Presentation to Association Francaise des Sciences et Technologies de l'Information et des Systemes (AFCET)

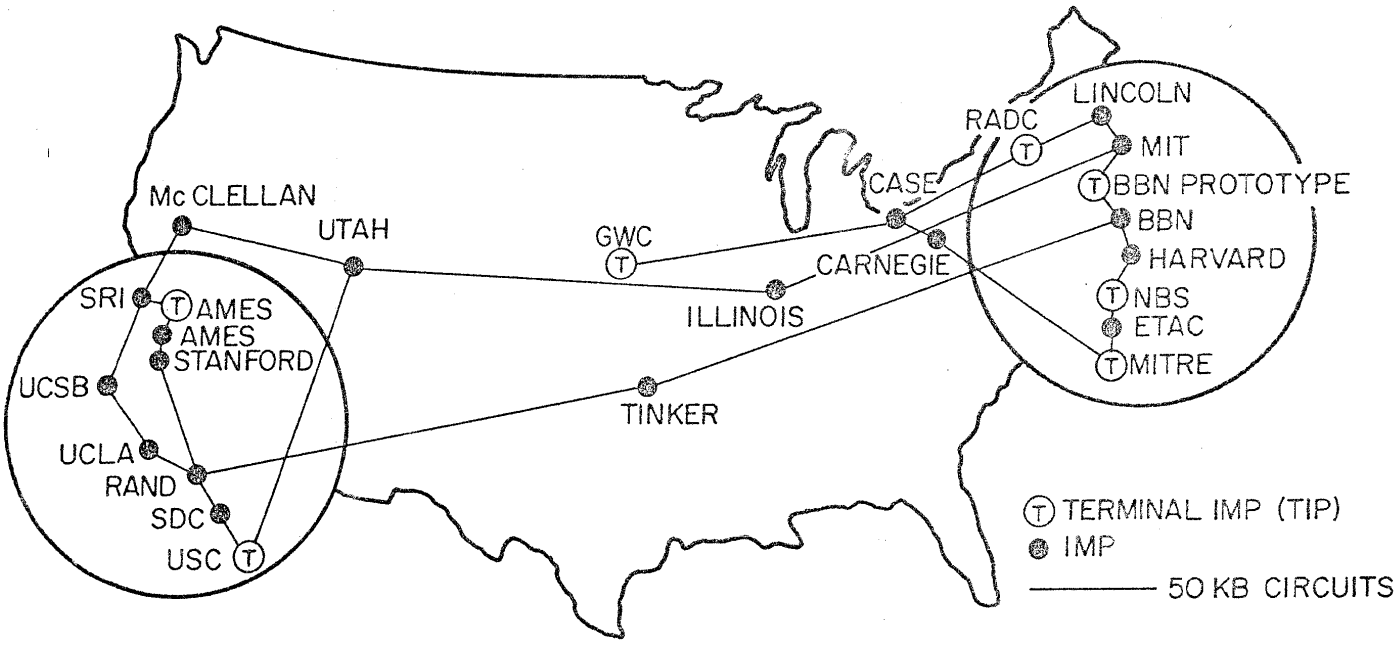
David Walden

Paris, Francis, May 25, 1972

(Introductory remarks)

The title I have chosen for my talk today is The Interface Message Processor, Its Algorithms, and Their Implementation. In the talk I propose to discuss the system design of the communications subnetwork of Interface Message Processors (or IMPs) in the ARPA Computer Network, and how this communications system has been implemented with hardware and software. The fact is that a significant portion of the IMP design has been implemented in hardware, and the operational IMP is far from being a standard Honeywell computer. However, in keeping with the area the organizers of this conference suggested, the talk will emphasize those portions of the system implemented in software. This emphasis is appropriate as it is the software we have been readily able to change as we gained experience with the network and the functioning of our algorithms, and near the end of the talk I shall report on a major modification to the IMP program correcting several significant flaws found during the first two and one-half years of the ARPA Network's operation.

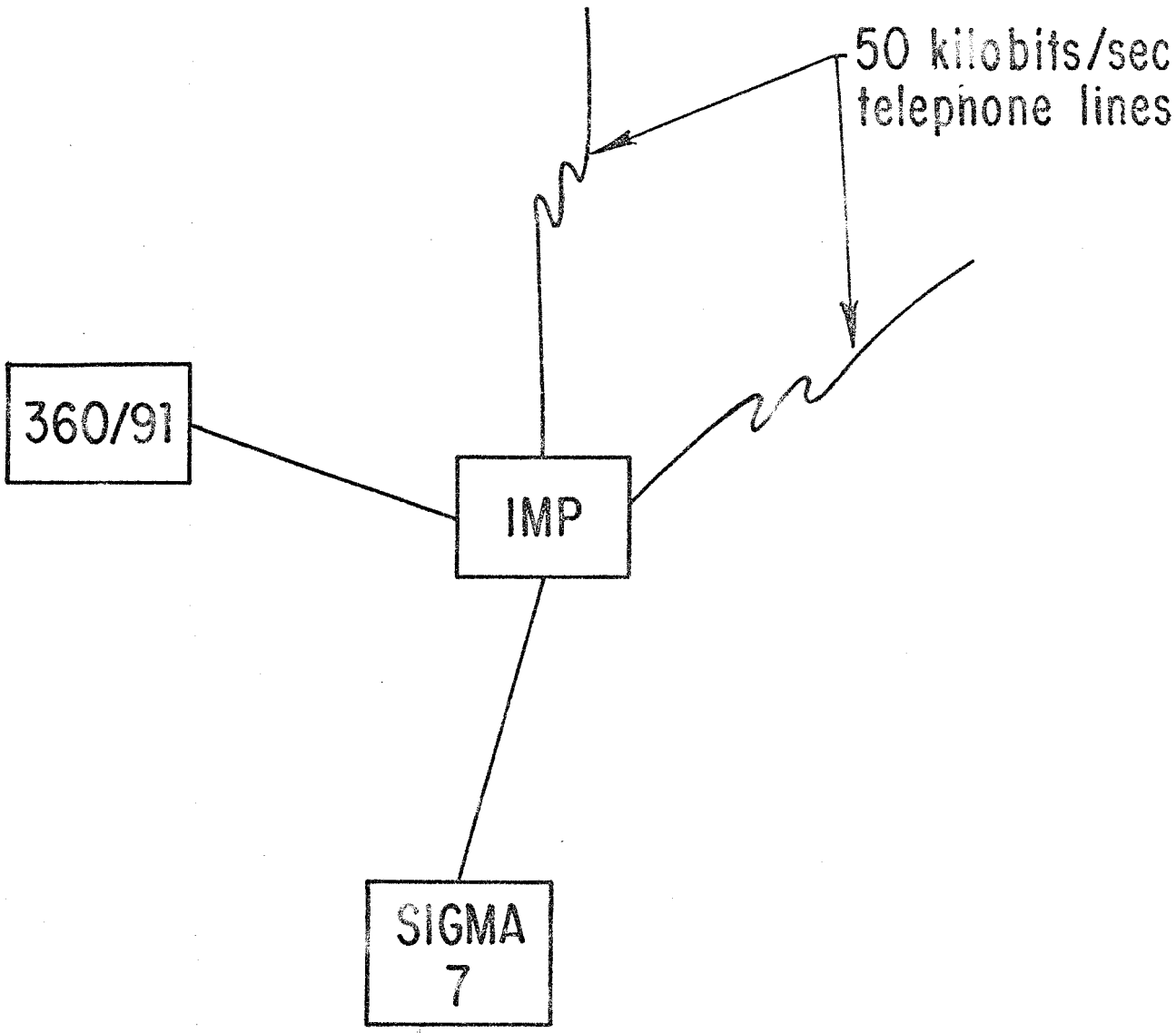
You will probably have noticed by now that the documents you have in front of you do not include the text of my talk. I trust you will forgive me for this as there is a good reason for not submitting the written text of my talk. The reason is this. The IMP project is very much a team effort and it is as a team we have written the major papers discussing the IMP. However, our work schedule does not now permit us to spend the time and effort to produce the team effort necessary to write the major paper required for the important topic the organizers of this conference picked for me. Therefore, in lieu of the complete written text, you have been given an extended abstract. At the end of the abstract is a list of papers written by the IMP group, and by reading these papers you will perhaps get a more accurate view of how the IMP group sees the ARPA Network and the IMP than the view you will hear from me today. The papers are listed in more or less chronological order. Although I have listed exclusively papers written by members of the IMP group at Bolt Beranek and Newman, that does not mean that we do not recognize the efforts of others in the field. In fact, each of the papers in the list has an extensive bibliography. We have particularly been influenced by Dr. Lawrence Roberts and Mr. Steve Crocker of the Advanced Research Projects Agency (or ARPA), Professor Leonard Kleinrock and his students at the Network Measurement Center at the University of California at Los Angeles, Dr. Howard Frank and his colleagues at the Network Analysis Corporation, and the network group at the National Physical Laboratory in England.



ARPA NETWORK, GEOGRAPHIC MAP, MARCH 1972

(figure -- geopraphic map of the ARPANET)

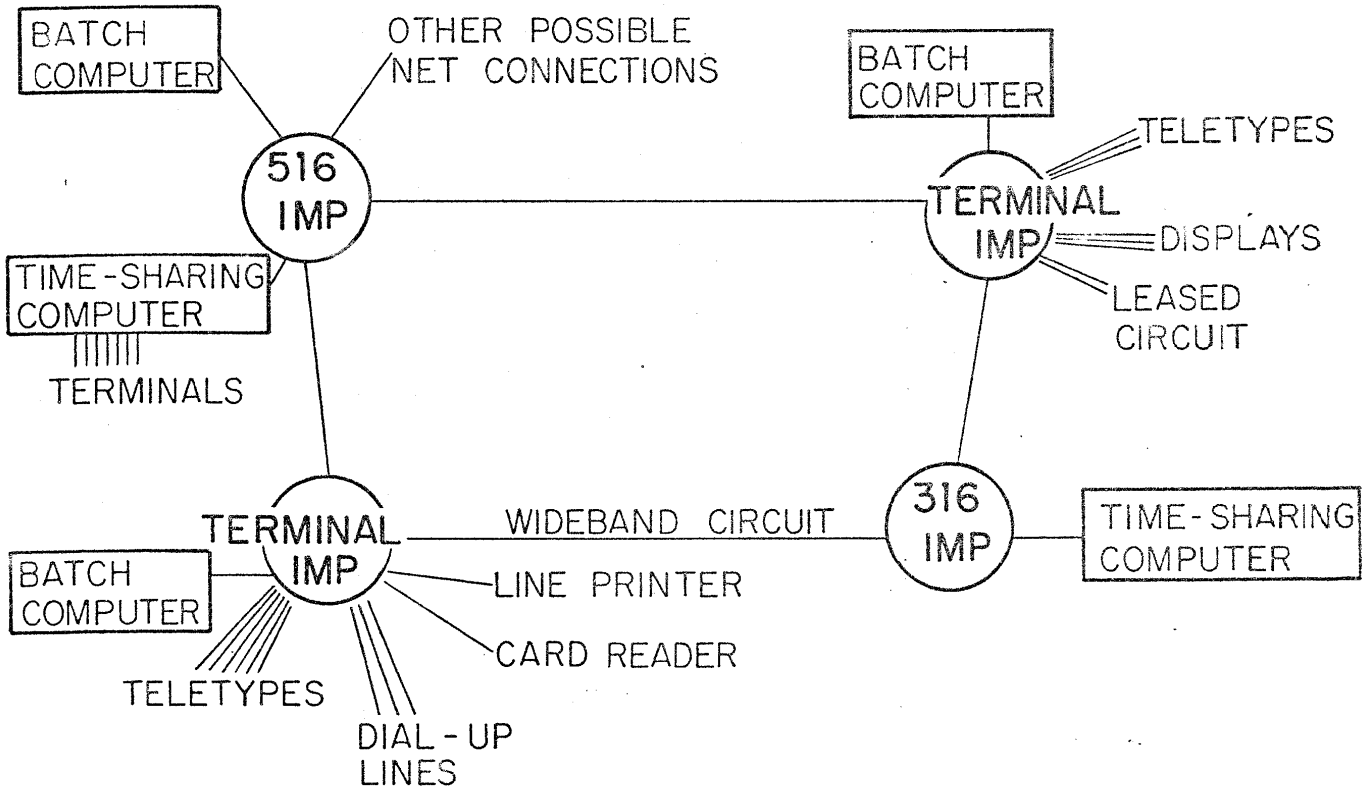
As can be seen from the map, the ARPANET consists of a number of geographically separated sites spread across the United States.


