL DATA

MESTAB - Pointer to message table.
MESNUM- Message number for this packet.
MESBIT - Bit corresponding to this packet.
PKTN - Packet number for this packet.
MESSID - Message number and source IMP for this packet.
ORB - Our reassembly block.
ORS - Our reassembly slot.
TEND - Temporary queue end pointer.
READY - Pointer to next packet to give to host.
READYE - Queue end pointer corresponding to READY.
NPKTS - Packet counter for reassembly.
SOURCE - IMP number of source of this packet.
PPTASK - Pointer to PPT or PLT for this reply.
TWDPA - First word of leader to give to host.
LOCHNO - Number of host to give message.
LOCHST - Number of host to give message.

SHARED DATA

1) TMESS, RMESS - Transmit and receive message numbers.
SRQ, ERQ - Retransmit queue.
2) HIHD - Host up/down indicator.
3) NFA,NFS,NREA,NRES,NALA,NALS - Storage utilization counters.

I/O PERFORMED

None.

RUNNING TIME

1)	50 cycles
2)	30 cycles
3)	30-100 cycles
4)	50 cycles
5)	35-150 cycles

SIZE (OCTAL)

1)	100 words
2)	40 words
3)	100 words
4)	5 words
5)	70 words

PROGRAM DESCRIPTION 4.7

NAME

Background

FUNCTION

1) Call each Fake Host input process.
2) Call each Fake Host output process.
3) Call each Back Host process.
4) Run the nice-stop code if necessary.
5) Calculate the light register display.
6) Call VDH Background.

CONTROL STRUCTURE

Background runs in a loop, performing all of its functions repeatedly. All interrupts are enabled for the most part, so Background runs when no other more important processes are running. Thus its functions can be characterized as periodic but non-essential. Functions 1 through 4 are called as coroutines, each background call returns where the previous call left off.

ENTRY POINTS

Entered from Initialization and run continuously thereafter.

EXTERNAL CALLS

1) Return by calling DOZE.
2) Return by calling WAIT.
3) Return by calling SLEEP.

INITIALIZATION

The coroutine entries for 1 through 4 are initialized.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

1 and 2) FAKENO - The number of the Fake Host last run.
3) BACKNO - The number of the Back Host last run.
1) DZTB - Table of saved return addresses.
2) WTTB - Table of saved return addresses.
3) SLTB - Table of saved return addresses.

SHARED DATA

None.

I/O PERFORMED

5) Light display output from the A register.

RUNNING TIME

5) 100 cycles

SIZE (OCTAL)

4) 100 words
5) 30 words

PROGRAM DESCRIPTION 4.7.1

NAME

SUCK

FUNCTION

Simulate the IMP-to-Host interface hardware for the Fake Hosts. Wait until the next output is sent, the fetch each word through the output pointer, and increment the pointer. If the buffer is empty (the output and end pointers are equal) and it is a final output, set the end-of-message indicator for the Host. If the output and end pointers cross, give an output interrupt. Return when a new word is ready for output.

CONTROL STRUCTURE

SUCK is called by each Fake Host output process as a suboutine to receive one word from the IMP. It returns when a word is ready, and makes a coroutine return to the main background loop if there is no output ready.

ENTRY POINTS

Called by:
Fake Host 0 - TTY output.
Fake Host 1 - DDT output.
Fake Host 2 - PARAMETER CHANGE.
FAKE HOST 3 - DISCARD.

EXTERNAL CALLS

IHSB - Software interrupt of IMP-to-Host.
WAIT - Coroutine return to Background.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

SUCT - Table of saved return addresses.

SHARED DATA

IHBB - Simulated DMC output pointers.
IHBC - Simulated DMC output end pointers.

I/O PERFORMED

None.
(Simulated I/O includes output transfers,
and software interrupt on output buffer empty.)

RUNNING TIME

17 cycles per word.

SIZE (OCTAL)

40 words

PROGRAM DESCRIPTION 4.7.1.1

NAME
TTY SUCK

FUNCTION
Outputs messages sent to fake host TTY
1) as ASCII characters if parity bit is on, or
2) as octal numbers, after an optional
3) octal print of the leader, utilizing a
4) common send character routine.

CONTROL STRUCTURE
A strung out loop which first locks out TTY input, saves the source if foreign, and commits the octal print bit (2) to memory before printing anything. It then octal prints the header (3) if TTY JAM last sent a multi-character message and the host simulator flag was on then. Otherwise it skips over the second word of the leader noticing if it was the last word and if it is, checks the subcode and types a backslash if it is non-zero. If the link is not the last word, the non last data words are either printed out as octal numbers (2) using an octal print routine or as ASCII characters (1) using a routine (4) shared by the octal print routine. This common send routine (4) rejects characters with zero parity and sucks up the rest of the message if output is in progress. This requires that other TTY sucks keep the send routine informed as to whether they are the last so send doesn't mistakenly throw away the next message!

ENTRY POINTS
Coroutined with SUCK/WAIT

EXTERNAL CALLS
None

INITIALIZATION
None

CLEANUP
None

DATA STRUCTURES

LOCAL DATA

```
TTOW - last word returned by SUCK (1,2,3,4)
TTNM - set while processing last word (1,2,3,4)
OCTL - set minus for octal print (2)
OC01 - octal print digit counter (2,3)
OC03 - octal print temporary (2,3)
```

SHARED DATA

```
HSGO - HSFG at beginning of last TTY JAM message
WHOTTY - last foreign source
```

I/O PERFORMED

None

RUNNING TIME

```
1) 55+odd(6)+2(SUCK)+characters(23+SUCK+2(Send))
2) 55+odd(6)+2(SUCK)+words(13+SUCK+Octal)
3) add 24+3(Send)+2(Octal)
   2 word HS total is 74+2(SUCK)+3(Send)+2(Octal)
Send: 23+waits(11)
  if parity is off, 5
  if interrupts output, 19-ttnm(2)+wordsleft(8+SUCK)
  if TTY  wont  output, 21-ttnm(2)+wordsleft(8+SUCK)
Octal: 131+6(Send)
\ (i.e. incomplete transmission): 64+2(SUCK)+Send
nothing (e.g. rfnm): 58+2(SUCK)
```

SIZE (OCTAL)

```
1) 56
2) 3
3) 15
HS two word) 3
Send) 31
Octal) 25
\) 11
```

PROGRAM DESCRIPTION 4.7.1.2

NAME

DDT SUCK

FUNCTION

Feed messages a character at a time to DDT

CONTROL STRUCTURE

DDT SUCK is coroutined with DDT JAM to form one DDT process. The DDT SUCK process saves the SUCK message leader for DDT JAM and resets BBNF if the message is from the PDP-1D, or TTY at IMP 5 or 30. It then breaks each word up into characters and calls a subroutine which gives them to DDT JAM (via DDTC) and waits for them to be taken (DDT JAM zeroes DDTC when it takes a character.) However if the parity bit is not set it returns immediately, and if it is a break it sets the DDT_JAM wait return to the DDT reset address and resets the suppress output flag (DDTI which TTY JAM may have set.) At the end of a message it sets a flag at which DDT JAM will look before its next read and if it is set will close its JAM message.

