INTERFACE MESSAGE PROCESSORS FOR THE ARPA COMPUTER NETWORK

Frank E. Heart

Bolt Beranek and Newman, Incorporated

Prepared for:

Advanced Research Projects Agency

January 1974

UNCLASSIFIED

Security Classification

AD773417

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Mass. 02138

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
QUARTERLY TECHNICAL REPORT NO. 4, INTERFACE MESSAGE PROCESSORS

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)
1 October 1973 to 31 December 1973

5. AUTHOR(S) (First name, middle initial, last name)
Bolt Beranek and Newman Inc.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
January 1974	36 39	10

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F08606-73-C-0027	
b. PROJECT NO. 2351	Report No. 2717
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Advanced Research Projects Agency Arlington, Virginia 22209

13. ABSTRACT

The ARPA computer network provides a communication medium which allows dissimilar computers (Hosts) to interchange information. Each Host is connected to an Interface Message Processor (IMP), and IMPs are interconnected by leased common carrier circuits. There is frequently no direct circuit between two communicating Hosts, and the intermediate IMPs store and forward the information. IMPs regularly exchange information which is used to adapt routing to changing network conditions. IMPs also report a variety of parameters to a Network Control Center, which coordinates diagnosis and repair of malfunctions. The Terminal IMP (TIP) permits the direct attachment of 63 character-oriented terminals. The Satellite IMP (SIMP) will allow multi-station use of a single earth satellite channel. A High Speed Modular IMP (HSMIMP) is under development; one goal of this effort is to increase IMP performance by an order of magnitude. Specialized mini-Hosts under development will provide for: connection of remote batch terminals; simulation of a leased point-to-point circuit; encrypted Host communication.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computers and Communication						
Store and Forward Communication						
ARPA Computer Network						
Interface Message Processor						
IMP						
Terminal IMP						
TIP						
Satellite IMP						
SIMP						
Honeywell DDP-516						
Honeywell H-316						
Multi-Line Controller						
MLC						
Network Control Center						
NCC						
Host Protocol						
High Speed Modular IMP						
HSMIMP						
Lockheed SUE						
RJE mini-Host						
Private Line Interface						
PLI						

Report No. 2717

Bolt Beranek and Newman Inc.

INTERFACE MESSAGE PROCESSORS FOR THE ARPA COMPUTER NETWORK

QUARTERLY TECHNICAL REPORT NO. 4
1 October 1973 to 31 December 1973

Submitted to:

IMP Program Manager
Range Measurements Lab.
Building 981
Patrick Air Force Base
Cocoa Beach, Florida 32925

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Range Measurements Laboratory under Contract No. F0806-73-C-0027.

TABLE OF CONTENTS

1. OVERVIEW

This Quarterly Technical Report, Number 4, describes aspects of our work on the ARPA Computer Network under Contract No. F08606-73-C-0027 during the fourth quarter of 1973. (Work performed from 1969 through 1972 under Contract No. DAHC-15-69-C-0179 has been reported in another series of Quarterly Technical Reports numbered 1-16.)

During the quarter we delivered one 316 IMP to the Naval Air Station at Moffett Field (California) and three TIPs to Wright Patterson Air Force Base (Ohio), Rutgers University (New Jersey), and the Network Control Center at BBN. In conjunction with the installation of the new TIP at BBN, the 516 IMP at BBN was moved from the NCC area to the TENEX area. In addition, the IMP which was shipped to MIT's Information Processing Center during the third quarter was connected into the network. The Spare TIP was sent to replace the TIP at NOAA/Department of Commerce (Colorado) because of hardware failure in their machine, which was later returned to Honeywell for repair or replacement.

During this quarter we began field retrofits of the IMP's Real Time Clock. (The new clocks have been installed in all new machines starting with the Norway TIP in early June.) The new Real Time Clock incorporates a seven-word configuration card in place of the 14-bit IMP number card. This expansion will enable the software to adapt more easily to a greater number of hardware configurations.

By the end of the fourth quarter there were four Very Distant Hosts in the network. During this last quarter, the VDH program has been changed to ease the introduction of a new VDH site. The program now determines the Host-modem pair used

by the Very Distant Host at run time (at this time only one VDH is allowed on an IMP), rather than requiring a specially tailored system for each site.

The IMP program has continued to undergo changes through this quarter. Many of these changes are associated with efforts to improve the reliability and efficiency of packet routing in the network. The changes which have occurred during the past half-year are summarized in Section 2. During our investigation of routing techniques, Dr. Price from the NPL spent a day visiting BBN, the main topic of discussion being isarithmic networks. These discussions indicated that isarithmic networks have advantages over the ARPA Network flow control only at times of network overload.

A recent lockup of reassembly storage in the network has caused us to rethink the allocation mechanism for reassembly blocks. A summary of the problem and a possible solution are presented in Section 3.

A survey of the High Speed Modular IMP (HSMIMP) and its current status is given in Section 4.

Work on the Satellite IMP has continued on several fronts during this quarter. Our interactions with COMSAT have continued in an effort to resolve issues of system design for connecting Satellite IMPs to a broadcast satellite channel. COMSAT has been particularly concerned with some of the political issues which must be resolved before such a connection can be realized. This has resulted in COMSAT obtaining INTELSAT approval of a tariff for broadcast operation. Some progress has also been made in understanding the modifications which must be made to the SPADE system channel unit. A survey of the Satellite IMP program and system is included in Section 5.

The Network Control Center machine has begun expansion in several directions. The machine itself (currently identical to a 316 IMP with no modem) is being upgraded to a TIP to provide better backup with other machines in the NCC. The NCC program has been modified so that privileged users on BBN-TENEX or the PDP-1 can examine the contents of memory locations. This will eventually ease putting network status information on-line, and will permit these Hosts to verify the memory of the NCC.