50 kilobits/sec
telephone lines
360/91
IMP
SIGMA
7

(figure -- typical site)

Each site consists of one to (potentially) four independent computer systems called Hosts and one communication computer system called an Interface Message Processor or IMP. All of the Hosts at a site are connected directly to the IMP.

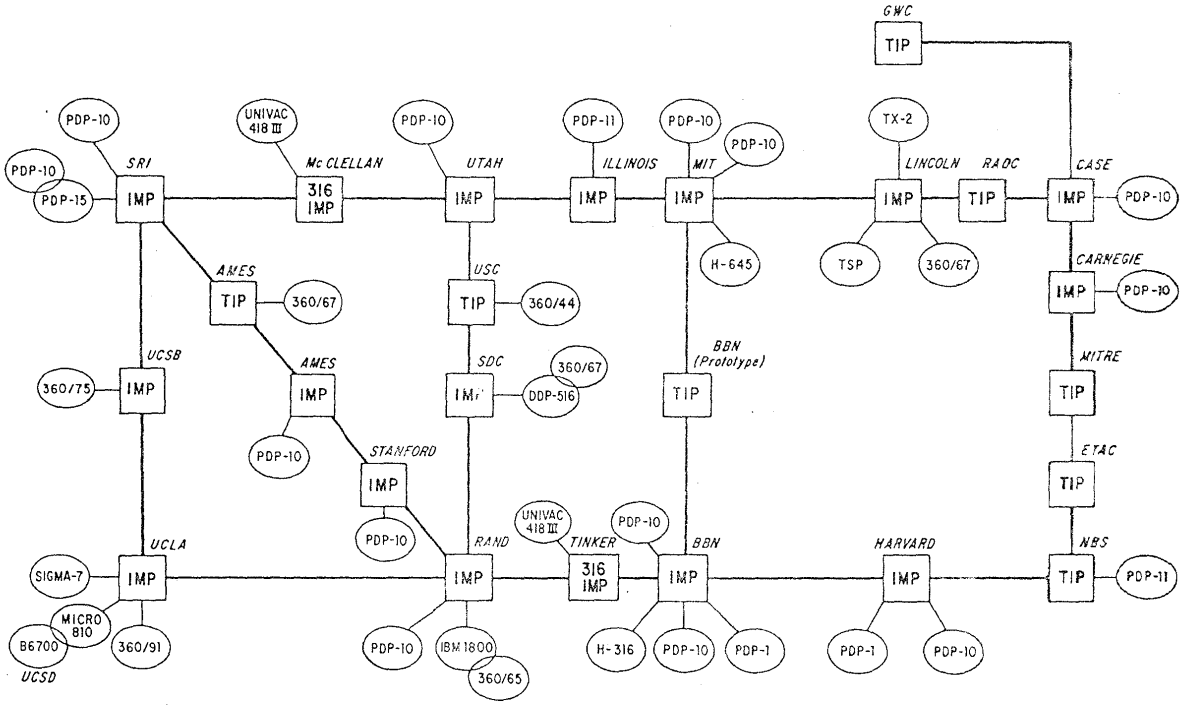
A SIMPLE IMP NETWORK



(figure -- example IMP and Host network)

The IMPs themselves are connected together by phone lines, although each IMP is only connected to two-to-five other IMPs. The phone lines typically have a bandwidth of 50 kilobits per second. The IMPs provide a communications subnetwork through which the Hosts communicate.

Suppose one Host wishes to send data to another Host. The first Host presents a chunk of data, called a message and required to be less than about 8100 bits long, to its IMP for transmission. On the message is written an address or destination. The IMP promptly breaks the message into things which are one thousand or less bits long and are called packets. Each packet is given the address the message had as a whole. The IMP looks at each packet's address and makes a decision to which of the neighboring IMPs to send the packet and sends it. The next IMP looks at the packet's address and again makes a routing decision and passes the packet along toward its destination. Finally, the packets of the message arrive at an IMP which passes them into the destination Host. So that the source Host may know that the message has been successfully transmitted to the destination, the destination IMP contructs an acknowledgment message, which we call Ready For Next Message or RFNM, which is sent back across the network to the source Host.



ARPA NETWORK, LOGICAL MAP, MARCH 1972

(figure -- logical map of the ARPANET)

In the actual ARPANET, which is shown in the figure as of March this year, the process of passing packets from IMP to IMP can involve a rather large number of IMPs with a routing decision at each. For instance, consider sending a message from MULTICS at MIT to the IBM360/91 at UCLA. There are a minimum of five hops between the two sites and numerous alternate routes.

All right. Now you have heard a once over lightly description of the operation of the IMP subnetwork. So far probably most of you have heard little that is new to you. So let us now consider the operation of the IMP subnetwork in a little greater detail. I'll try to point out some of the more significant algorithms used in the IMP subnetwork as I go along, and I'll try to say a few words about the implementation of each.

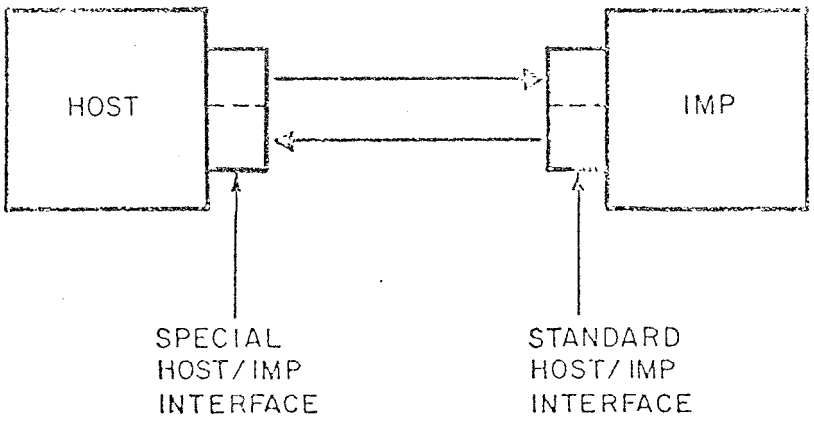


FIG.2-4 HOST/IMP INTERFACE

(figure -- Host/IMP interface)

We already noted that a Host presents messages to its IMP which are less than about 8100 bits long. These messages are transmitted between the Host and IMP over the Host/IMP interface which is rigorously defined in reference 7. This interface has two parts, the hardware part and the software part.

The hardware part itself has two parts, a standard portion supplied with the IMP which is identical (almost) for all Hosts and a special portion supplied by the Host. For simplicity and greater power, the standard interface has been defined to be full-duplex. Electrically, the standard interface follows a bit by bit handshaking procedure. The procedure is something like "I'm ready to send a bit." "I'm ready to receive a bit." "Here's the bit." "I got the bit." "Good!" The procedure also provides for saying "That's the last bit in the message." In our opinion this procedure is important. We have to connect together all kinds of disimilar computers with different word lengths, different speeds, different loadings, and so forth; and we did not want to place constraints on the Host's behavior. We wanted both the Host and IMP to be able to start and stop transmission whenever they wished. That is why an asynchronous, bit by bit serial interface is used. Note that the algorithm I have just described is implemented entirely with hardware.

The software interface between an IMP and a Host is also simple, using a minimum number of control messages. The Host specifies to its IMP the destination of a message and a few other things in the first 32 bits of the message. This 32 bits is called the leader. Messages arriving at the Hosts have the same information in the first 32 bits of the message except the destination is replaced by the source.

Neither the hardware nor the software interface between the IMP and a Host puts any constraint on the content of messages other than that they must have legal leaders and must have less than the maximum length. In particular, messages may be sent through the network containing arbitrary sequences of bits.