ENTRY POINTS

Coroutined with SUCK/WAIT

EXTERNAL CALLS

None

INITIALIZATION

Done by TTY JAM

CLEANUP

None

DATA STRUCTURES

LOCAL DATA
DINW - word returned by SUCK

SHARED DATA
DINC - character for DTT input routine

I/O PERFORMED
None

RUNNING TIME
32+2(SUCK)+words(32+SUCK+waits(8+WAIT))
add 19+waits(8+WAIT) for odd byte

SIZE (OCTAL)
54

NAME

Parameter Change

FUNCTION

Accept new values for the parameters govening the operation of the Trace and Statistics programs. The format of a message to Parameter Change is a set of two word pairs. The first word gives the number of the parameter to change, and the second word gives the new value for the parameter.

CONTROL STRUCTURE

Parameter Change has the form of a loop, with a coroutine call to SUCK functioning as the implied wait. Interrupts are enabled.

ENTRY POINTS

Parameter Change runs whenever SUCK has a word from the IMP.

EXTERNAL CALLS

None.

INITIALIZATION

All parameters are initialized to zero. Periodically, those parameters necessary for NCC reports and diagnostic messages are set to their nominal values.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA
BTR2 - A pointer into PARAMT.

SHARED DATA
PARAMT - The table of parameters:

0) TRON - Trace on/off
1) SNON - Snapshot statistics on/off
2) SON - Cumulative statistics on/off
3) MGON - Message Generator on/off
4) DIAGON - diagnostic reports on/off
5) TPON - NCC reports on/off
6)

7) TLNK - Trace link
10) STATL - Links for: Snapshot statistics
11) Cumulative statistics
12) Message Generator
13) Diagnostic reports
14) NCC reports

15) TDST - Trace destination
16) STATD - Destinations for: Snapshot statistics
17) Cumulative statistics
20) Message generator
21) Diagnostic reports
22) NCC reports

23) TF - Auto Trace frequency
24) STATF - Frequencies for: Snapshot statistics
25) Cumulative statistics
26) Message Generator
27) Diagnostic reports
30) NCC reports

31) MGNL - Message Generator length

I/O PERFORMED
None.

RUNNING TIME
10 cycles + 2 calls to SUCK per parameter
= 50 cycles per parameter

SIZE (OCTAL)
20 words

PROGRAM DESCRIPTION 4.7.1.4

NAME

Discard

FUNCTION

Discard is a process which simply accepts messages from the IMP via SUCK. It is used to guarantee the return of a RFNM or Incomplete Transmission to a source Host, by virtue of the standard IMP to Host mechanisms and the fact that Discard is always a responsive Host. In particular, when a destination Host takes too long to accept a message, or goes down when the IMP is holding messages for it, the messages are marked incomplete and put on Discard's queue. When the messages are accepted by Discard and thrown away, an Incomplete Transmission is automatically sent back to the source Host.

CONTROL STRUCTURE

Discard has the form of a loop, with a coroutine call to SUCK functioning as the implied wait. Interrupts are enabled.

ENTRY POINTS

Discard runs whenever SUCK has a word from the IMP.

EXTERNAL CALLS

None.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

None.

SHARED DATA

None.

I/O PERFORMED

None.

RUNNING TIME

20 cycles per word

SIZE (OCTAL)

3 words

PROGRAM DESCRIPTION 4.7.2

NAME

JAM

FUNCTION

Simulate the Host-to-IMP interface hardware for the Fake Hosts. Receive a word from the Host, store it through the input pointer, and increment the pointer. If the end-of-message indicator is on, or the buffer is full (the input and end pointers cross), give an input interrupt. Wait until a new input is possible, and then return.

CONTROL STRUCTURE

JAM is called by each Fake Host input process as a subroutine to send one word to the IMP. It returns when that word has been taken and another input is logically possible. If another input is not possible, it makes a coroutine return to the main background loop.

ENTRY POINTS

Called by:
Fake host 0 - TTY input.
Fake host 1 - DDT input.
Fake Host 2 - Trace input.
Fake Host 3 - Statistics input.

EXTERNAL CALLS

HISB - Software interrupt of Host-to-IMP.
DOZE - Coroutine return to Background.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

GAMT - Table of saved return addresses.

SHARED DATA

HIBB - Simulated DMC input pointers.
HIBC - Simulated DMC input end pointers.

I/O PERFORMED

None.
(Simulated I/O includes input transfers, and software interrupt on end-of-message and input buffer full.)

RUNNING TIME

16 cycles per word

SIZE (OCTAL)

40 words

PROGRAM DESCRIPTION 4.7.2.1

NAME

TTY JAM

FUNCTION

1) Process teletype interrupts and input typed characters or echo backslash if last character not taken.
2) Send single character message to crosspatch destination, and if a break, reset crosspatch destination to local DDT.
3) Send multi-character messages to message host and link.
4) Send octal numbers within multi-character messages.

CONTROL STRUCTURE

Function 1 is performed by an interrupt routine which dismisses output completion interrupts for tty output and backslashes, types a backslash if the last character has not been taken, and reads the tty and leaves the character, and an indication of its arrival for the tty fake host i.e. the tty jam background process. Function 2 is performed by two coroutines one of which either returns with a character with its parity bit ored on or dozes while waiting for the teletype interrupt routine. The other checks if the character is the message escape character "";"" and jumps into a function 3 coroutine if it is. Otherwise it JAMS the crosspatch header, zero link, the character left justified, resetting the crosspatch header to local DDT for a break, and padding. Function 3 is performed by two coroutines one of which uses function 2's read/doze routine to get characters, jumps back to the function 2 jam coroutine for a " ; " and jumps into the function 4 routine for the number escape character " : ". The other coroutine JAMS the message header and link, and then loops, two characters to a word. Function 4 is performed by the function 3 read coroutine and a coroutine which accumulates an octal number from the three least significant bits of successive characters, terminating on carraige return, echoing a linefeed, and jumping back into the function 3 JAM coroutine.