In an effort to improve the reliability of the network from a TIP user's point of view, a project will be completed in early January which will test the dial-up modems in use on network TIPs. When this system is in operation, a program on BBN-TENEX will (at an interval of perhaps one hour) call each dial-up modem in the network. If the connection is successful, the program will then send the hunt character, an ASCII "E", and examine the response. An incorrect response will be noted, and will cause the NCC operators to be notified if it persists.

Finally, during this quarter BBN has continued its discussions of the Private Line Interface design with ARPA and NSA.

2. CHANGES TO ROUTING

The last half of this year has seen a number of changes to the routing program in the IMPs. These modifications can be grouped under the general heading of improvements in the reliability, the flexibility, and the efficiency of the propagation of routing information. After discussion with ARPA and other interested parties, it was decided to implement these modifications before any steps were taken toward high bandwidth routing, satellite routing, or area routing; however, we have made an effort to take our plans in those areas into account in designing the new routing propagation techniques.

Specific modifications made to date include the following.

2.1 Efficiency

The goal here is to make the propagation of routing data as rapid as possible.

- The process of changing the best route to an IMP is delayed for a few seconds. This time allows the adjacent IMPs to learn of the change, adapt to it, and therefore not conflict with the changeover. By simulation and some measurement of the network, we see that this technique speeds up the propagation of new information appreciably, by 1/3 to 1/2.
- The routing computation has been made incremental and input-driven, so that the routing tables are updated within a few milliseconds of the arrival of new routing data, rather than hundreds of milliseconds later on a timeout. Therefore, the routing information

in IMPs many hops away is more up to date by several seconds.

- The routing tables have been made smaller and faster to manipulate. There is no longer a routing table per adjacent IMP, eliminating a costly multiplicative factor in storage costs. The routing tables are triple-buffered, one for input, one for output, and one idle. This allows rapid table updates without copying, and without the need to synchronize with output. Again, the propagation of routing is speeded up by providing the output process with the most up-to-date table possible.
- The output of routing messages is triggered by the arrival of new routing data, rather than being strictly synchronous. This completes the system for rapid propagation of routing changes.

2.2 Flexibility

The goal here is to extend the operation of the routing program to different line speeds.

- The IMP program measures the bandwidth of the circuits to which it is connected, and the excess capacity available on each circuit.
- Routing messages are sent at a rate proportional to the bandwidth of a circuit.
- Routing messages are also sent more often when a circuit is lightly loaded, to improve the speed with which alternative paths can be recognized. The overhead due

to routing messages therefore ranges between about 3% and 15%, depending on loading, for any line speed in the range 5 Kbs-250 Kbs.

- At the same time, the criteria for declaring circuits usable have been made variable with circuit bandwidth, allowing more errors on slower lines before declaring them unusable.

2.3 Reliability

The goal here is to localize and minimize the effects of any hardware or software failure in the routing process.

- Routing messages carry a checksum generated in software at the time the message is built.
- The checksum is verified at the time the message is sent to catch intra-IMP failures such as processor or memory errors, software bugs which change the data, and so on.
- The checksum is verified at the time the message is received to catch inter-IMP failures such as bus transfer or modem interface errors, failures in the memory at the receiver, and so on.
- Errors in both categories have been detected. When an error is detected, the routing message is discarded, preventing the spread of any erroneous routing data through the network, and a copy is sent to the NCC for diagnosis.

- The routing directory held at each IMP also carries a checksum, which is incrementally modified whenever an entry changes, and periodically checked. Errors have been detected here also, and result in a program reload.
- In addition to checksums on routing data, the IMP maintains checksums on all routing programs. The input routing processing code is checksummed before it is executed; the same is true of the output routing processing code and the periodic routing processing code. Errors in the program code have been detected by this means, and result in immediate reload of the program before the erroneous program is executed even once.
- The goal of these measures has, in large part, been achieved, since several failures have been detected and corrected while local, preventing network-wide repercussions.

In summary, the first phase of the modifications to the routing program is largely finished. Some refinements and extensions to the techniques described may be implemented in the future, but the main thrust of our work will now be in the remaining areas of routing development named above--high bandwidth routing, satellite routing, and area routing.

3. A REASSEMBLY LOCKUP

Near the end of the fourth quarter, a new kind of reassembly lockup* occurred around the UCLA IMP, precipitating a new look at the way the IMP system handles reassembly. The NMC had turned on cumulative statistics and snapshots statistics all around the network. The lockup happened when a message arrived at UCLA, which had reassembly packet buffers allocated for it, but no reassembly blocks were free. A reassembly block is a piece of storage used in the actual process of putting the packets back together. Both a reassembly block and up to 8 packet buffers are needed to reassemble a message. The IMP reserves the packet buffers in advance, and assumes that there is always a free reassembly block available The lockup pointed out an oversight in the IMP program: not enough reassembly blocks had been assigned to cover all situations.

The lockup lasted for a few hours, while the NCC staff located the problem, and normal operations resumed when they contacted the NMC and requested that the NMC stop the experiment. Subsequently, the IMP program has been changed to allow the NMC to continue its experiments without any possibility of network lockup. Having experienced this problem and developed a solution, we reexamined the reassembly processing in the IMP.

*See QTR number 13 in the previous QTR series for a discussion of a type of reassembly lockup which was discovered and fixed.

How many reassembly blocks are needed? The worst-case analysis is as follows, given reassembly storage for N packets.