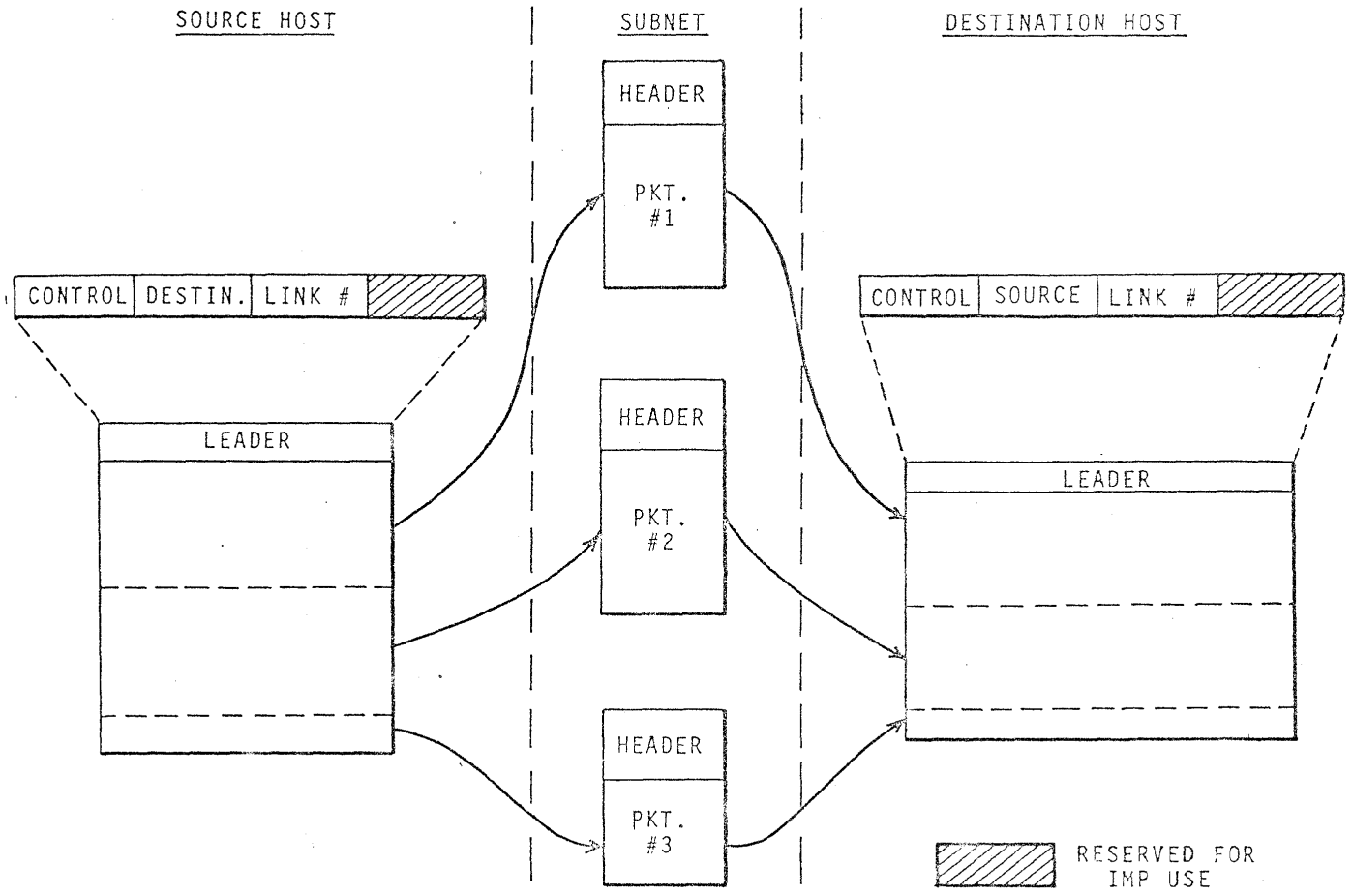


FIG. 3. MESSAGES AND PACKETS

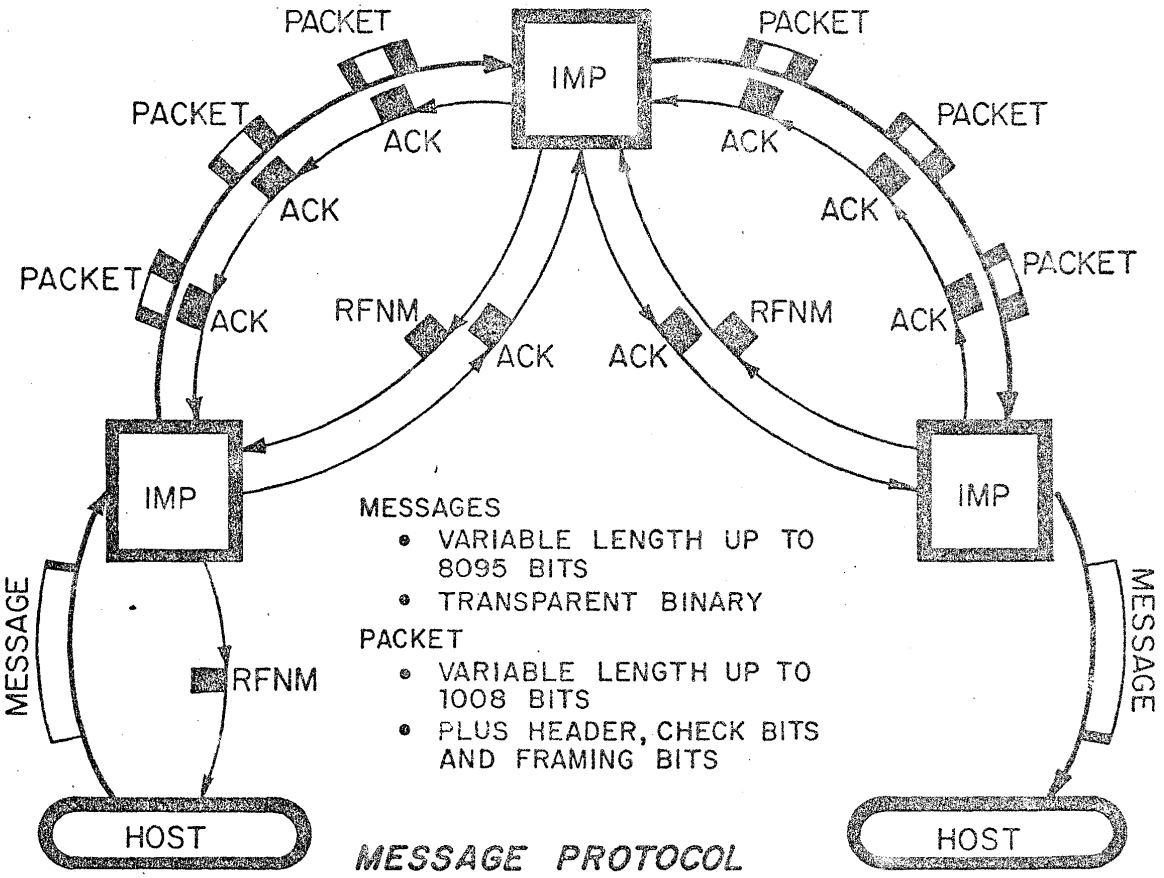
(figure -- segmentation of a message into packets)

On the left of the figure we see an example of a message, this one something over 2000 bits long. Notice the leader on the front of the message which includes some control information, the destination of the message, and something called the link number which is used as a message identifier, between the Host and the IMP.

As stated before, the IMP breaks messages arriving from its Hosts into packets 1000 or less bits long. There are several reasons for breaking messages into packets: 1) a packet is more convenient to buffer in the IMPs than messages would be; 2) shorter checksums are sufficient for packets than would be required for messages; but, mainly, 3) the earlier packets in a message can begin their journey across the network while the later packets are still coming into the IMP from the Host. This last point we feel is very important. So important, in fact, that we go to all of the trouble of handling multi-packet messages when a system based on only one packet messages would be enormously more simple.

As the IMP segments a message into packets, it appends to the front of each packet some control information called the header. The header contains the destination, the packet number, a sequence number which is used for ordering messages upon their arrival at the destination and for discarding duplicate packets and messages, and so forth.

This segmentation into packets is completely invisible to the Hosts as the packets of a message are reassembled into the message at the destination IMP before the message is sent into the destination Host.



MESSAGE PROTOCOL

(figure -- three color message/packet/ack/RFNM)

In this figure we can again see a message being broken into packets -- this is the message leader. We can see the packets being transmitted between IMPs -- these are the packets' headers. And we can see the reassembled message being passed into the destination Host. The figure also shows the Ready For Next Message or RFNM which we mentioned earlier, the acknowledgment of the message, being passed back across the network from the destination IMP to the source Host.

Also shown in the figure are these little "ack" things. These are positive acknowledgments of packets as they are transmitted between IMPs. Packets are transmitted onto the phone lines via a 24-bit feedback shift register which computes a check sum for the packet. This is the check sum. As a packet is received from the phone line by the IMP, the packet is drawn through a matching 24-bit feedback shift register which recomputes the packet's check sum and checks whether the packet still has the same checksum as when it was transmitted. If the packet has been received incorrectly, it is discarded. If the packet has been correctly received, a positive acknowledgment is return to the transmitting IMP. Each IMP holds on to a packet until it gets a positive acknowledgment from the next IMP down the line that the packet has been properly received. If it gets the acknowledgment, all is well; the IMP knows that the next IMP now has responsibility for the packet and the transmitting IMP can discard its copy of the packet.

If the transmitting IMP does not receive a positive acknowledgment soon, it retransmits the packet, thus producing essentially error-free transmission. It keeps on doing that. Individual packets, even within a single message, may go out over different lines to the same destination. At every node the packets go though an IMP, each IMP stores the packet in core memory, and sends it out again along the next line, each time tenaciously holding the packet and retransmitting it until it is acknowledged. As we have already said, at the destination IMP the packets are reassembled into a message and transferred into the destination Host computer. The process of acknowledgments and retransmission in their absence can result in duplicate packets. These must be sorted out at the destination IMP.

Notice that the Ready For Next Message is itself a message and is acknowledged as it passes between each pair of IMPs.

Let me emphasize two points about packets:

First, packets are permitted to be of variable length from one character (excluding the header) to about 1000 bits (including the header). The IMP's modem interface transmission hardware adds framing characters to each packet as it is transmitted onto a phone line. At the front of the packet two

characters (DLE & STX) are added indicating the beginning of the packet. At the end of the packet two characters (DLE & ETX) are added indicating the end of the packet. The checksum is appended after the end of packet characters. The IMP's modem interface reception hardware has the capability of detecting the start and the end of a packet from these characters thus freeing the program from the burden of detecting packet boundaries.

Second, there is no restriction on the content of a packet. The system can handle transparent binary with no code restrictions. This transparency is achieved by a method known as DLE doubling (QUOTE QUOTE for fans of LISP). If the data itself contains a DLE, the character which introduces the packet start and end characters, that DLE is doubled by the transmission hardware. At the receiver, the hardware collapses double DLEs back into one. So there is complete transparency on the IMP to IMP channels as well as the Host to IMP channels. This is a very important point. All too many networks require transmission to be limited to characters from a particular character set. Besides preventing the network's users from sending arbitrary messages, the designers of the network are prevented from such things as loading programs over the network.

Notice that much of our IMP to IMP communication algorithm is performed with hardware. Programs are particularly bad at generating checksums, scaning for message boundries, and so forth, especially when it must be done on a character by character basis. Therefore, we do this with hardware.

You now have a fairly detailed understanding of the design of the IMP subnetwork. Our main concern has been that bits get transmitted reliably across the network, and to this end a number of other features have been built into the IMPs and the network which are worth mentioning.

- cross patching
- automatic power failure recovery
- watchdog timer
- network control center
- network measurement center

(figure -- other features of the IMP)

First there are our cross patching facilities. Whenever something goes wrong in a complicated system, there is a great deal of finger pointing, and the more people involved, the more fingers there are to point. We felt it was essential to be able to isolate faults; and so all the interfaces on the IMP, both to the Hosts and to the lines, can be automatically cross patched, output back into input, under program control. We are able to experiment and to decide whether a fault is in our IMP or the telephone lines or the Host. If we tell the telephone company it is their phone line, then it always is their phone line; we are never wrong about that. This is a very important point. People who build systems have to think through, from the beginning, how they are to be tested under actual operation. The employment of automatic program controlled cross patching has turned out to be an extremely powerful tool.

The IMP has automatic power failure recovery. When the power fails, which happens rather frequently in the United States, the IMP turns itself off cleanly. When the power returns, hopefully equally frequently, the IMP turns itself back on and starts up at the correct place.