ENTRY POINTS

Coroutined with JAM/DOZE

EXTERNAL CALLS

None

DATA STRUCTURES

LOCAL DATA

TTFG	- interrupt flag word
TINA	- interrupt A register save
TTCR	- interrupt character
TTCH	- Read/DOZE character
TTIW	- word so far

SHARED DATA

OTGO	- output in progress flag
HSFG	- Host Simulator flag
HSGO	- Host Simulator flag at beginning of last message

I/O PERFORMED

TTY input and output

RUNNING TIME

1) extraneous(!) interrupt: 14
   output interrupt: 28
   \ output interrupt: 25
   last not taken and \: 26
   last not taken and no \: 28
   last taken and character read: 31
2) 92+break(6)+3(JAM)
3) 28+3(JAM)+characters/2(30+JAM)
4) words(17)+25+JAM+dozes(10+DOZE)

SIZE (OCTAL)

1) 35
2) 61
3) 30
4) 37

Initialization) 16

2/73

INITIALIZATION

Zeroes OTGO, HSFG, HSGO, and TTCH; AND SETS TTY crosspatch destination to local DDT. Zero DINC and set DDT destination to local TTY for DDT. Set interrupt mask enabling TTY and disabling all others.

CLEANUP

None

PROGRAM DESCRIPTION 4.7.2.2

NAME

DDT JAM

FUNCTION

Output DDT characters to the last DDT SUCK source

CONTROL STRUCTURE

DDT and its input and output routines are run on the DDT JAM process. The DDT input routine first tests if it has encountered a message end and if it has clears the message end flag and calls part of the DDT output routine to send off any output that has been done (i.e. end the current JAM message.) If the test fails or once the output routine returns, the input routine checks if DDT Suck has left it a character (in DDTC.) If it has it returns it to DDT (having zeroed DDTC.) DDT and the DDT output routines are coroutines. The output routine is transparant to the A, B, and X registers. The saving/restoring is common to all of the output routine's entrys/returns. The output routine JAMs the header DDT SUCK has saved and then returns/calls DDT for characters which it JAMs (oring on parity) two at a time. It breaks output up into many messages for some commands and for some long-winded input messages (recall that the DDT input routine closes the DDT JAM message on encountering an input message end.) The DDT output routine performs an additional function: if a flag which TTY Suck sets is set, it resets DDT by altering the output routine's return address and returning (which allows the TTY to interrupt the local DDT.)

ENTRY POINTS

Coroutined with JAM/DOZE

EXTERNAL CALLS

None

INITIALIZATION

Done by TTY JAM

CLEANUP

None

DATA STRUCTURES

LOCAL DATA

DOTW	output word save
DOTA	saved A register
DOTB	saved B register
DOTX	saved X register
DCNT	output message word count

SHARED DATA

DINC	input character
DEND	end of input message flag

I/O PERFORMED

None

RUNNING TIME

DDT input routine: 27+dozes(15+DOZE)
(closing a message adds 51+JAM cycles)
DDT output routines:
38+2(JAM)+S/R+oddbytes(19+S/R)+evenbytes(20+JAM+S/R)
interrupt saves 7+JAM cycles of last odd character
Saving/Restoring(S/R): 39

SIZE (OCTAL)

DDT input routine) 24
DDT output routine) 35
interrupt) 4
S/R) 24
Initialization) 12

PROGRAM DESCRIPTION 4.7.2.3

NAME

Trace

FUNCTION

1) Initiate a message to the selected destination.
2) Find complete trace blocks on the active trace queue.
3) Copy them into the message.
4) Return the blocks to the free trace queue.

CONTROL STRUCTURE

Function 1 is performed, then functions 2,3, and 4 are performed in a loop until the end of the active trace queue is reached. Then the message is closed off and the program goes to sleep for one background loop. At the next background loop, if there is anything on the active trace queue, the cycle is repeated. Functions 2 and 4 run with interrupts locked, otherwise interrupts are enabled.

ENTRY POINTS

Called as a background coroutine once every background loop.

EXTERNAL CALLS

None.

INITIALIZATION

A dummy message is sent off at Initialization.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

T2BX	Copy loop counter.
T3BX	Counter for blocks copied.
OLD2	Queue pointer used in search.
OLD1	Packet pointer used in copy.

SHARED DATA

TTF, STRQ - Free and active trace blocks.
TTO - Trace overflow counter.

I/O PERFORMED

None.

RUNNING TIME

1) 20 cycles
2) 20 cycles per block
3) 200 cycles per block
4) 10 cycles per block
Total 250 cycles for one block message

SIZE (OCTAL)

1) 10 words
2) 30 words
3) 30 words
4) 10 words
Total 100 words

PROGRAM DESCRIPTION 4.7.2.4

NAME

Statistics

FUNCTION

1) Detect transitions in the cumulative statistics on/off indicator and update the statistics-gathering locations.
2) Check each active statistics program to see if it is time to call it. If it is, send out the message leader.
3) Call SNAP, the Snapshot Statistics program, if necessary.
4) Call SEST, the Cumulative Statistics program, if necessary.
5) Call GENM, the Message Generator program, if necessary.
6) Call TRBL, the NCC Trouble Report program, if necessary.
7) Call DIAG, the NCC Diagnostic Report program, if necessary.

CONTROL STRUCTURE

The five Statistics programs are multiplexed on a single fake host port. The Statistics slot is a coroutine in the background loop which performs function 1 and then performs the function 2 for each of the 4 programs, using shared code. Both the functions run with interrupts enabled. When it is time to run a given statistics program, then functions 3,4,5, and 6 may be performed. These programs run with interrupts enabled for the most part.

ENTRY POINTS

Called as a background coroutine once every background loop.

EXTERNAL CALLS

None.

INITIALIZATION

1) The saved copy of the Cumulative Statistics on/off flag is initialized to be off, and the statistics-gathering locations are initialized to their nominal contents.
2) All the statistics programs are initialized to be off.

CLEANUP
None.