1) A large number of requests for allocation arrive at once.
2) N/8 allocates are sent, each good for an 8-packet message.
3) These messages turn out to be 2-packet messages; the second packets arrive before the first; the remaining 6 packet buffers for each message are declared free for other uses.
4) (N - 2* N/8) /8 = 3N/32 allocates are sent.
5) Same as 3.
6) (N - 2* 3N/32) /8 = 13N/128 allocates are sent.

And so on; eventually, all storage is allocated and N/2 reassembly blocks are in use.

3.1 A Simple-Minded Solution

Since the core retrofit (described in QTR Number 1, April 1973) has given us approximately 9 x 8 = 72 packet buffers for reassembly, there could conceivably be 36 messages being reassembled at once. One way to eliminate lockups would be to assign 36 reassembly blocks as part of the IMP's permanent storage. But each block is currently 12 words; this requirement of 36 x 12 = 432 words is excessive, considering that it is rare to be reassembling more than 2 or 3 messages at once. The whole network went for a year before an IMP had to reassemble more than 8 messages at once.

3.2 The Current Solution

In response to the lockup, we changed the procedure at the destination IMP to look like this:

1) Get request for allocation.
2) Wait till storage is free for 8 packets.
2a) Wait till reassembly block count shows a free block.*
3) Send allocate.
3a) Increment reassembly block count.*
4) Receive some packet of the message (not necessarily the first).
Find a free reassembly block (should always be possible); the format is:
- chain pointer
- message number
- source IMP number
- number of packets in this message
- number of packets received so far
- 8 packet pointers

Take the block from the free list and put it on the active list.
Set up the message number, IMP number.
5) For each packet received:
Set up the packet pointer (duplicate packets detected here).
Increment the received count.
6) When the last packet is received:
Set up the number of packets in this message.
7) When all packets have been received:
Wait till message number is next in sequence, then chain packets together, put on Host queue, and free reassembly block.
7a) Decrement reassembly block count.*

*These are the program changes.

3.3 Reassembly Reexamined

A natural direction for change in the future is to tie the reassembly block to the allocation of packet storage. The remainder of this section will discuss a possible change along this line.

At the time the allocation is sent (step 3 above), the IMP can wait until it can take a reassembly block. Then it can add the block to the active list, mark the IMP number and note the number of packets received as -1, to denote an empty block. Not only does this ensure that a reassembly block exists for every message, simplifying the first-packet processing, but it has an important additional benefit. *This gives us a data structure to record allocates that have been sent and are unused.* No such structure exists currently, so allocates sent off to an IMP which dies or malfunctions are not explicitly cleaned up. (There is an idle timer which causes all outstanding allocates to be voided after 2 minutes of inactivity, but this mechanism is not very satisfactory.)

With this scheme, when a packet of a multipacket message arrives, the program will search the reassembly queue for that message number and IMP number. If a match exists, the message is already being reassembled. If not, a second search is made for a block for that IMP which is empty (number of packets received = -1). There should always be a block of one type or the other.

Additionally, when an IMP goes down, all the other IMPs in the net can clean up their reassembly queues in a uniform fashion, freeing the reassembly space and packets from blocks in use, the allocation from empty blocks, and the blocks themselves in either case. Also, the processing of give-backs can

be made a little tighter, since a reassembly block should be set up for every give-back received.

3.4 An Extension with Message Numbers by Host

We are planning to change the IMP to assign message numbers on a per Host basis, and have been planning to use one message number for both the 8-packet request and the message. This is possible since on a per Host basis, no other traffic can flow between the request and the message. When we do this, the process which acquires the reassembly block before sending off the allocate can, in addition to setting up the IMP number, set up the message number of the reassembly block. This eliminates the need for a double search of the reassembly queue for the first packet. The search is still necessary in the case that the message is using a piggybacked allocate rather than an explicitly requested one.

3.5 An Extension with Message Length Given by Host

We are also planning to add a message length to the Host-IMP leader, to allow the Host to tell the IMP how long its message is. This information can be used to request an allocation of exactly as many packets as are necessary. Then when the reassembly block is acquired prior to sending the allocate, the exact size of the allocation can be saved in the "number of packets in this message" field. Again, piggybacked allocates are different, and will always refer to 8 buffers, but an improvement is possible here also. Currently, each packet has 3 bits for packet number and 1 bit for last packet or not. When we know message size ahead of time, the packet header can contain 3 bits of packet number and 3 bits of "total packets in this message". Then when any packet

of a message is received, the surplus between the number of packets allocated and the number of packets in the message can be freed.

3.6 Summary

The reassembly processing would then look like this (compare with section 3.2):

1) Get request for allocation of N packets (or have RFNM on which to piggyback an allocation of N = 8).

2) Wait for N packet buffers to be free; claim them.

3) Wait for a reassembly block to be free; when it is:
   - take it off the free reassembly block list
   - set up the IMP number, the Host numbers
   - set up the message number (if not a piggyback allocation)
   - set the total packet count to N
   - set the packets so far to -1
   - put the block on the active reassembly block list.

4) When a packet arrives:
   - search the active reassembly block list for that IMP/Host/message number.

   If found:
   - set up packet pointer (discard if duplicate)
   - set up total packet count according to packet header, discard any difference between this and previous contents as free space
   - increment count of received packets
   - done if received packet count = total packet count.

If not found:

- search for an empty block for this IMP
- set up message number and do above processing.