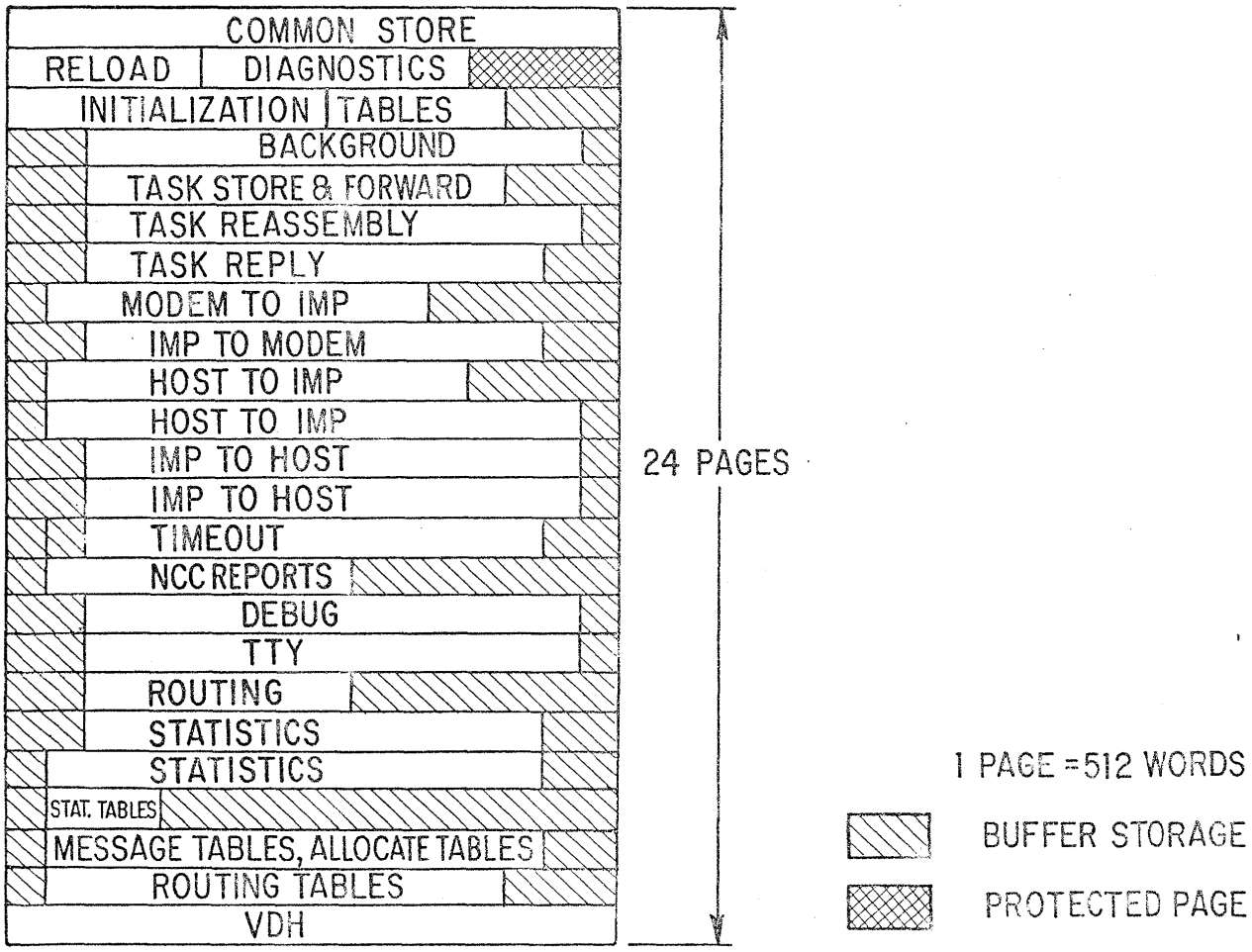
The next point is a little more complicated. This is a system called the watchdog timer. Every few hundred milliseconds if it is operating correctly, the program resets the watchdog timer. If the watchdog timer is not reset for too long, it decides that program is not operating properly and generates an interrupt to a particular place in memory. The program at this place in memory then tries to reload a new copy of the whole IMP program from one of the neighboring IMPs. This scheme is not foolproof, but it does repair some program breaks. It also has an interesting and powerful side benefit. The fact that an IMP can be reloaded from its neighbors means we can change the programs in all of the IMPs from BB&N. We can set up at BB&N, make a change in the program in our IMP, and then cause the entire system to reload, suitably controlling that process from BB&N. This has turned out to be a powerful tool for a distributed network of this kind.

At BB&N we also have a network control center which can do things like debug programs in remote IMPs. Each IMP in the field has something which has been labelled DDT, a debugging program which is part of the operational IMP program, and which can be run remotely from the network control center. We can examine core, change core, and look into the system as it is operating in each of the IMPs in the field. When a Host has trouble debugging its IMP/Host program, for instance, we can look at IMP core and help the Host debug their program without leaving home. This saves a lot of airplane trips. The network control center also receives a message from every IMP in the network every few minutes that gives the operational status of the IMP, the status of the Hosts connected to the IMP, and the status of the telephone lines connected to the IMP. These messages also include a record of the amount of traffic that is passing between the IMP and its Hosts. This information is tabulated monthly and is passed on to ARPA to give them some indication of the utilization of their network.

Since the ARPA network is an experimental system, a number of people are interested in how it works. One of the Host organizations, Professor Kleinrock's group at UCLA, was given the specific contract to measure the performance of the network as it develops. This is done at the network measurement center at UCLA. Because of the desire to measure performance, the IMP program includes a lot of extra storage space and code to examine themselves as they operate. This is not done all the time, but can be turned on when required. Synchronized snapshots of all the IMPs can be taken every one-half second, and traffic summaries over ten second intervals can be taken. The snapshots and summaries are packaged up in special messages which are sent to Professor Kleinrock or at least to UCLA. To aid in the measurement process, the IMPs can also be made to generate artificial traffic. There is also a very powerful facility which we call tracing available in the IMPs. The notion here is that one can put a special bit on in messages which will cause every packet in that message to be traced as they pass through IMPs on their way to their destination. Thus as each packet to be traced passes through an IMP, the IMP sends a special message to the network measurement center saying that the packet just went by. This provides a method of making a very fine grained study of the operation of the system, but it also tends to create an enormous number of messages, all of which are dumped on Professor Kleinrock.

The organizers of this conference specifically asked me to discuss the IMPs software, so I'll do that now.

Implementation of the IMPs required the development of a sophisticated operational computer program. The principal function of the 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 store and forward packets, retransmitting unacknowledged packets, reassembling packets into messages for transmission into a Host, and generating Ready For Next Messages, acknowledgments, and other control messages. The program also monitors network status, gathers statistics, and performs on-line testing. This real-time program is an efficient, interrupt driven, involute machine language program that occupies somewhat more than 6000 words of 16-bit memory. It was originally designed, constructed , and debugged over a period of about one year by three programmers. More recently, after about two and one half years of operation in what is now 25 IMPs throughout the network, the operational program has undergone a significant revision. This revision was implemented and debugged by one programmer over a period of about six months.


COMMON STORE
RELOAD
DIAGNOSTICS
INITIALIZATION
TABLES
BACKGROUND
TASK STORE & FORWARD
TASK REASSEMBLY
TASK REPLY
MODEM TO IMP
IMP TO MODEM
HOST TO IMP
HOST TO IMP
IMP TO HOST
IMP TO HOST
TIMEOUT
NCC REPORTS
DEBUG
TTY
ROUTING
STATISTICS
STATISTICS
STAT. TABLES
MESSAGE TABLES, ALLOCATE TABLES
ROUTING TABLES
VDH
24 PAGES
1 PAGE = 512 WORDS
BUFFER STORAGE
PROTECTED PAGE

(figure -- map of core storage)

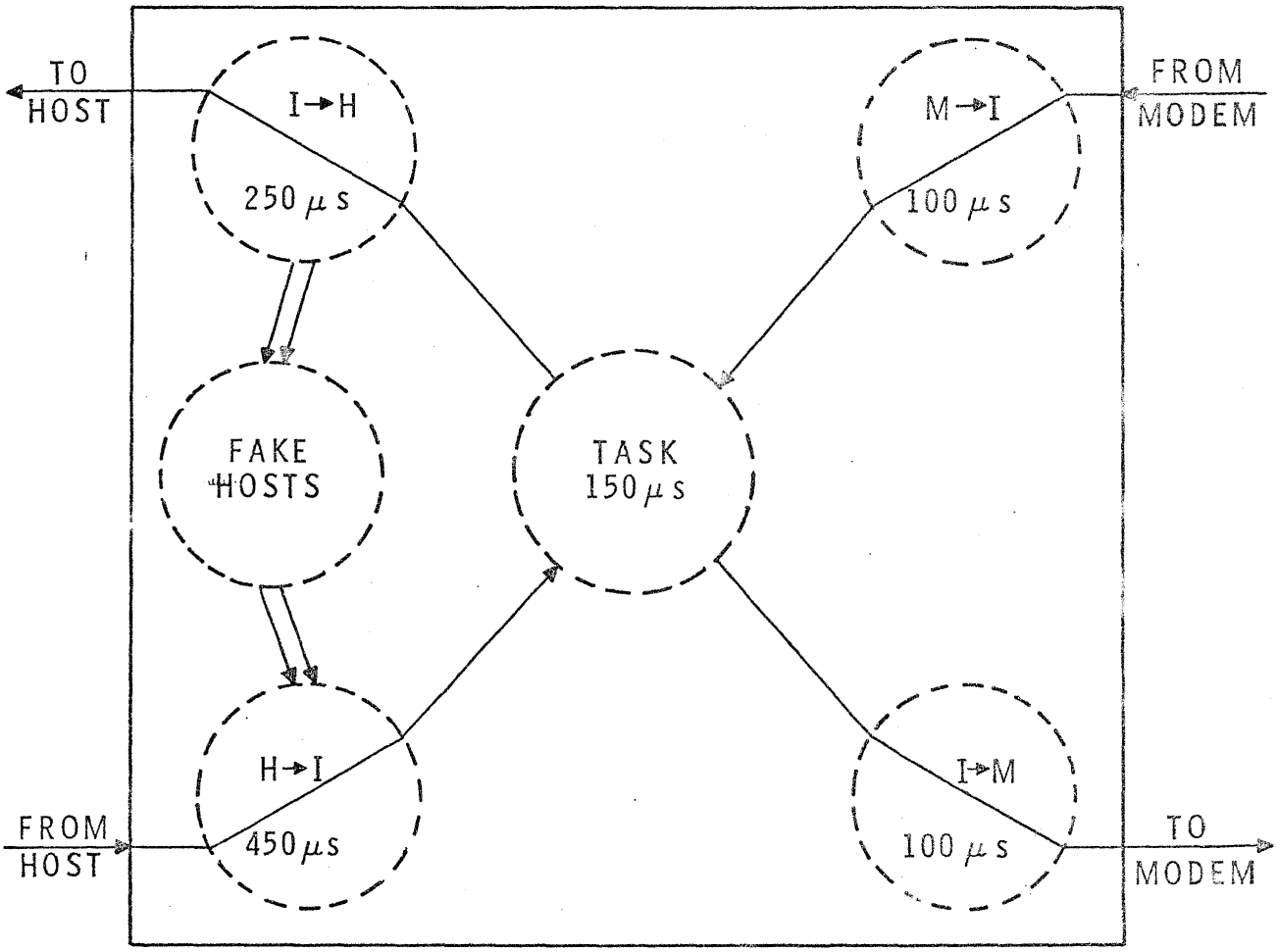
The operational IMP program is composed of a number of functionally distinct pieces; each piece occupies no more than one or two pages of core -- a core page is 512 16-bit words. These programs communicate primarily through common registers that reside in page zero of the machine and are directly addressable from all pages of memory.

The progams main data structures are tables and queues. The buffer storage space is partitioned into about 50 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 passed onto another IMP or a Host, its buffer is returned to the free list. The buffer space is partitioned in such a way that each process (store-and-forward, 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 forware traffic, all processing is on a per packet basis. Further, although traffic to and from Hosts is composed of messages, the IMP rapidly converts to dealing with packets; the Host transmits (we presume) 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 memory. An equivalent inverse process occurs on output to the Host at the destination IMP. No attempt is ever made to collect the packets of a message into a contiguous portion of the memory.

Buffers currently in use are either dedicated to an incoming or an outgoing packet, chained on a queue awaiting processing by the program, or being processed. Occasionally, a buffer may be simultaneously found on two queues.


TO
HOST
I→H
250 μs
M→I
100 μs
FROM
MODEM
FAKE
HOSTS
TASK
150 μs
H→I
450 μs
I→M
100 μs
FROM
HOST
TO
MODEM

(figure -- simplified five circle diagram)

The figure shows in schematic form the five program modules most directly concerned with packet processing.

The Host to IMP routine handles messages being transmitted from one of the IMP's Hosts. The routine uses the leader to construct a header that is prefixed to each packet of the message. It also segments the message into packets and passes each of the packets to the task routine and pokes the task interrupt. The routine then acquires a free buffer and sets up a new input. The routine is serially reentrant and services all Hosts connected to the IMP.

The task routine uses header information to direct packets to their proper destination. The task routine is driven by the task interrupt, a program settable interrupt, which is set by each program which puts a packet on the task queue. Packets for a local Host are passed to the IMP to Host program. Packets for other destinations are placed on a modem output queue as specified by the routing table.