DATA STRUCTURES

LOCAL DATA

1) SOFO - Saved copy of the Cumulative Statistics on/off flag.
SB1, SC1, SW1 - Tables for statistics-gathering locations, their nominal contents, and their contents when cumulative statistics are turned on.
2) OLDS - Table of the times that each statistics program was last run.

SHARED DATA

1) SON - Cumulative Statistics on/off flag.
2) SNON, SON, MGON, TPON, DIAGON - The on/off flags.
STATF, STATD, STATL- Tabled parameters for each statistics program, giving frequency, destination, and link.
3) TIME - Local time.
NHA, NHS - Host queue counters for each host. NFA, NFS, NSFA, NSFS, NREA, NRES, NALA, NALS - Storage utilization counters.
RUT, RST - Route Use Table and Route Send Table.
4) SYNC - Global time.
STTB - Table of statistics counters.
6) HIHD - Host alive/dead flags for each host.
SWS - Switch setting word.
RSFNCC - Restart/reload indicator.
HLTLOC, HLTA, HLTX - Saved PC,A,X from last halt.
NFA, NFS, NSFA, NSFS, NREA, NRES, NALA, NALS.
VERS - IMP program version number.
HOST34 - IMP confguration word.
TIPVER - TIP program version number.
HLNM, HLSNT, HLRCVD - Number of host being tested, number of test messages sent and received.
LINE, NEIGHB - Tables for each line, giving up/down status and neighbor imp number.
RTSSNT, E123, E321 - Tables of number of routing messages sent, received, and missed for each line.
THRUPT - Table of number of acks per line.
NTRTAB, HTPTBL- Table of pointers and a table of host throughput counters.
DIAGQ - queue of error packets waiting to be sent to the NCC.

I/O PERFORMED

None.

RUNNING TIME

1) 10 cycles (+200 cyles if transition)
2) 15 cycles per program (+200 cycles if program is run)
3) 7000 cycles
4) 7000 cycles
5) 25 cycles per word
6) 2500 cycles
7) 2000 cycles

SIZE (OCTAL)

1) 30 words
2) 100 words
3) 30 words
4) 20 words
5) 50 words
6) 130 words
7) 20 words

PROGRAM DESCRIPTION 4.7.3

NAME

Back Hosts

FUNCTION

The Back Hosts serve as the source of several important control messages.

1) BACK0 - Send Allocates, RFNMs, and RFNMs with Allocates.
2) BACK1 - Send Incomplete Transmissions.
3) BACK2 - Send Give Backs.
4) BACK3 - Send non-request single packet messages.
5) BACK4 - Send out-of-range replies, re-routed packets from dead lines, and destination dead messages.

CONTROL STRUCTURE

The Back Hosts are a series of coroutines called from the main Background loop. Each routine determines if a particular kind of message needs to be sent. If one does, it formats the appropriate message, gives it to TASK and waits for it to be accepted. If at any time processing cannot continue, the routine makes a coroutine return to the Background.

ENTRY POINTS

Coroutine entrance from Background.

EXTERNAL CALLS

GIVTSK - Call the TASK routine with a packet, generating its checksum, and wait for it to be accepted.
SLEEP - Return to the Background loop.

INITIALIZATION

The coroutine entries, tabled in SLTB, are initialized for each routine.

CLEANUP

1) No RFNM is delayed more than 1 second by waiting for a piggy-backed allocate.

DATA STRUCTURES

LOCAL DATA

1)
2) BACK1P - Pointer to message table. One entry is checked per call.
3) BACK2D - Destination IMP for Give Back.
4) BACK3P - Pointer to PPT. One entry is checked per call.

SHARED DATA

TSKFLG - A communication flag for each Back Host which indicates whether TASK accepted or rejected the last input.
1) RALLY - Table of pending RFNMs and Allocates.
NALA, NALS, NREA, NRES - Storage utilization counters.
2) TMESS - Transmit message number table.
3) TALLY - Transmit allocate table.
4) PPT - Pending packet table.
5) SRQ, ERQ - Reply, reroute, and destination dead queue.

I/O PERFORMED

None.

SIZE (OCTAL)

1) 200 words
2) 100 words
3) 30 words
4) 30 words
5) 20 words
Other 60 words
Total 500 words

PROGRAM DESCRIPTION 4.8

NAME

Very Distant Host (VDH)

FUNCTION

To implement the Very Distant Host interface described in Appendix F of BBN Report 1822. This includes handling the VDH modem interface, implementing the Reliable Transmission Package described in Appendix F, and efficiently interfacing the Reliable Transmission Package to IMP to Host and Host to IMP while at the same time simulating the Regular Host Interface as far as Host to IMP and IMP to Host are concerned.

NAME

VDH Initialization

FUNCTION

1) To initialize the entire VDH package when the system is restarted.
2) To reinitialize the VDH acknowledgement system when the VDH phone line has failed, when IMP to Host flaps the ready-line, or when the system is restarted.

CONTROL STRUCTURE

Both functions 1 and 2 are non-reentrant subroutines.

ENTRY POINTS

1) The entry point for function 1 is VDH.I which is called by IMP initialization thruogh ""VDH1."".
2) The entry point for function 2 is VD.REI which is called by function 1 and by VDH Timeout, by the latter when the VDH phone line goes dead or IMP to Host flaps the ready-line. VDH Timeout also calls VD.REI when a " spurious ack " is detected.

EXTERNAL CALLS

1) VD.I calls VD.REI to have the acknowledgement system reinitialized.
2) VD.REI calls flush to return unacknowledged buffers.

INITIALIZATION

1) The call to VD.I is initializated as part of the VDHDEF code in IMP initialization.
2) None.

CLEANUP

None.

DATA STRUCTURES

None specific to VDH Initialization.

LOCAL DATA

1) None.
2) VD.REX is used as a temp for the index register.

SHARED DATA

All tables or variables initialized by VDH Initialization are, of course, in some sense shared with the other VDH routines. Therefore, appropriate interlocks must be taken with calls to VD.REI.

I/O PERFORMED

None.

RUNNING TIME

1) Negligible.
2) 200 cycles plus a call to FLUSH.

SIZE (OCTAL)

1) 10 words.
2) 32 words.
Total 42 words.