Note that we have eliminated any special processing for the last packet, and almost all special processing for the first packet. As a final thought, even this processing could be avoided. We plan to have at least 2 bits per message indicating

- request processed
- message processed

and we could define a state meaning "message being processed -- at least one packet received" which would differentiate the case of empty and non-empty reassembly blocks:

	Request Legal?	Message Legal?
00 - nothing received for this message	Y	Y (piggyback allocate)
01 - request received	N	Y
10 - message being processed	N	Y
11 - message completely processed	N	N

For single packet messages, state 10 is meaningless. We might want to do something similar to this approach for single packet messages, say claim a packet buffer and put the allocate in it, in order to keep track of outstanding 1-packet allocates. Then we could garbage-collect them in the event that the source IMP goes down.

4. HIGH SPEED MODULAR IMP

Descriptions in previous Quarterly Technical Reports of this series have concentrated on recent developments. In this section, however, we will survey the system as we now see it. The last such survey can be found in Number 14 of the previous series of Quarterly Technical Reports mentioned in the Overview. Another recent description which provides more insight into the software is found in [1].

The High Speed Modular IMP (HSMIMP) is designed to provide an order of magnitude increase over the throughput of the DDP-516 IMP. The architecture is highly modular and allows for IMPs of greater or lesser processing power than the present 516/316-based IMPs, as well as for many more and more varied phone line and Host interfaces. The architecture also provides for highly reliable configurations. The hardware consists of busses joined together by special bus couplers of our design. Figure 4-1 shows the structure of the prototype system. There are processor busses each of which contains two processors, each in turn with its own "private" 4K memory to store frequently run code. The more processor busses, the greater the system processing power. There are memory busses to house the segments of multi orted "common" memory -- the more memory busses, the more memory ports. Finally, there are I/O busses which house device and line controllers as well as a special (priority ordered) task disburser (the PID) which replaces the traditional priority interrupt system. The latter allows equality among the processors so that if some fail the rest can continue to run all system tasks, albeit at reduced capacity.

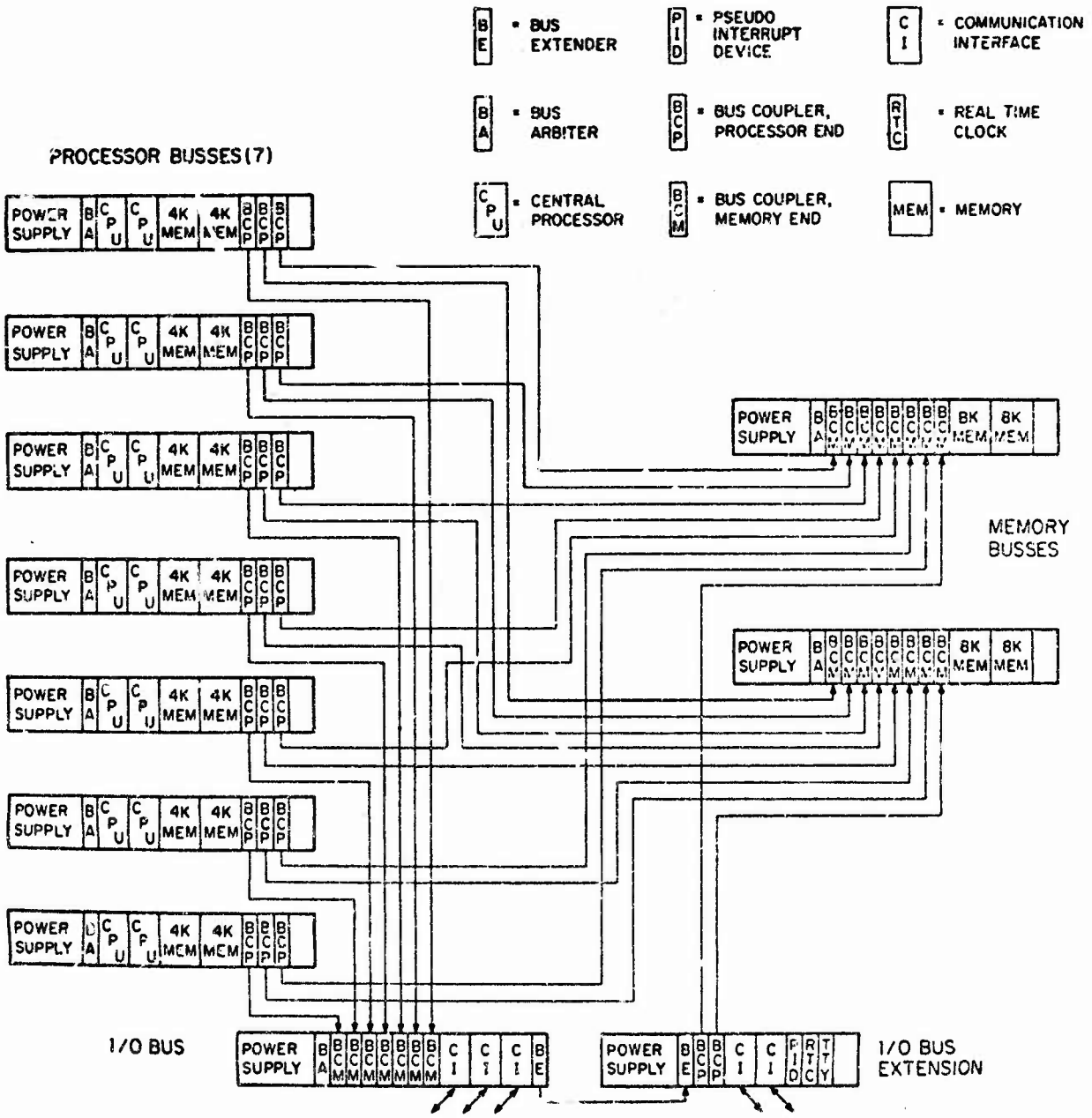

Figure 4-1 Prototype System

4.1 Design Issues

In this section we describe features we have designed into the system, some of the more interesting of which relate to reliability.