The IMP to modem routine transmits successive packets from the modem output queues, and sends acknowledgments for packets received by the modem to IMP routine.

The modem to IMP routine receives packets from the phone lines and passes them to the task routine.

The IMP to Host routine sets up successive outputs of the packets of a message found on the Host output queue.

It also constructs Ready For Next Messages for messages passed to a Host and gives them to the task routine.

F ≡ fast time out

M ≡ medium time out

S ≡ slow time out

(figure -- timeout routine execution pattern)

The time-out routine is started every 25.6 milliseconds by a clock interrupt. The routine has three sections: the fast time-out, the medium time-out and the slow time-out. The timeout routines do such things as wake up Host or modem routines which have asked to be awakened, garbage collection of tables, transmission and updating of the routing tables, marking lines alive or dead, and so forth.

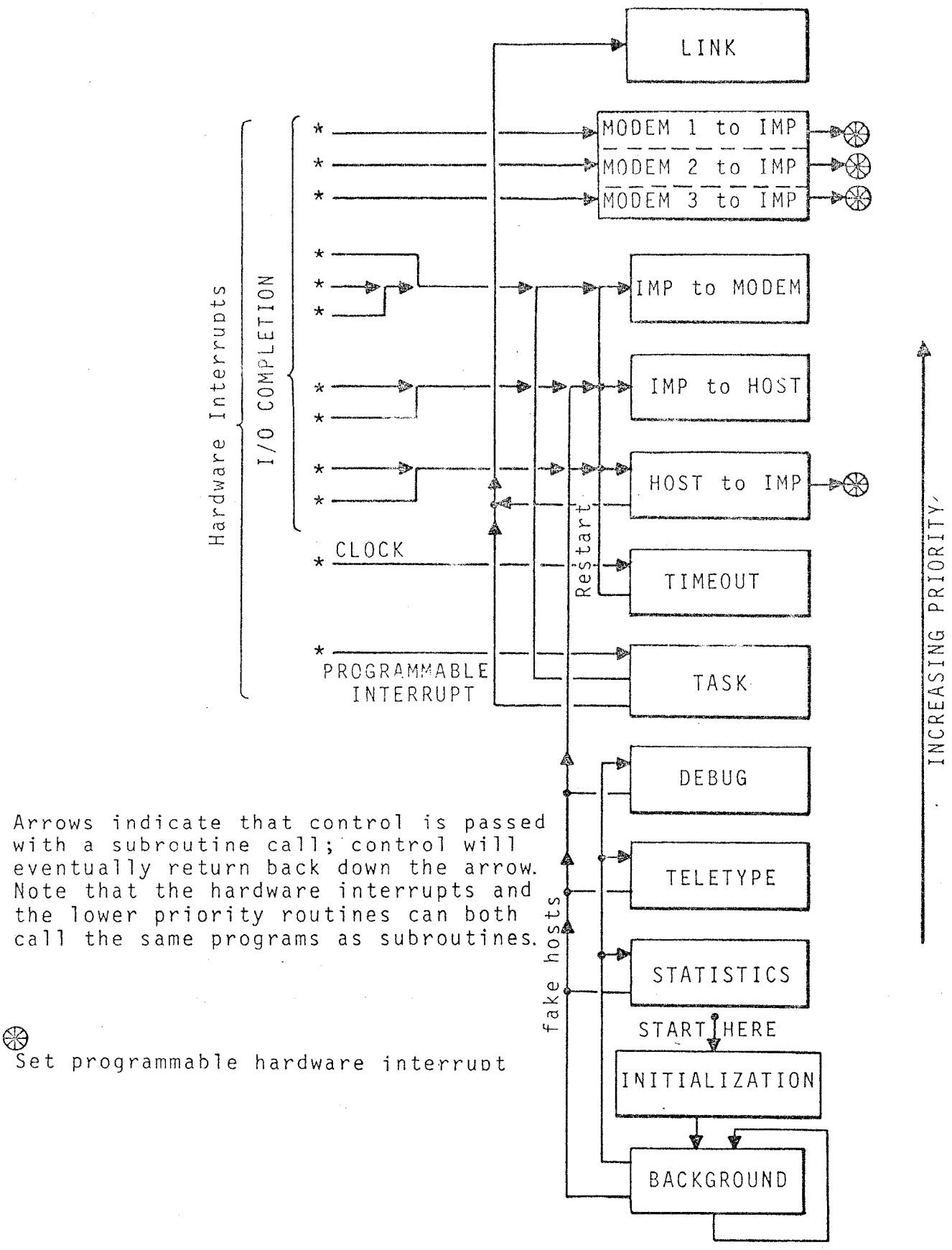
The three routines execute in the pattern show in the figure, and although they run off a common interrupt, are constructed to allow faster routines to interrupt slower routines should a slower routine not complete execution before the next time-out period.

As just mentioned, one of the tasks of the timeout routines is garbage collection. Every table, most queues, and many states of the program are timed out. Thus if an entry should remain in a table abnormally long or if a program (say IMP to Host) should remain in a particular state for abnormally long, this entry or state is garbage collected and the table or program is returned to its initial or nominal state. In this way, abnormal conditions which do happen in big systems, especially when autonomous computer such as a Host and the IMP are communicating, are not allowed to hang the system up indefinitely.

The way we frequently scan a table from the timeout routines is interesting. For instance, suppose we must look at every entry in a 64 entry table every now and then.

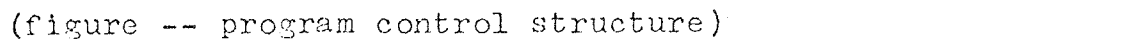
We could just wait for the proper interval and then look at every entry in the table on one pass through the timeout routine. However, this would cause an enormous transient in the average timing of the IMP program as a whole. So instead, we look at one entry each time through the timeout routine. This takes a little more time total but is much less disturbing to the program as a whole. A particular example of the use of this technique is with the transmission of routing information to the IMP's neighbors. In general, an IMP can have five neighbors. Therefore, it sends routing information to one of its neighbors at each of the medium and the slow timeout periods.

The program has a small initialization section and a sizable background loop. The backgroud loop includes handling the IMP's teletype, a debugging programming which resides in each IMP, the statistics programs, the trace program, and several other functions. The network control center and the network measurements center frequently find it useful to communicate with one of these background programs. So that these programs may send and receive messages from the network, they are treated as fake Hosts. The Host to IMP and IMP to Host routines both think they can handle eight Hosts although there will never be more than four real Hosts on an IMP. The other four Hosts are these background programs which simulate the operation of the Host/IMP data channel hardware so that the Host/IMP routines are unaware they are communicating within anything other than a real Host. This trick saved a large amount of code.



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 -- program control structure)

It is characteristic of the IMP program that many of the main routines are entered both as subroutine calls from other programs and as interrupt calls from the hardware. The resulting control structure is shown in the figure. 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 higher priority programs are finished. No program may execute 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.

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, we modified the computer hardware to include a low-priority hardware interrupt that can be set by the program. When this interrupt has been honored (that is, when all other interrupts have been serviced) the task routine is executed. Thus, control is directed where needed without violating the priority rules.

Some routines must occasionally wait for long intervals of time. 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 back to

of responsibility among various programs achieve the following timing goals:

1) No program stops or delays the system while waiting for an event.

2) The program gracefully adjusts to the situation where the machine becomes compute-bound.

3) 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.

4) The program will almost always deliver packets waiting to be sent as fast as they can be accepted by the phone line.

5) Necessary periodic processes (in the timeout routine) are always permitted to run, and do not interfere with input-output processes.

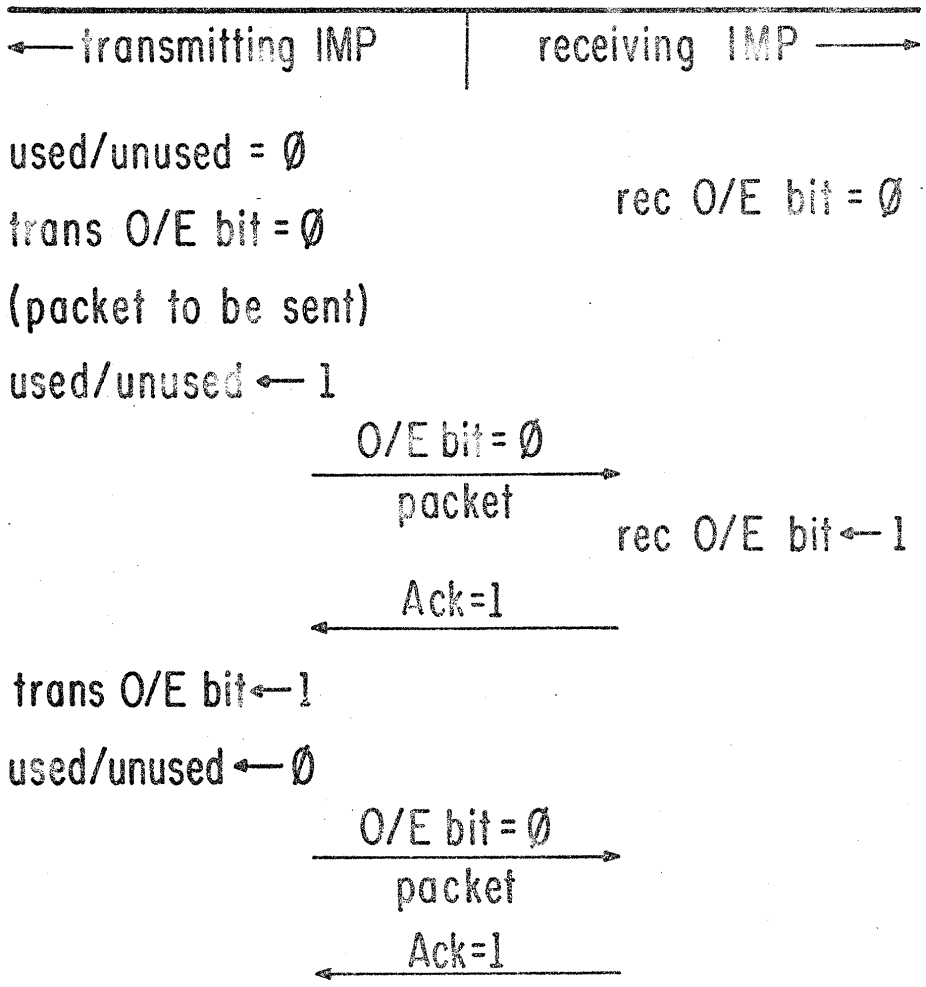
- IMP-to-IMP transmission control
- source-to-destination transmission control
- flow control and lockup prevention
- routing

(figure -- four algorithms)

The last section of my talk will describe in considerable detail four of the algorithms used in our subnetwork design. Each of these four algorithms is implemented with software. The algorithms are:

1) the IMP to IMP transmission control algorithm
2) the IMP's Host to Host transmission control algorithm
3) the flow control and lockup prevention algorithms
4) the routing algorithm.

The first three of these algoriths have recently been changed after extensive use in the operational network. In each case I'll sketch what was done before and then describe the current practice. The fourth algorithm will probably change in the near future. In this case I'll describe the current algorithm, its features and failings, and the proposed replacement algorithm.