PROGRAM DESCRIPTION 4.8.2

NAME

VDH Input Interrupt

FUNCTION

1) To process modem input interrupts from VDH phone line and to initiate new modem inputs for this same line.
2) To process acknowledgements.
3) To keep the phone line alive if acks are received.
4) To pass packets received from the phone line to VDH Background.
5) To mark for acknowledgements to be sent.
6) To detect duplicate packets received.

CONTROL STRUCTURE

VDH Input Interrupt is a non-reentrant subroutine called by the VDH modem input interrupt. Included is one local subroutine which is called to process acknowledgements.

ENTRY POINTS

The entry point to VDH Input Interrupt is VD.II which is called by the VDH MODEM INPUT INTERRUPT. VDH Input Interrupt runs on a priority below the other modem input interrupts but above the modem output interrupts -- that is, it runs at the same level as the other modem input interrupts until it gets an interrupt and then it enables the other modem input interrupts. VD.AP is the entry point to the local subroutine used to process acknowledgements.

EXTERNAL CALLS

If a spurious ack is received, HLTNCC is called to send a trap to the NCC and VD.REI is called to resync the acknowledge sequence. DODXA is also called after the registers are saved to enable interrupts.

INITIALIZATION

The call to VD.II is set up in IMP initialization by the VDHDEF code. All other initialization of VDH Input Interrupt is done by VDH Initialization.

CLEANUP

None other than that already mentioned for spurious acks.

DATA STRUCTURES

LOCAL DATA

The mask, keys, and registers are saved in VD.IM, VD.IK, VD.IA, and VD.IX. VD.IIB, VD.RBL, VD.CWP, and VD.RCN are used to save the input buffer pointer, the buffer length, a pointer to the input buffer control word, and the receive channel number.

SHARED DATA

VDH Input Interrupt accesses the FREE queue and NFS, accesses the counters VD.R and VD.T which are shared with other of the VDH routines, and accesses the tables VD.TOE, VD.TB, VD.RE, VD.ROE, and VD.RB which are shared with the other VDH routines.

I/O PERFORMED

Does VDH modem inputs.

RUNNING TIME

205 cycles plus a call to DODXA assuming the received packet is good and acceptable and carries one non-duplicate ack to process.

SIZE (OCTAL)

1), 4), and 5) 75 words
2) and 3) 37 words
6) 11 words
Save and restore mask, keys, and registers 25 words
Local storage 7 words
Total 201 words

PROGRAM DESCRIPTION 4.8.3

NAME

VDH Output Interrupt

FUNCTION

1) To process modem output interrupts from the VDH phone line and to initiate modem outputs for this line.
2) To build acks.
3) To retransmit unacknowledged packets.
4) To free buffers of acknowledged packets.
5) To pass packets from VDH Background to the phone line.

CONTROL STRUCTURE

VDH Output Interrupt is a non-reentrant subroutine called by the VDH modem output interrupt and by VDH Background, the latter when the output interrupt sequence must be restarted. ncluded are two local subroutines: one to build the acknowledge field in the control word and the other to service each output channel in turn.

ENTRY POINTS

The entry point to VDH Output Interrupt is VD.OI which is called by the VDH modem output interrupt and by funtction 3 of VDH BACKGROUND. VDH Output Interrupt has the same priority as the other modem output interrupts. VD.OIT is the entry point to the subroutine used to build the acknowledge field in the control word. VD.OIS is the entry point to the subroutine used to service each output channel in turn. The subroutine VD.OIS is peculiar in that when there is nothing to do for a particular channel, the subroutine returns, but when there is something to do for a given channel, then rather than returning, the subroutine jumps to VD.OI3.

EXTERNAL CALLS

VD.OI calls DODXA after the registers have been saved to enable interrupts. VD.OIS calls FLUSH to return acknlwledged buffers.

INITIALIZATION

The call to VD.OI is set up in IMP Initialization by the VDHDEF code. All other initialization of VDH Output Interrupt is done by VDH Initialization.

DATA STRUCTURES

LOCAL DATA

The mask, keys, and registers are saved in VD.OM, VD.OK, VD.OX, and VD.OA. VD.OB and VD.CW are used to save pointers to the output buffer and to build the control word field.

SHARED DATA

VDH Output Interrupt accesses the FREE queue via FLUSH, ACCESSES THE COUNTERS VD.R and VD.D shared with the other VDH routines, and accesses the tables (e.g., VD.TE and VD.TB) shared with the other VDH routines.

I/O PERFORMED

Does VDH modem outputs.

RUNNING TIME

150 cycles plus a call to DODXA plus a call to FLUSH assuming the that the is a buffer to transmit and one acknowledged buffer to release.

SIZE (OCTAL)

1), 3), 4), and 5) 64 words
2) 10 words
Save and restore mask, keys, and registers 27 words
Local storage 6 words
Total 131 words

PROGRAM DESCRIPTION 4.8.4

NAME

VDH Timeout

FUNCTION

To decrement VD.R, VD.T, and VD.D as appropriate, to test the IMP/Host ready-line, and to call VD.REI if the IMP/Host ready-line has been flapped or if the VDH phone line has gone dead.

CONTROL STRUCTURE

VDH Timeout is a non-reentrant subroutine.

ENTRY POINTS

The subroutine entry point is VD.TO which is called through VDH3. by Slow Timeout.

EXTERNAL CALLS

VD.REI is called when the VDH phone line goes dead or the IMP/Host ready-line is flapped.

INITIALIZATION

The call to VD.TO from Slow Timeout is initialized as part of the VDHDEF code in IMP initialization.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

None.

SHARED DATA

The counters VD.R, VD.T, and VD.D are shared with the other VDH routines. The flag VD.RDY is shared with IMP to Host.

I/O PERFORMED

None.

RUNNING TIME

20 cycles normally.
22 cycles plus a call to VD.REI in the worst case.

SIZE (OCTAL)

Total 36 words

PROGRAM DESCRIPTION 4.8.5

NAME

VDH Background

FUNCTION

1) To pass packets from VDH Input Interrupt to Host to IMP.
2) To pass packets from IMP to Host to VDH Output Interrupt.
3) To wake up VDH Output Interrupt if it has languished.

CONTROL STRUCTURE

VDH Background is a non-reentrant subroutine called from IMP Background. The subroutine is constructed of three essentially straight line section of code to handle each of the three functions. An attempt to run all three sections is made each time through VDH Background.

ENTRY POINTS

The entry point to VDH Background is VD.B which is called out of IMP Background via "VDH2."