4.1.1 Addressing and Locking

The Lockheed SUE, with a 15-bit address, can directly address only up to 32K words. 8K of a processor's address space are used for direct references to its private memory. (Although we expect to use only 4K, 8K has been set aside to allow for growth.) Another 8K is used principally for addressing system I/O (on the up to four I/O busses). We assign 8 addresses to each I/O device for pointers and status and control registers; 960 devices can be accommodated in all.

16K of each processor's address space is mapped through the bus couplers to common memory. At the processor end of each coupler (BCP) are four program-settable map registers for each possible processor on the bus. (We expect to use only two processors per bus but up to four are permitted.) These map registers expand a 15-bit address to a 19-bit system address on the memory busses. By use of the maps, each processor can thus access, at any one time, four 4K pages in system address space. Read accesses through a particular one of these windows are turned by the coupler into read-clear operations, thereby providing the indivisible test-and-modify operation required for program interlocking in a multi-processor. (The processor itself presently lacks such an instruction.)

The coupler paths that connect processor busses into memory and I/O busses have program settable enabling switches at their

far (memory and I/O) ends, thus permitting processors to be cut in and out of the system. To allow processors to access one another and to permit reloading as discussed below, we have provided reverse paths in the processor to I/O couplers which also have enabling switches. Normally the forward paths to memory and I/O are turned on and the backward paths are shut off. Since these paths represent a hazard whereby a "sick" processor or device could damage healthy processors, we have arranged that only by storing a password at the proper address can a switch be changed. This greatly reduces the probability that a berserk processor painting memory will affect the path. A processor can neither enable nor disable its own access paths but one processor, deciding that another is sick and should be eliminated from the system, can amputate the bus of the offending processor. It can be similarly reinstated later.

The logic upon which amputation decisions are based is not yet fully understood and will be worked out as experience grows. We expect to require all processors to execute periodic healthiness-proving tasks. A regular system task, performed by any free processor, verifies that all processors have passed their tests and amputates any unhealthy one(s). Protective embellishments easily suggest themselves and we expect to do what seems necessary.

4.1.2 Discovery

The operational program implements the IMP algorithm with whatever hardware is working at a particular site at a given time. The program discovers the hardware configuration as follows: Memory is found by trying to access it; a failure interrupt results if Memory is not there. Processors are found by

accessing a register whose response indicates if the processor is absent, running or halted. I/O Devices are found by reading the 1st word of every possible device in I/O space -- a failure interrupt means no Device, a response returns a unique 16-bit device type. Any parameters needed to run the devices are available as status words in the 8-word block. It is somewhat harder to find where the bus boundaries are, but they too can be found by searching for the bus coupler disable switches. In the event that there is some property we cannot otherwise discover, we have set aside 3 registers (associated with the Real Time Clock) to hold this information. For example, the IMP number (used for network routing) is contained in 8 bits of these registers.

The Discovery logic is not an initialization phase; rather the program periodically runs through the Discovery logic and reconfigures whenever a change occurs. It thus automatically adapts the IMP algorithm not only to the wide variety of possible configurations but also to those which contain broken components.

4.1.3 Parity

At present the memories we are using do not store parity; however, we have built into our system design (and into the hardware) mechanisms to incorporate parity. These mechanisms have been tested with prototype parity memory and we have recently ordered parity memories for our production machines. We use a novel parity computation based not only upon the contents of a word but also on its address. The scheme also detects both "all ones" and "all zeros" failures. For writes to common memory, parity is computed at the processor end and fed, via the coupler, to the memory where it is stored with the word. Reads from

memory fetch this stored parity, which is compared to a recomputed parity at the processor end of the coupler, thus checking both the memory and coupler paths in both directions. For units on the I/O bus, in order to check the coupler paths, a special card computes and transmits parity for all words being read from the I/O bus by the processors and checks parity on all words arriving from processor busses.

4.1.4 Reloading

At present we use paper tape to load the system. The operator starts a processor which, from tape, loads its own private memory, its map registers and thereby any or all of common memory. It also loads, using backward coupling, the private memories on all other processor busses in the system. After the memory has been loaded, a startup procedure is executed which finally turns on the other processors.

Since all crucial switches, parameters, registers and control flip flops have been made addressable by reads and writes, loading the system and starting it up can be done by externally force feeding it with the right set of addresses and data. Although we presently use paper tape in conjunction with a bootstrap ROM executed by a processor for this purpose, we are planning to construct a means whereby the system can be force fed directly from the network. The mechanism for this is a device on the I/O bus which monitors phone lines from adjacent IMPs looking for a special format which signals arrival of reload information. The card then performs the reload by executing store type bus cycles using the reload data.

This sort of operation, which looks forward to elimination of paper tape, switches, and other operator dependent functions,

is appropriate to the IMP's job. If a running system fails, as viewed from the net, the first step is to send it a regular "for IMP" message which causes a standard system restart to be attempted. If that seems not to work, the next step is to send another regular message trying to activate the reload-from-the-net code in hopes that it is still intact. Only if that fails would one attempt to force a full restart from scratch, in which case the special card described above is called into play. The first data sent halts the processors in order to stop any interfering activity. Then the reload-from-the-net code is refreshed and finally a processor restarted running that code which then completes reload via the normal packet mechanism.