← transmitting IMP
receiving IMP →
used/unused = Ø
trans O/E bit = Ø
(packet to be sent)
used/unused ← 1
O/E bit = Ø
packet
rec O/E bit = Ø
rec O/E bit ← 1
Ack=1
trans O/E bit ← 1
used/unused ← Ø
O/E bit = Ø
packet
Ack=1

(figure -- serial acks)

We now take a close look at the algorithm used for IMP to IMP transmission control. As we have already noted, the modem interface hardware has the capability of generating checksums for outgoing packets and checking the checksums on incoming packets. This allows packets which are incorrectly transmitted to be discarded without acknowledgment. Packets correctly received are acknowledged. A good IMP to IMP transmission control algorithm, besides detecting errors, acknowledging good transmissions, and providing retransmission in the event of errors, would be improved if it detected duplicates that are generated by retransmission. Our IMP to IMP transmission control algorithm has all of the above properties.

A number of logical "channels" are maintained between each pair of IMPs. Let's consider but one channel to begin and let's further consider packet transmissions only one direction on this channel. Of course, acknowledgments go the other direction on the channel. At both the transmit and receive end of this channel a one bit sequence number is kept. We call this bit an odd/even bit, represented as "o/e" in the figure. Both the transmit and receive odd/even bits are initialized to zero. Also, at the transmit end, a used/unused bit is kept for the channel. It is of course initialized to zero meaning unused. When it is time to transmit a packet, a check is first made for the channel being unused. If it was previously unused, it is marked used and the packet is transmitted. The

state of the transmit odd/even bit is included with the packet. When the packet arrives at the receiver, assuming the packet is received correctly, the packet's odd/even bit is checked against the receive odd/even bit. If they match, the packet is accepted and the receive odd/even bit is complemented. Otherwise the packet would be ignored. In any case the receive odd/even bit is returned as an acknowledgment. At the transmitter, if the acknowledgment bit does not match the transmit odd/even bit, the packet has been successfully sent and acknowledged and the packet is discarded, the channel is marked unused, and the transmit odd/even bit is complemented. Otherwise the acknowledgment is a duplicate and is ignored. Suppose now a second copy of the packet arrived at the receiver, a packet which was sent before the first acknowledgment had a chance to get back to the transmitter. When this packet arrives at the receiver, its odd/even bit does not match the receive odd/even bit and so that packet is discarded as a duplicate. Nonetheless, an acknowledgment is sent for the packet using the present state of the receive odd/even bit. When the acknowledgment gets to the transmitter, it does match the transmit odd/even bit, so the acknowledgment is a duplicate and is ignored.

The only subtlety in this algorithm is that the acknowledgment bit is the state of the receive odd/even bit after it is perhaps complemented rather than before it is perhaps complemented. Hence the need for the "not match" rule when the acknowledgment arrives at the

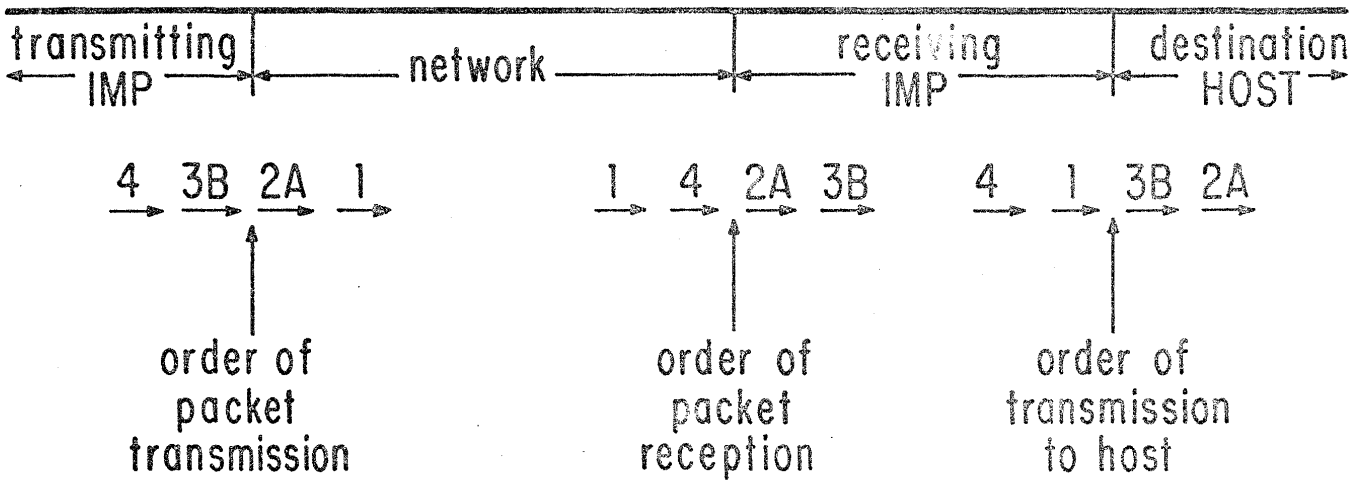
The above algorithm will be familiar to the representatives of the National Physical Laboratory.

Because of the potentially long distances between IMPs, one channel is not enough to keep the inter-IMP lines fully loaded. Therefore, we have chosen to supply eight logical channels between each pair. It is not necessary to maintain ordering of IMP to IMP transmissions since packet ordering is performed at the destination IMP. This means that the transmit channels can be filled in any convenient order, and at the receive side, packets can be forwarded through the network as soon as they are correctly received regardless of the channel over which they arrived.

To avoid requiring separate packets for acknowledgments, something we did until recently, acknowledge bits are "piggy-backed" in packets going the other way on the line. In fact, all eight receive odd/even bits are transmitted with every packet going the other way. In the absence of any traffic going the other way on the line, a packet carrying only the eight acknowledgments is sent. This results in acknowledgments getting back to the transmitter very fast. Therefore, the transmitter knows very soon whether a packet requires retransmission or not, and (especially) in the absence of other traffic can retransmit the packet immediately without waiting for any timeout period. This is also in contrast to our practice up until recently. We used to wait over 100 milliseconds for an acknowledgment before we decided the acknowledgment would not be forthcoming and retransmitted the packet. Our new system of packing eight acknowledgments into

every packet going the other way and retransmitting without waiting for a timeout period, saves both program and line bandwidth, lessens packet delay, and eases our buffering problems.

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. We do not want traffic which is essentially interactive to be subjected to waiting for perhaps several thousand-bit packets to be transmitted, multiplying by ten or more the effective delay seen by the source. We therefore, have instituted the following transmission ordering scheme: first we send new priority packets, then any new non-priority packets, and then if there are no new packets to send, we retransmit previously unacknowledged packets.


transmitting IMP
network
receiving IMP
destination HOST
4 3B 2A 1
order of packet transmission
1 4 2A 3B
order of packet reception
4 1 3B 2A
order of transmission to host

(figure -- message ordering)

We move now to our Host to Host transmission control algorithm. We introduced the technique of breaking Host messages into packets to minimize the delay seen for long transmission over many hops. We also allow several messages to simultaneously be in transit between a given pair of Hosts. However, the several messages and the packets within the messages may arrive at the destination IMP out of order, and in the event of a broken IMP or line, there may be duplicates. The task of our Host to Host transmission control algorithm is to reorder packets and messages at their destination, to cull duplicates, and after all the packets of a message have arrived, pass the message on to the destination Host and return a Ready For Next Message to the source. Until recently sequential message numbers were assigned to each message over each link -- by over a given link, I mean all of the messages having a common number in the link field of their leader. This message number was used to detect and discard packets from messages other than the current one based on the rule that on each link between a pair of Hosts, only one message may be in transmission.

We recently changed the above strategy in two ways.

First, it should be possible to have more than one message in transit between a pair of processes. In the past a Host could use more than one link to achieve this effect by "spraying" the tranmissions from one process on many links. However, it seemed that this was not the correct way

to use links. They should be used for (and the Host/Host protocol uses them for) multiplexing connections to the various processes in a Host. The IMP did not control the number of links in use, except to set an upper bound, but this lack of control led to congestion problems which I'll come to shortly. Therefore, we decided that the old function of the message number would be expanded to include the function of ordering Host to Host transmissions.

Specifically, we now allow up to four messages to be in transmission from a given source IMP to a given destination IMP. All of the source and destination Hosts share this message space. There is a message number assigned to each message at the source and the destination has a window of four acceptable message numbers out of a message-number space of 256. Messages with out-of-range message numbers are discarded, as well as duplicate messages and duplicate packets. Ready For Next Messages are still returned for each message successfully received. The Hosts know nothing of these message numbers: they are used internally to the IMP subnetwork to order messages into the destination Host.

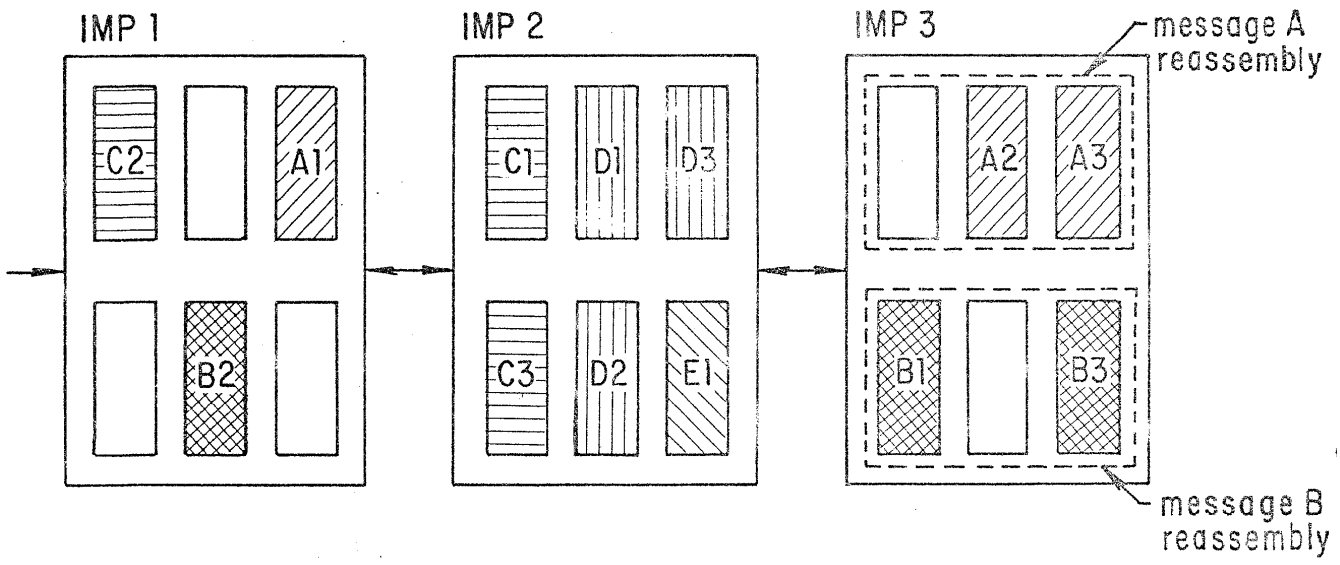
Second, we also wished to allow for a priority path between Hosts, which was able to bypass the regular message ordering scheme. That is, there should be a second path between Hosts in which messages can flow independent of the regular path, and when the next priority message is ready, or the next regular message is ready, the appropriate message is given to the destination Host. We have implemented this by

adding a two-bit priority-ordering number to the message number. Thus the message number serves the functions of detecting duplicates for all messages and ordering regular messages; the order number orders priority messages.

For example, if we use the letters A, B, C, and D to denote the 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 1, then priority messages 2A and 3B, and the non-priority message 4. Suppose the destination IMP receives these messages in the order 3B, 2A, 4, 1. It is accepting message numbers 1 through 4, and message number 1 and order number A are the next in sequence to be passed on to the Host. Therefore, it passes these messages to the Host in the order 2A, 3B, 1, 4. Note that message number 1 could have been sent to the destination Host first if it had arrived at the destination first, but that the priority messages are allowed to "leapfrog" ahead of message number 1 in the event it is delayed in the network. Note also that the IMP holds 3B until 2A arrives as the Host must receive priority message A before it receives priority message B. Likewise message 1 must be passed to the Host before message 4.