EXTERNAL CALLS

1) FLUSH is called to release the buffer the leader came in and to release the buffer Host to IMP had up for input. An interrupt to Host to IMP is faked via a call through VD.HII when a buffer is ready for Host to IMP and then DODXA must, of course, be called.
2) GETFRE is called to get a free buffer to swap with the buffer IMP to Host has set up for output. After each buffer has been sent, an interrupt to IMP to Host is faked via VD.HOI and, of course, a call to DODXA is then also required.
3) If VDH Output Interrupt does not have an interrupt pending, an interrupt is faked via VD.OI which, of course, must be followed by a call to DODXA.

INITIALIZATION

The call to VD.B is initialized as part of the VDHDEF code in IMP Initialization. All other necessary initialization is done by VDH Initialization.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

The mask is saved in VD.BM -- VDH background runs at IMP to Host level. Function 2 uses VD.HOL 1 to determine whether to send a leader or not. The variables VD.IB, VD.BB, VD.BBT, and VD.BBF are used to swap buffers with the IMP to Host and Host to IMP routines.

SHARED DATA

VDH Background shares all of the various VDH tables with the other VDH routines.

I/O PERFORMED

VDH Background does no actual I/O but simulates the I/O hardware as far as IMP to Host, Host to IMP, and VDH Output Interrupt are concerned.

RUNNING TIME

3) 10 cycles plus a call to DODXA plus, if VDH Output Interrupt is to be woken up, a call to that.
Save and restore mask 20 cycles

SIZE (OCTAL)

1) 72 words
2) 102 words
3) 5 words
Save and restore mask 15 words
Local storage 5 words
Total 221 words

PROGRAM DESCRIPTION 4.9

NAME

Initialization

FUNCTION

Initialize all data structures, initialize and start up the rest of the IMP system.

CONTROL STRUCTURE

Sequential code entered and executed once only at startup time. The following is a list of initialization procedures in the order performed:

1) Set restart flag appropriately for trouble reports.
2) Initialize TIP (if necessary).
3) Clear the zero-storage area.
4) Set up MINE, OBIT, CBIT.
5) Set up nice-stop
6) Select and set IMP/TIP dependent locations.
7) Select and set IMP/VDH dependent locations.
8) Initialize queue structure.
9) Build free list.
10) Build reassembly and trace free lists.
11) Initialize message tables (TMESS, RMESS).
12) Set variable interrupt masks and entrances (HOST34 dependent).
13) Set background co-routine start addresses.
14) Initialize Timeout.
15) Fire off a trouble report.
16) Initialize fixed routing header.
17) Start up Modem to IMP.
18) Start up IMP to Host and Host to IMP.
19) Initialize VDH (if necessary).
20) Enable all interrupts.
21) Exit to background.

DATA STRUCTURES

LOCAL DATA

IT1 and IT2 are used for temporary storage.

SHARED DATA

See descriptions of programs being initialized.

I/O PERFORMED

IMP number and real-time clock are read; modem inputs on five channels are performed.

SIZE (OCTAL)

Approx. 1020

NAME

Power Failure Restart

FUNCTION

To provide automatic recovery in case of a power failure.

CONTROL STRUCTURE

Power failure is detected by an interrupt which cannot be masked off or inhibited. Upon receipt of this interrupt, the system prepares to restart from initialization when hardware detects return of power.

ENTRY POINTS

The hardware interrupt comes to RSTR.

EXTERNAL CALLS

Initialization is begun following return of power.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

FREE, NHA, and the index register are usurped to provide for the halt and restart.

SHARED DATA

RSFLAG - Used to signal that power failure occurred.

I/O PERFORMED

The Watchdog Timer is poked.

RUNNING TIME

18 cycles

SIZE (OCTAL)

14

PROGRAM DESCRIPTION 4.10.2

NAME

Watchdog Timer

FUNCTION

Provide a check on the running IMP system, and if the system should fail to run properly, cause a system reload.

CONTROL STRUCTURE

A properly running system pokes the Watchdog Timer more often than once every two minutes. Should two minutes of real time elapse without such a poke, the hardware generates an interrupt to the reload code. This interrupt cannot be masked out or inhibited. This feature operates in conjunction with the HALT-INHIBIT switch, which keeps a broken machine running so that the interrupt may occur. Currently the Watchdog Timer is poked when Timeout runs. (Applies only to DDP-516 IMPS.)

ENTRY POINTS

The hardware interrupt comes to WDTM.

EXTERNAL CALLS

Initialization is begun following a successful load.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

None.

SHARED DATA

RSFLAG - Used to signal that reload has occurred.

I/O PERFORMED

Modem inputs and outputs to request and receive core images; Real-time clock input to select random line.

SIZE (OCTAL)

Approx. 210

5. Logic Diagrams

The logic diagrams for the IMP program are not being updated and are therefore not included with this volume. Sets of the original diagrams valid as of February 1973 but now obsolete are available on request while a supply lasts. Address:

Computer Systems Division
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

Attn: Robert Brooks

6. Data Formats

```
IMP PACKET FORMAT
                     --------------------
          /          !                  !  CHAIN PTR
          !          --------------------
          !          !                  !  IT OR ST: INPUT TIME OR SENT TIME
          +          --------------------
          !          !                  !  PTRT: TRACE PTR
          !          --------------------
          \   /--!                  !
              !      --------------------
              !      --------------------
          /   !      !                  !  ACKH
          !   !      --------------------
          !   !      !                  !  HEAD
          !   !      --------------------
HEADER----+   !      !                  !  HEAD1
          !   !      --------------------
          !   !      !                  !  HEAD2 OR CNTL
          !   !      --------------------
          \   !      !                  !  HEAD3
              !      --------------------
              !      --------------------
63 WORDS--!   !      !                  !  DATA WORDS
              !      --------------------
              !      --------------------
          !   !      !                  !  BUFE: PTR TO LAST USED DATA WORD
              !      --------------------
              !
              !      !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
              \--!  !XXXXXXXXXXXXXXXXXXXXXX!          !
                     !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
                       !                            \-----/
                       !                               \--INCH: INPUT CHANNEL
                       \--1 IF FROM HOST, Ø IF FROM MODEM
```