4.1.5 Mechanical Modularity

We have settled on a modular mechanical structure well matched to the modular logical structure of the system. This structure is important in that it allows easy construction of systems of varied size and permits repair of parts of a system while the rest of it continues to operate. The basic unit is a cooling module which houses either 1) a 16-slot bus complete with its own power supply, 2) a 24-slot bus without power, or 3) a power supply for such a 24-slot bus. These units, each with its own set of fans, sit on rails in a vertical tier in a rack, five of them filling a standard height rack. (The 14-processor system requires three racks.) Figure 4-2 shows how the cooling modules stack. Air flow is from back to front so that racks placed beside one another do not directly heat each other. A tilted pan at the bottom of each module separates the air flow between stacked modules, thus eliminating chimney effects. Cards plug in from the front and all device and coupler cables also

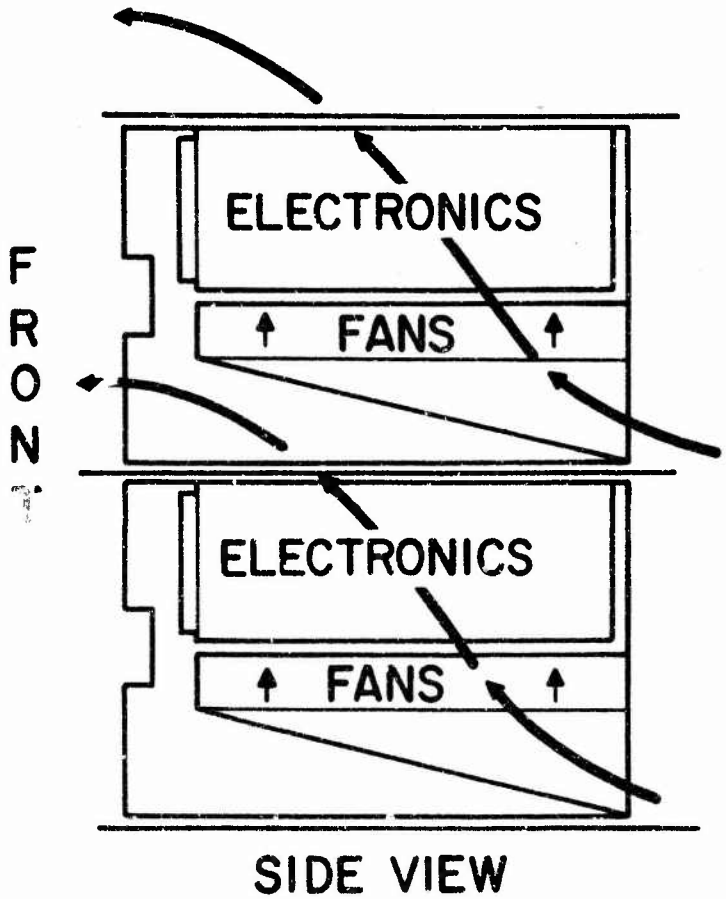

Figure 4-2 Mechanical Structure

connect on that side. An entire unit can be removed to the rear for repair or replacement of the bus, fans, etc. -- all without disturbing operation of the remainder of the system.

4.2 The Test Program

The primary design objective of the test program is to exercise all of the hardware as intensively and extensively as possible, detecting all failures and reporting them precisely and comprehensibly. Extensive testing implies a wide variety of test modules; intensive testing implies permitting the entire computational power of the system to be focused on individual components at times. These objectives led to the selection of a system based on processes, analogous to a time-shared system's jobs. Processes are not tied to processors; a given process will switch rapidly from one processor to another. Nor is a process in general tied to a specific copy of code; like time-shared jobs, processes share a single copy of sections of pure procedure.

There are four types of processes: the "system" processes, including the clock, timeout, and type-out processes; the device-specific processes, which are tied to particular I/O devices, two processes per device; the "GART" (Get A Random Test) processes, which select a test at random from a table of tests to be performed; and a dummy process, whose sole purpose is to assure that there is always a runnable process.

Each GART test is designed to test a particular element or feature of the system. These range from standard processor and memory tests (the latter are also useful for checking bus couplers) to exercising the various bus coupler switches and maps. The I/O devices are kept busy by circulating various data through

them. We are currently in the process of making the test program more robust so that it can survive transient problems, report on them and continue.

4.3 Where We Stand

Although the system uses Lockheed SUE processors, busses, memories, etc., we have so far designed and built nine BBN card types for the system: three coupler cards for each of the three bus types, a full-duplex memory channel card, a Host interface card (which operates at speeds up to 1.5 megabit), transmit and receive modem cards, the pseudo-interrupt card and a clock card. These designs are virtually all finalized and many are in production (printed circuit or similar) form.

We are presently finishing the design of two other cards: the first of these is the parity checking card for the I/O bus described above under the discussion of parity. The second is a checksum/block-transfer card which flows a block of memory through itself computing a checksum as it goes. This is used to checksum critical code from time to time [2], to compute checksums for network end-to-end checking of messages, and other useful checking purposes. A transfer mode can be enabled so that it can also be used to move blocks of information about in memory (checksumming as it goes if desired). In addition we are presently embarking on modifications to the modem transmit and receive cards which will allow them to deal with 1.5 megabit lines and design of the special interface which monitors incoming inter-IMP lines watching for reload information as described above.

At present we are running several systems. Two small systems are being used for testing and debugging of the IMP program.

These are sometimes run as separate single bus IMP systems which are connected together with our prototype 516 IMP into a three-node network. At other times the two busses are combined into a single system using a bus coupler. In this case one bus is used as a dual processor bus and the other as a combined memory and I/O bus. This system then works with the 516 IMP to form a 2-node net.