In addition to the window of acceptable message numbers that the source and destination IMPs maintain, there is a set of bits corresponding to outstanding messages. The source IMP keeps track of whether a response has come in

in the form of a Ready For Next Message typically for each message sent in order to detect duplicates. The destination IMP keeps track of whether the message is complete (that is, whether all the packets have arrived) in order to detect duplicate transmissions. The source IMP also times out the message number, and if a response has not been received for a message within 30 seconds, the source IMP sends out a control message with the timed-out message number, questioning the possibility of an incomplete transmission. The destination IMP must always return a Ready For Next Message for such a control message stating whether it saw the original message or not, and the source IMP will send the message number question every 30 seconds until it receives a response. This technique allows the source and destination IMPs to be synchronized in the event of a lost message or Ready For Next Message. It should be noted that this kind of failure is very infrequent, and happens only when an intermediate IMP fails and in doing so destroys a message.


IMP 1
IMP 2
IMP 3
C2
A1
B2
C1
D1
D3
C3
D2
E1
A2
A3
B1
B3
message A reassembly
message B reassembly

(figure -- reassembly lockup)

And now on to flow control and lockup prevention.

Until recently the IMPs placed no restriction on the number of messages that could simultaneously be in transit between a pair of Hosts other than that only one message could be outstanding on a given link at any time. We have long known that this is inadequate protection against network congestion and even network lockup. For instance, even with the four message window described above, reassembly storage at the destination can be completely used up by partially reassembled messages from several Hosts, and neighboring IMPs can fill with store-and-forward packets for the destination IMP. Once this kind of congestion has developed, a lockup which we call reassembly lockup can easily occur when the missing packets for the messages being reassembled are held two or more hops away from the destination.

We have developed and recently installed a method of controlling such congestion which is based on allocation messages sent from the destination to the source. When an IMP has a multi-packet message to send, it first sends off a "request for allocation" of reassembly space to the destination. Some time later it will receive an "allocate" message which means the destination IMP has reserved space in which to reassemble the multi-packet message and then the source IMP can send the multi-packet message. This procedure ensures that the destination is never swamped and that reassembly lockup will not occur.

The request/allocate sequence does introduce a certain amount of overhead, however, for multi-packet messages. Since we wish to provide as much bandwidth as possible for multi-packet messages, we provide a mechanism such that there is no necessity for the "request for allocation" in the case of later messages in a steady stream of traffic. When the destination IMP has given a multi-packet message to its Host, it returns a Ready For Next Message to the source and at the same time allocates reassembly storage for the anticipated next message. The source IMP receives, in effect, a new allocate with the Ready For Next Message and if the source Host sends another message to the destination within 125 milliseconds, the message can be transmitted without waiting for an allocate. If the source Host waits too long, the source IMP will return the allocation with a "give back" message. After this, the next time the Host tries to send, the IMP will transmit a "request for allocation", and wait for an "allocate" before proceeding.

For single packet messages, we are interested in minimizing the delay encountered through the network. The request mechanism used for multi-packet messages would slow down a one packet message too much. Instead, we send the one packet messages along with their "request for allocation," and save a copy of the message in the source IMP. If the destination IMP can take the message, it does so imediately and returns a Ready For Next Message to the source. If there is not enough storage at the destination, it sends back an "allocate" message when the storage becomes

available. When the source IMP receives this allocate, it retransmits the message (without the request indication this time). In this approach, Ready for Next Messages are passed along to the source Host as before, but requests and allocates are internal to the IMP subnetwork.

- adaptive
- low delay for interactive traffic
- avoidance of congestion
- does not "bang"
- load splitting
- maximizes global flow
- fair

(figure -- properties of a routing algorithm)

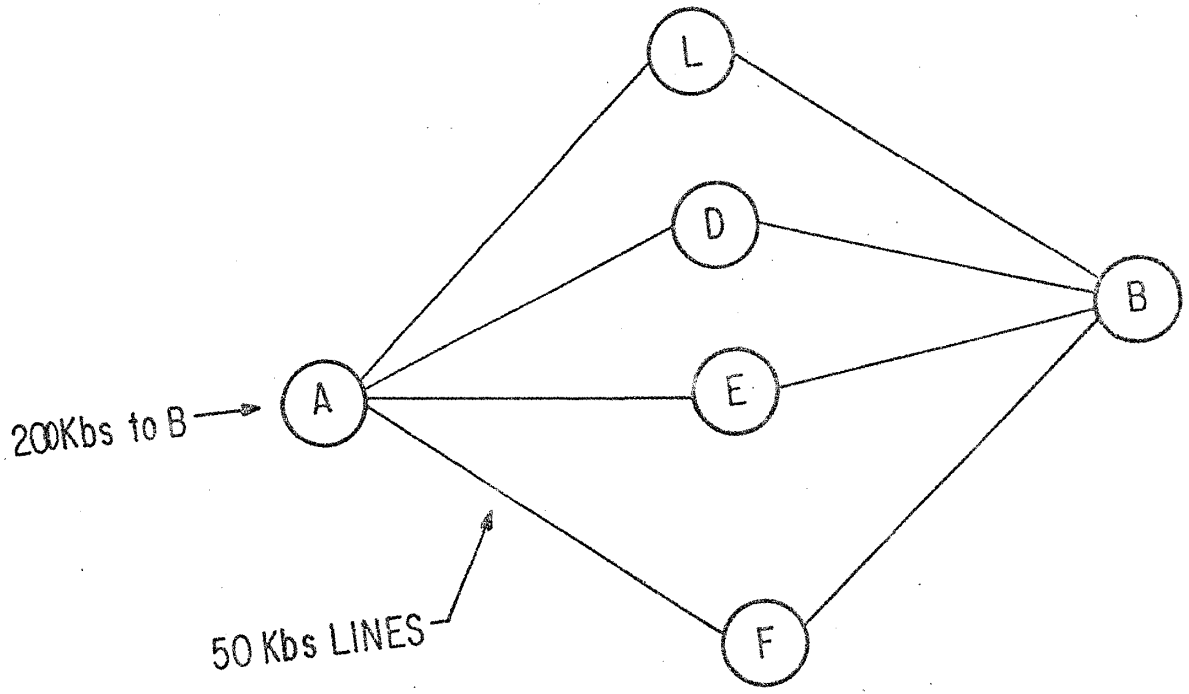
Finally, let's discuss the IMP's routing algorithm. First, I'll list the properties we believe a good routing algorithm should have.

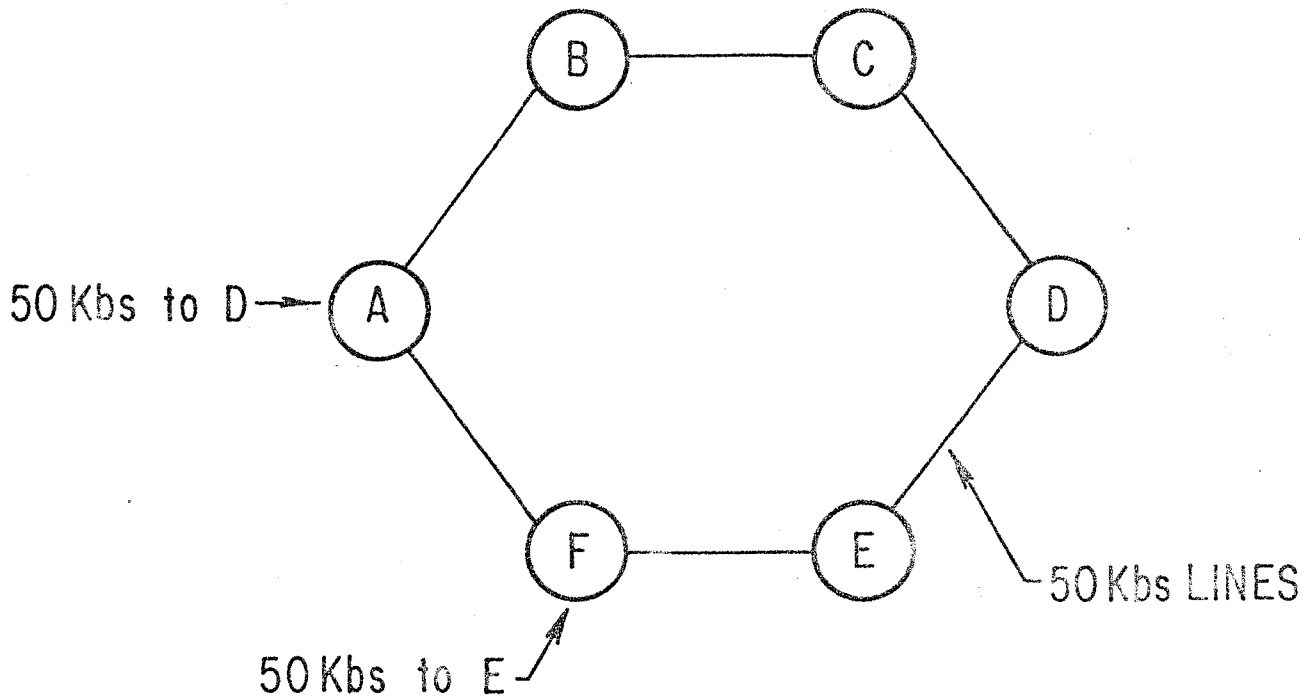
A good routing algorithm should be adaptive to the ups and downs of IMPs and lines. That is, traffic should be routed around broken lines and IMPs as long as any path exists to the desired destination. This may even require that a packet be returned over a path it has already been over. For instance, consider a packet going the short way around a circle from its source to its destination when a line ahead breaks. The packet should turn around and go back the other way around the circle. Incidentally, if a routing algorithm is adaptive to the ups and downs of lines and IMPs, it becomes very easy to change lines and add IMPs without any change to the routing algorithm.

A good routing algorithm should move interactive traffic through the network with minimal delay. This suggests interactive traffic should be routed via the shortest path.

A good routing algorithm avoids the creation of congestion or congestion itself by routing traffic around the congestion or potential congestion.

A good routing algorithm probably should not "bang." By "banging" we mean the abrupt shifting of traffic routes. We don't have theoretical evidence that banging is bad but most of us feel that banging is bad. It must be better


L
D
B
200Kbs to B
A
E
50 Kbs LINES
F


B
C
50 Kbs to D
A
D
F
E
50 Kbs LINES
50 Kbs to E

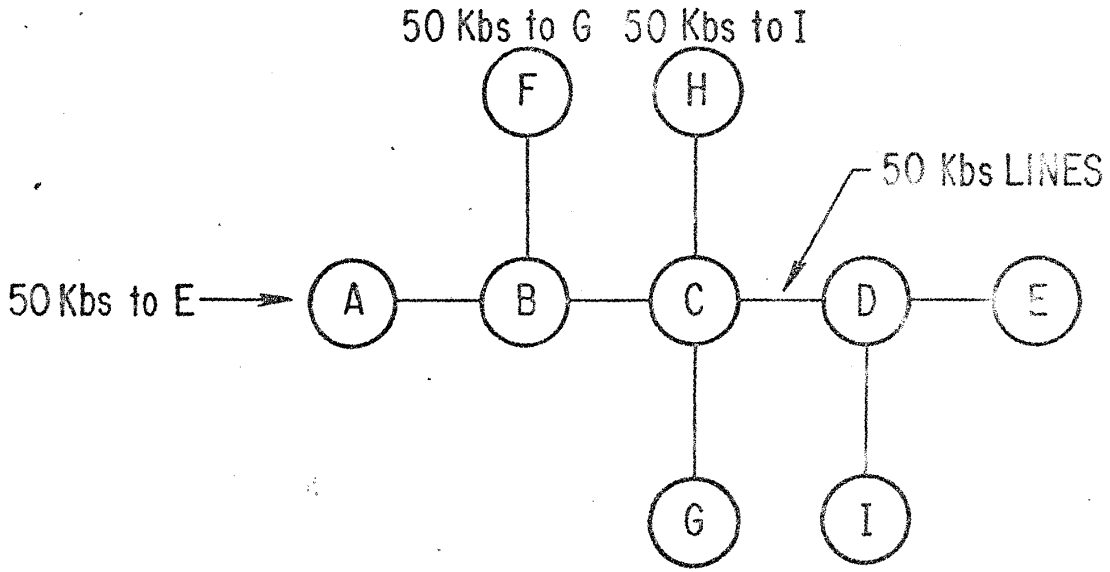
to change things smoothly.