IMP HEADER FORMAT FOR NET TRAFFIC

```
                     --------------------
          /       /--|                  |
          |       |  --------------------
          |     /-+--|                  |
          |     | |  --------------------
HEADER----+   /-+-+--|                  |
          |   | | |  --------------------
          | /-+-+-+--|                  |
          | | | | |  --------------------
          \ | | | |  |        |XXXXXXXXX|  LINK NUMBER
            | | | |  --------------------
            | | | |
            | | | |  |-|-,-,-|-,-,-|-,-,-|-,-,-|-,-,-|
            | | | \--| |   |         |               |
            | | |    |-|-,-,-|-,-,-|-,-,-|-,-,-|-,-,-|
            | | |     | \-/ \-------/ \-------------/
            | | |     |  |      |            \--ACKNOWLEDGE BITS
            | | |     |  |      \--CHANNEL NUMBER
            | | |     |  \--WHICH QUADRANT ACKS REFER TO
            | | |     \--ODD-EVEN BIT
            | | |
            | | |    |-|-,-,-|-,-,-|-,-,-|-,-,-|-,-,-|
            | | \----|               | | |   | | | | |
            | |      |-|-,-,-|-,-,-|-,-,-|-,-,-|-,-,-|
            | |       \-------------/ | | \-/ | | | |
            | |               |       | |  |  | | | \--Ø(NON-ROUTING)
            | |               |       | |  |  | | \--GIVE BACK (T) OR
            | |               |       | |  |  | |       ALLOCATE (R)
            | |               |       | |  |  | \--1=TRANS, Ø=REPLY
            | |               |       | |  |  \--INCOMPLETE (T OR R)
            | |               |       | |  \--ORDER NUMBER
            | |               |       | \--REQUEST FOR ALLOCATE (T) OR
            | |               |       |      RENM (R)
            | |               |       \--1=ONE-PACKET
            | |               \--MESSAGE NUMBER
            | |
            | |      |-|-,-,-|-,-,-|-,-,-|-,-,-|-,-,-|
            | \------| | | | |X|     |   |           |
            |        |-|-,-,-|-,-,-|-,-,-|-,-,-|-,-,-|
            |         | | | |   \---/ \-/ \---------/
            |         | | | |     |    |       \--DESTINATION IMP
            |         | | | |     |    \--DESTINATION HOST
            |         | | | |     \--PACKET NUMBER (T) OR HOST DEAD (R)
            |         | | | \--FOR OCTAL
            |         | | \--TRACE
            |         | \--FOR IMP
            |         \--PRIORITY
            |
            |        |-|-,-,-|-,-,-|-,-,-|-,-,-|-,-,-|
            \--------| | |X| |XXX| |X|   |           |
                     |-|-,-,-|-,-,-|-,-,-|-,-,-|-,-,-|
                      | |   |     |   \-/ \---------/
                      | |   |     |    |       \--SOURCE IMP
                      | |   |     |    \--SOURCE HOST
                      | |   |     \--FROM HI NUMBERED GUY ON LINE
                      | |   \--FROM OCTAL
                      | \--FROM IMP
                      \--LAST PACKET
```

```
IMP ROUTING MESSAGE FORMAT - RST TABLE
                 --------------------
          /      !XXXXXXXXXX!       !  ACKNOWLEDGE BITS
          !      --------------------
          +   /--!                  !
          !   !  --------------------
          \   !  !                  !  "SYNC" TIME
              !  --------------------
              !  --------------------
NIMP------! /-+--!                  !
            ! !  --------------------
            ! !  --------------------
          ! ! !  !                  !  CHECKSUM
            ! !  --------------------
            ! !
            ! !  !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            ! \--! !X!           !XXXXX! ! !X! ! !
            !    !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            !     !   \---------/       ! !   ! !
            !     !        !            ! !   ! \--1(ROUTING MESSAGE)
            !     !        !            ! !   \--NULL PACKET OF ACKS
            !     !        !            ! !         (NO RUT TABLE)
            !     !        !            ! \--CORE DUMP (NO RUT TABLE)
            !     !        !            \--I HEARD YOU
            !     !        \--SOURCE IMP
            !     \--SEND CORE (NO RUT TABLE)
            !
            !    !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            \----!         !                     !
                 !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
                  \-------/ \-------------------/
                      !               \--DELAY
                      \--HOP COUNT
```

```
TRACE BLOCK FORMAT
             --------------------
          /  !                  !  CHAIN PTR
          !  --------------------
          !  !                  !  TIT: INPUT TIME
          !  --------------------
          !  !                  !  TTT: TASK TIME
          !  --------------------
          !  !                  !  TST: SENT TIME
          !  --------------------
          !  !                  !  TAT: ACK TIME
          !  --------------------
          +  !                  !  THED: HEADER
          !  --------------------
          !  !                  !     "
          !  --------------------
          !  !                  !     "
          !  --------------------
          !  !                  !     "
          !  --------------------
          !  !                  !  TQUE: OUTPUT CHANNEL
          !  --------------------
          \  ! !XXXXXXXXXXXXXXXX!  TDONE: 1 IF DONE
             --------------------

REASSEMBLY BLOCK FORMAT
               --------------------
          /    !                  !  CHAIN PTR
          !    --------------------
          +  /--!                 !
          !  ! --------------------
          \  ! !                  !  RMAX: NO OF PACKETS IN MESSAGE
             ! --------------------
             ! --------------------
8---------!  ! !                  !  PACKET PTRS
             ! --------------------
             ! --------------------
          !  ! !                  !  RSF: PACKETS IN SO FAR
             ! --------------------
             !
             !
             ! !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
             \--!                 !XXX!          !
               !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
                \-------------/     \---------/
                      !                  \--SOURCE IMP
                      \--MESSAGE NUMBER
```

```
TRANSMIT MESSAGE TABLE
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
NIMP------!  !               !   !   !       !
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
              \-------------/ \-/ \-/ \-----/
                     !         !   !     \--BIT=1 MEANS THAT MESSAGE
                     !         !   !           ANSWERED
                     !         !   \--LAST ORDER NUMBER TRANSMITTED
                     !         \--TIMEOUT BITS
                     \--LAST MESSAGE NUMBER TRANSMITTED


RECEIVE MESSAGE TABLE
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
NIMP------!  !               !XXX!   !       !
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
              \-------------/     \-/ \-----/
                     !             !     \--BIT=1 MEANS THAT MESSAGE
                     !             !           COMPLETE
                     !             \--NEXT ORDER NUMBER EXPECTED
                     \--OLDEST INCOMPLETE MESSAGE NUMBER + 3


PENDING PACKET TABLE - PARALLEL TABLE 2 WORDS WIDE
               --------------------
          / /--!                  !
6---------+ !  --------------------
          \ !  !                  !  HOST STAT TIME SENT
            !  --------------------
            !
            !  !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            \--! !X!                           !
               !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
                !   \-------------------------/
                !            \--POINTER TO PACKET
                \--GOT ALLOCATE, MUST RETRANSMIT

TALLY
TABLE OF IMPS WITH SPACE ALLOCATED
             --------------------
8---------!  !XXXXXXXXXXX!      !  NUMBER OF IMP WHO ALLOCATED 8 FOR US
             --------------------
```

```
PENDING LEADER TABLE - PARALLEL TABLE 4 WORDS WIDE
                   --------------------
           /    /--!                  !
           !    !  --------------------
           !  /-+--!                  !
16,--------+  ! !  --------------------
           !/-+-+--!                  !
           !! ! !  --------------------
           \! ! !  !                  !   HOST STAT TIME SENT
            ! ! !  --------------------
            ! ! !
            ! ! !  !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            ! ! \--!                ! ! !XXXXXX!    !
            ! !    !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            ! !     \-------------/ ! !       \---/
            ! !            !        ! !         \--LOCAL HOST NUMBER
            ! !            !        ! \--REQUEST (USEFUL FOR INC. TRN.)
            ! !            !        \--THIS ENTRY IS IN USE
            ! !            \--MESSAGE NUMBER
            ! !
            ! !    !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            ! \----! ! ! ! !XXXXXXXX!   !           !
            !      !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            !       ! ! ! !         \-/ \----------/
            !       ! ! ! !          !      \--DEST IMP
            !       ! ! ! !          \--DEST HOST
            !       ! ! ! \--OCTAL PRINT
            !       ! ! \--TRACE
            !       ! \--FOR IMP
            !       \--PRIORITY
            !
            !      !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
            \------!                !XXXXXXXXXX!    !
                   !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
                    \-------------/         \---/
                           !                  \--SUBCODES
                           \--LINK NUMBER
```

```
AMESS
PARALLEL WITH RALLY
                   ---------------------
NIMP-------!  !         !XXXXXXXXXX!  NEXT MESSAGE NUMBER TO REPLY TO
                   ---------------------


RALLY
PENDING ALLOCATES/RFNMS TO GO
THIS TABLE IS PARALLEL WITH AMESS
EACH WORD IS 4 FOUR-BIT BYTES, USE BYTE N FOR N=AMESS MOD 4:
              !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
NIMP-------!  !       !       !       !       !
              !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
               \-----/ \-----/ \-----/ \-----/
                  !       !       !      \--BYTE 0
                  !       !       \--BYTE 1
                  !       \--BYTE 2
                  \--BYTE 3


FORMAT OF A RALLY BYTE IS AS SHOWN, TT AND MM CODED AS FOLLOWS:
TT:   00=ENTRY NOT IN USE.
      01=RFNM OR RFNM/ALLOCATE
      10=INCOMPLETE TRANSMISSION
      11=DESTINATION DEAD
MM:   00=8 PACKET ALLOCATE
      01=8 PACKET RFNM/ALLOCATE
      10=1 PACKET ALLOCATE
      11=1 PACKET RFNM

THE FORMAT OF ONE BYTE IS SHOWN:
              !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
           !  !   !   !XXXXXXXXXXXXXXXXXXXXXXXX!
              !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
               \-/ \-/
                !    \--MM
                \--TT
```

```
RMFLG
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
CH--------!  ! !             !             ! !
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
              ! \-----------/ \-----------/ !
              !       !             !       \--ROUTING MSG PENDING
              !       !             \--SPEED HOLD-DOWN COUNTER
              !       \--0=5K,1=10K,2=50K,3=250K NOMINAL LINE SPD
              \--SET IN INIT TO SUPPRESS TRAP ON LOAD


ROUTE SEND TABLE (RST) AND NEW RST (NRST)
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
NIMP------!  !         !                     !
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
              \-------/ \-------------------/
                  !               \--DELAY
                  \--HOP COUNT


ROUTE USE TABLE, BEST LINE DIRECTORY
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
NIMP------!  ! !   !         !     !         !
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
              ! \-/ \-------/ \---/ \-------/
              !  !      !       !       \--LINE # +1 OF SHORTEST DELAY
              !  !      !       \--COMING UP DELAY COUNTER
              !  !      \--LINE # +1 OF SHORTEST HOP PATH
              !  \--GOING DOWN DELAY COUNTER
              \--IMP DOWN,UNREACHABLE OR ISOLATED


RUTW - RUT WORKING TABLE
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
NIMP------!  ! !   !           !   !         !
             !-!-,-,-!-,-,-!-,-,-!-,-,-!-,-,-!
              ! \-/ \---------/ \-/ \-------/
              !  !       !       !      \--0
              !  !       !       \--HOLD-DOWN TIMER FOR MIN DELAY PATH
              !  !       \--0
              !  \--HOLD-DOWN TIMER FOR MINIMUM HOP PATH
              \--0
```