The growing prototype 14-processor system presently consists of three dual processor busses, two memory busses and one I/O bus (although a larger system with six of the seven processor busses has successfully run the test program). The system has grown gradually and it now operates with reasonable reliability under stress (shaking off cables, margining power supplies, shuffling of cards, etc.). By mid-1974 we hope to have two production copies of the large prototype working in the network. During 1974 we plan also to design satellite modem interface cards and to produce and deliver three moderate sized systems with satellite capability.*

The basic IMP system program is up and running in multiprocessor form, that is, with processors picking tasks up via the pseudointerrupt system and using locks to prevent interfering accesses to resources. It has been run on a five-processor system for short periods although most debugging has been on a two-processor system. The inner parts of the system, store and forward, Host, task, etc., seem solid. The work that remains is in implementing the system maintenance, monitoring, and debugging functions (i.e., system DDT, periodic status reports, etc.). This

*See section 5 of this QTR.

coding is about half done and needs finishing as well as debugging. The network error recovery code is ready for debugging. The special reliability code which keeps the system up when parts of the hardware fail is being designed.

At the other end of the performance spectrum, the small IMP is built on a single logical bus (consisting of two separate physical busses connected by an extender) which combines memory, processor and I/O. This system embodies none of the special reliability stemming from multiple hardware copies but is the least expensive version available. Small *reliable* systems are another matter and require, in general, doubling the system to provide complete redundancy of parts to allow for any single failure. Such systems may prove to be one of the more significant outgrowths of this development effort.

5. THE SATELLITE IMP

The communication circuits connecting the IMPs in the ARPA Network have typically been 50Kbs ground links, although 9.6Kbs and 230.4Kbs links are also in use. In the past year, two satellite links have been added to the network, one a 50Kbs link from California to Hawaii and the other a 9.6Kbs link from Washington, D.C. to Norway. In both cases these satellite links are conventional point-to-point links.

The likely introduction of further satellite links into the ARPA Network as it expands overseas, and the coming possibility of domestic satellite communication, have led to the development of a new variant of the IMP technology called the Satellite IMP. The Satellite IMP permits several network nodes to share a single satellite channel, enabling the nodes to statistically average their total load (at the satellite) rather than requiring each node-pair to average their traffic independently [3]. The manner in which the Satellite IMP provides such channel sharing is based on the concept of packet broadcast, first used in the ALOHA System [4].

When using a packet broadcast protocol, all nodes sharing the channel transmit discrete packets of data on the same transmission frequency. All nodes sharing the channel also receive on the same receive frequency, picking out packets addressed to themselves and discarding packets addressed to others.

The original ALOHA or "random" ALOHA system permits nodes to transmit their packets in a completely uncoordinated way. Thus, there is a substantial possibility of transmission conflict (and consequent destruction of the conflicting packets) with a frequent need for retransmission of packets. The retransmission

must somehow be randomized to avoid repeated conflict. As shown in [4], a channel operating in such a manner has an effective throughput capacity of 18%.

Roberts, in [5], pointed out that if an ALOHA channel were divided into slots (each able to hold a packet) and if the nodes modified their behavior to transmit packets so that the leading edge of the packet always coincides with the leading edge of a slot, even though the nodes remain free to transmit into a slot without regard for the transmission of other nodes, the effective channel capacity is doubled (to 36%). A channel operated in this manner is called a "slotted ALOHA" channel.

The Satellite IMP represents a practical application of the large body of theoretical knowledge about the operation of packet broadcast channels that has developed since the ALOHA System, and has been reported in [3, 5, 6, 7, 8, 9, 10]. The Satellite IMP is a conventional IMP with several additions and modifications both in hardware and software.

5.1 Hardware

The first hardware addition is of memory sufficient to buffer all the transmitted packets which can be awaiting acknowledgment simultaneously. Buffer space is necessary for some 32 packets assuming a 50Kbs channel and a one quarter-second propagation up to the synchronous satellite and back down. That is, 32 packets can be sent out before the acknowledgment returns for the first. The next hardware addition is a mechanism for signaling the satellite radio transmitter when to turn the radio carrier on and off as packets are transmitted; this is necessary because if two satellite ground stations have their radio transmitter carriers on simultaneously, they jam each other. The third

addition is time-keeping hardware necessary for the Satellite IMPs to accomplish accurate slotting. This hardware notes the arrival time of the leading edge of a packet (slot) and makes this time available to the program when the program fields the received packet interrupt and updates the program's estimate of the slot positions. This hardware also allows the program to accurately specify transmission of a packet at a specified time in the future.

One hardware modification was necessary for construction of the Satellite IMP. The IMP modem interface normally uses a unique character sequence to denote the end of a packet, thus requiring an escape character with escape-character doubling for data transparency. Escape-character doubling, however, can result in the length of a packet being temporarily increased while traversing the satellite, thus overflowing a slot. The IMP modem interface, therefore, had to be modified to use a word count to specify packet length.

5.2 Software

A number of software changes and additions are necessary to convert the normal IMP into a Satellite IMP.

We have implemented a slotting algorithm which operates as follows: each Satellite IMP tracks the location of the leading edges of all packets transmitted by other Satellite IMPs and averages them with exponential weighting to determine the average slot position, which is used as the standard.

We have added a mechanism for randomizing retransmissions. Whenever there is a packet for retransmission, we simply transmit into a slot or not, with a constant probability for each slot.

This is very simple to implement and nearly equivalent in effect to the harder-to-implement retransmission scheme analyzed extensively by Kleinrock and Lam [5, 8, 9].

The IMP had to be modified to process routing data coming from several different sources over a single channel. Each IMP can only be allowed to send routing data over the satellite channel once each routing period even though an IMP normally sends routing to *each* of its neighbors each routing period. When the IMP decides to route a packet out the satellite circuit, it must declare which Satellite IMP at the other end of the circuit is to receive and forward the packet to its destination. Neighboring IMPs normally exchange a pair of messages at least once each routing period to determine whether the IMP at the other end of the circuit (or the line between them) is alive. Over the broadcast satellite circuit, rather than pairwise exchange of these messages, each Satellite IMP broadcasts a message stating which of the other IMPs it has heard from (and it thus considers alive) since the last period.

The inter-IMP acknowledgment protocol normally allows only eight packets to be outstanding, awaiting acknowledgment, at a time. The satellite channel required expansion to thirty-two packets outstanding at a time. To simplify and optimize the inter-IMP acknowledgment scheme between Satellite IMPs, packets are acknowledged by the slot in which the packet arrived rather than independently for each source IMP.

Neighboring IMPs normally calculate the unused capacity and the delay over the circuit between them. This information is important to the IMPs' routing computation and the propagation of routing information across the network. The satellite version of

the IMP will eventually have to take into account that the delay over the broadcast satellite channel is statistical rather than fixed and that the excess capacity may have different characteristics over the broadcast satellite channel. The statistical nature of the delay over the satellite channel will undoubtedly have important consequences for network-wide timing.

Finally, use of a slotted channel requires modification to the IMP program reloading mechanism. Previously, IMPs reloaded by requesting and receiving a complete core image in a single transmission from an adjacent IMP. This technique has to be modified to send the core image one packet at a time.

5.3 Characteristics

The Satellite IMP as presently constructed includes all the features of a normal IMP including the ability to have Host connections. The Satellite IMP can have multiple ground circuits (fewer than five) to the terrestrial network, but only one packet broadcast satellite circuit. The Satellite IMP presently under construction will allow a satellite channel to be shared with only a small number of other Satellite IMPs. While the initial machine will undoubtedly be used with a 50Kbs satellite channel, it has the capacity to handle a channel on the order of 200Kbs. In fact, early calculations* suggest that Satellite IMP performance will have an upper bound of

$$7c/n + c < 1200 \text{ Kbs}$$

where n is the number of Satellite IMPs sharing the channel and c is the effective channel capacity. As with the IMP system, the Satellite IMP system will ultimately have some capability for measuring the performance and behavior of the Satellite IMP and broadcast channel.

*QTR No. 3, October 1973.

5.4 Testing

Testing the Satellite IMP requires a satellite channel simulator. For the purpose of testing a random ALOHA system, a satellite simulator was easily provided by inserting a conventional IMP with special software between two Satellite IMPs. The IMP is used as a quarter-of-a-second delay line between the Satellite IMPs, destroying packets from the two Satellite IMPs which are transmitted simultaneously. A more complete simulation is required to test a version of the system using slotting. This we provided by building a small hardware box to which four Satellite IMPs can be connected. The box ORs all data from the Satellite IMPs together, runs it through a quarter-second delay line, and passes the data to all the Satellite IMPs. Additionally, if more than one Satellite IMP has its carrier on/off signal on at any time, the box causes the two or more Satellite IMPs to jam each other.

5.5 Status

The most recent version of the Satellite IMP follows a random ALOHA protocol between a pair of nodes (two nodes are sufficient to test a broadcast protocol). We have almost completed another version of the Satellite IMP which follows a slotted ALOHA protocol between more than two nodes. A version of the Satellite IMP following a conventioral round-robin TDMA protocol would be very simple to implement, and we may experiment with this at some time in the future. Also, we will undoubtedly experiment with some novel reservation protocol(s), of which [3] and [7] have been given the most analysis to date.

As yet no Satellite IMP has been installed in the ARPA Network, mainly because of difficulty in choosing appropriate sites and obtaining tariff approval to carry out an experiment in packet broadcast communications. Presently we expect to install the first two Satellite IMPs in the first half of 1974.

REFERENCES

1. Heart, F.E. et al, *A New Minicomputer/Multiprocessor for the ARPA Network*, Proc. AFIPS 1973 NCC, pp. 529-537.

2. Crowther, W.R. et al, *Reliability Issues in the ARPA Network*, Proc. ACM/IEEE Third Data Communications Symposium, pp. 159-160, 1973.

3. Roberts, L.G., *Dynamic Allocation of Satellite Capacity through Packet Reservation*, Proc. AFIPS 1973 NCC, pp. 711-716.

4. Abramson, N., *THE ALOHA SYSTEM -- Another Alternative for Computer Communications*, Proc. AFIPS 1970 FJCC, pp. 281-285.

5. ARPA Satellite System Notes 1-53. A privately circulated series of notes contributed to by researchers from ARPA (L. Roberts), BBN (R. Rettberg, W. Crowther, F. Heart, S. Ornstein, and D. Walden), UCLA (L. Kleinrock and S. Lam), U. of Hawaii (N. Abramson, R. Binder, T. Gaarder, E. Kanehira, and S.-C. Liu), and Xerox PARC (R. Metcalfe).

6. Abramson, N., *Packet Switching with Satellites*, Proc. AFIPS 1973 NCC, pp. 695-702.

7. Crowther, W. et al, *A System for Broadcast Communication: Reservation-ALOHA*, Proc. HICSS-6, pp. 371-374, 1973.

8. Kleinrock, L. and Lam, S., *Packet-switching in a Slotted Satellite Channel*, Proc. AFIPS 1973 NCC, pp. 703-711.

9. Kleinrock, L. and Lam, S., *On Stability of Packet Switching in a Random Multi-Access Broadcast Channel*, HICSS-7, 1974.

10. Metcalfe, R., *Steady-State Analysis of a Slotted and Controlled ALOHA System with Blocking*, Proc. HICSS-6, pp. 375-378, 1973.