BIBLIOGRAPHY

1. Heart, F.E., Kahn, R.E., Ornstein, S.M., Crowther, W.R., and Walden, D.C. The interface message processor for the ARPA computer network. *Proceedings of AFIPS 1970 Spring Joint Computer Conference*, Vol. 36, pp. 551-567.

2. Kahn, R.E., and Crowther, W.R. A Study of the ARPA Network Design andPerformance. Report No. 2161, Bolt Beranek and Newman Inc., August 1971.

3. Kahn, R.E., and Crowther, W.R. Flow control in a resource sharing computer network. *Proceedings of the Second ACM IEEE Symposium on Problems in the Optimization of Data Communications Systems*, Palo Alto, California (October 1971), pp. 108-116.

4. Bolt Beranek and Newman Inc. Specification for the Interconnection of a Host and an IMP. Report No. 1822, revised , 1973.

5. MCQuillan, J.M., Crowther, W.R., Cosell, B.P., Walden, D.C., Heart, F.E. Improvements in the design and performance of the ARPA Network. *Proceedings of AFIPS 1972 Fall Joint Computer Conference*, Vol. 41, pp. 741-754.

6. McQuillan, J.B., Throughput study, BBN Report # .

ATTACHMENTS

I.	Concordance	IMPCON,3071,IMP
II.	Listing	IMPLST,3071,IMP