(figure -- load splitting)

A good routing algorithm allows what we call load splitting. Suppose Host A has 200 kilobits of traffic per second for Host B. Note that there are four 50 kilobit paths between Host A and Host B. Load splitting means that the desired 200 kilobits can simultaneously be sent on all four lines.

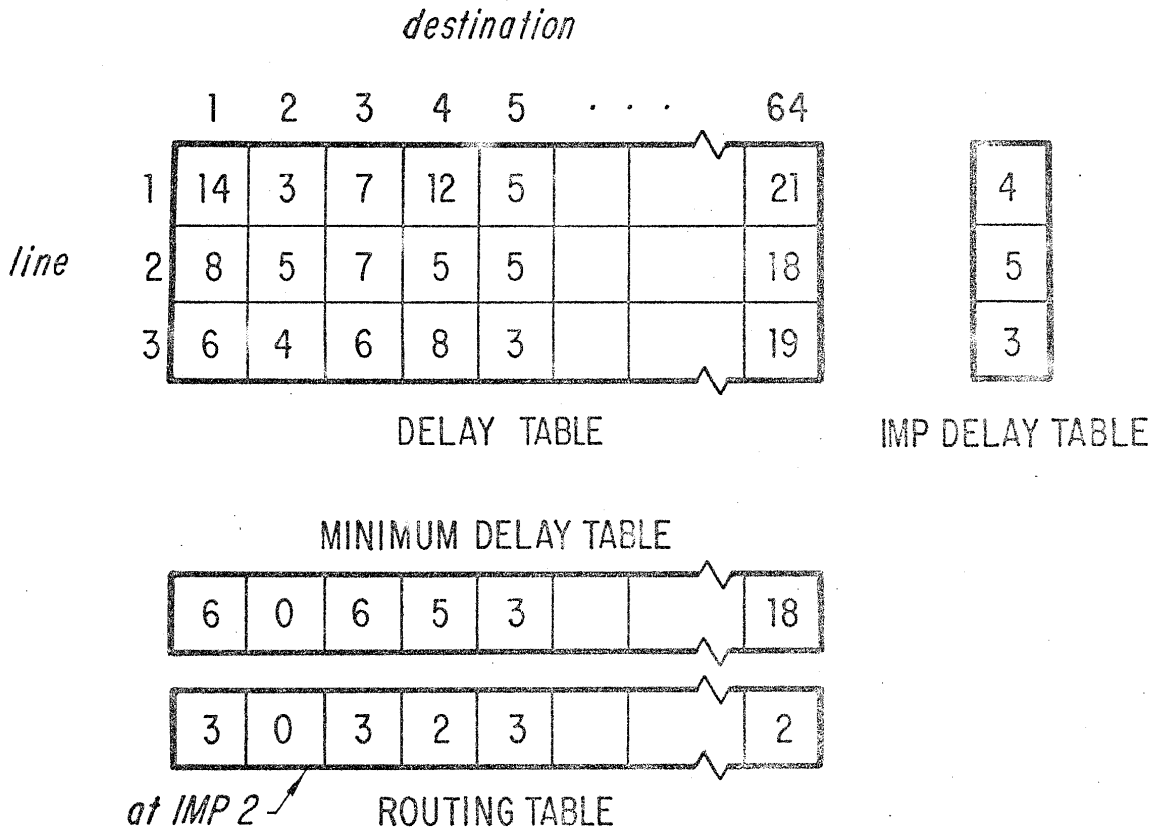
(figure -- global maximal flow)

A good routing algorithm routes high bandwidth tfaffic to achieve the maximal global flow. Suppose it is desired to send 50 kilobits of traffic from node A to node D and the same time it is desired to send 50 kilobits from node F to node E. If the traffic from F to E as well as the traffic from A to D must pass over link FE, the global flow is 50 kilobits. If however, either the traffic from A to D or the traffic from F to E were to go around the other way via link BC, the global flow would be 100 kilobits which is the maximal global flow under the desired traffic loading. Of course, it is clearly best for the A to D traffic to go via link BC since that also minimizes the global delay. However, we consider this a second order affect since we are concerned here with only the high bandwidth traffic. Interactive traffic has probably been sent via the shortest path.


50 Kbs to G
50 Kbs to I
F
H
50 Kbs LINES
50 Kbs to E
A
B
C
D
E
G
I

(figure -- fairness)

A good routing algorithm should be fair. Consider the figure in which it is desired to send 50 kilobits of traffic from A to E, from F to G, and from H to I. Clearly the maximal global flow is attained under this desired loading by stopping all traffic from A to E since every bit of traffic from A to E reduces the global flow by the same amount. But this is unfair. Fairness demands that A be able to get some traffic to E, even if it decreases the global flow. We don't really have a solid definition of "fair." It might be fairest to give all nodes an equal share of the available bandwidth. It might be fair to give bandwidth to each user in proportion to the amount of traffic they want to send, but this lets the big users drive the little users out. My inclination is to call it fair if no one is cut out entirely.


destination
1 2 3 4 5 . . . 64
line
1 14 3 7 12 5 21
2 8 5 7 5 5 18
3 6 4 6 8 3 19
DELAY TABLE
4
5
3
IMP DELAY TABLE
MINIMUM DELAY TABLE
6 0 6 5 3 18
3 0 3 2 3 2
at IMP 2
ROUTING TABLE

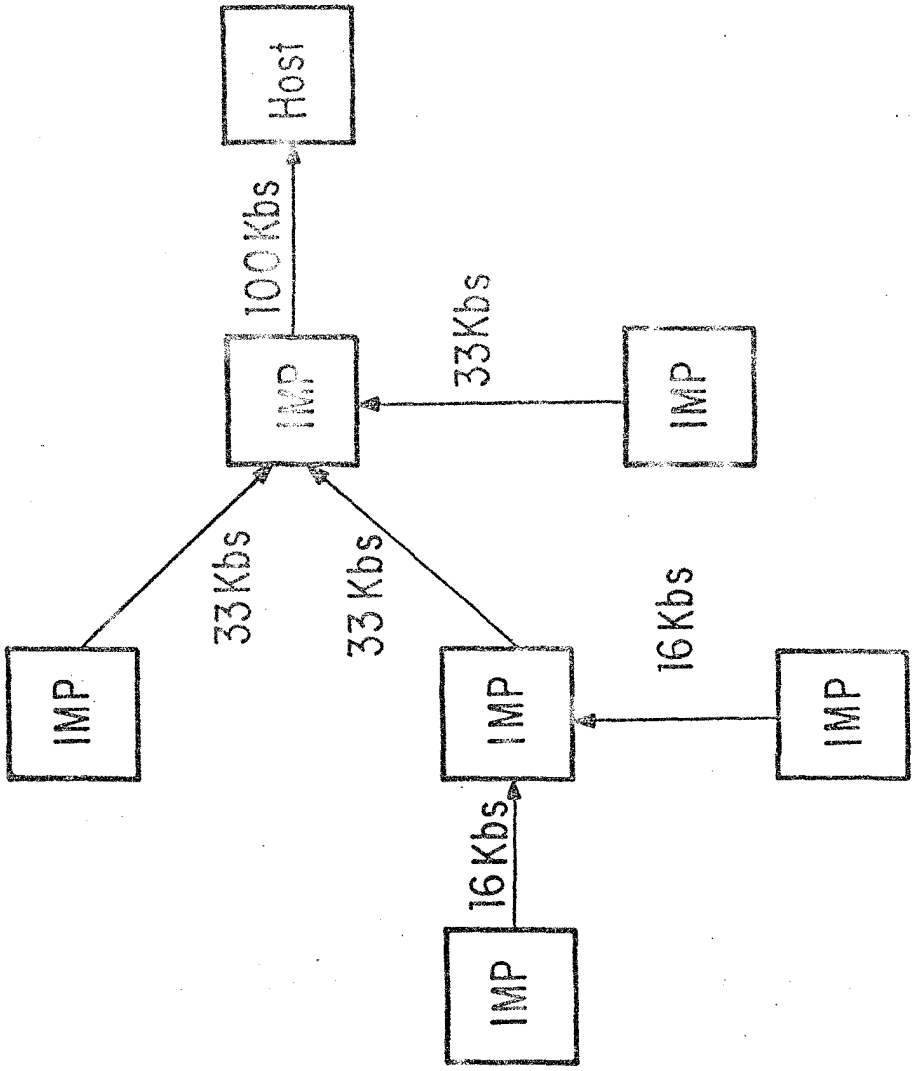
(figure -- three routing tables / current)

Now I will tell you about our current 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. This routing table is updated every half second as follows:

Each IMP maintains a table which gives an estimate of the delay it expects a packet to encounter in reaching every possible destination over each of its output lines. Thus the delay to destination 4 using line 2 is found in this entry. Periodically, every half second, the IMP selects the minimum delay for each destination and puts it in this table, the minimum delay table. It also notes the line giving the minimum delay and puts the number of the line in this table. This is the routing table and to route a packet to a particular destination, the correct line to send the packet out on is merely looked up in this table. Also every half second, the IMP passes its minimum delay table to each of its immediate neighbors; that is, it sends the minimum delay table out each of its phone lines. Of course, before the minimum delay table is transmitted to the neighboring IMPs, the IMP sets the minimum delay to itself to zero.

Since all of the neighbors of an IMP are also sending out their minimum delay table every half second, with their own entry set to zero, an IMP receives a minimum delay table from each of its neighbors every half second. These tables are read in over the rows of the delay table as they arrive. The row to be written over is the row corresponding to the phone line that the arriving minimum delay table came in over. After all the neighbors' estimates have arrived, the IMP adds to the neighbors' estimates the delay the IMP itself adds. This is done by adding the IMP delay table to each column of the delay table. Each entry in the IMP delay table is formed by adding the constant four to the number of packets on the output queue for the line corresponding to the table entry. The formation of the constant 4 is interesting. It includes 1 for the packet currently being output and thus no longer on the output queue. It includes 1 for the speed of light to get the packet from one end of the line to the other -- this is obviously a very gross approximatinn. It includes 1 because we felt like it. And it includes 1 because there is a bug in the program which has never been fixed. The effect of these last 2 is to bias the routing towards the shortest path. Theoreticians have told us that this bias prevents oscillations that would otherwise plague out routing algorithm. The fact is that in practice it does not oscillate.

This routing algorithm scores quite well on the properties we stated earlier that a good routing algorithm should have. It is adaptive to the ups and downs of lines and IMPs. Because of its strong bias toward the shortest path it tends to get interactive traffic quickly to its destination. It load splits over up to two lines. It is fair in the sense that it puts no limits on any source. Best of all, it is quite simple. In particular, it does not require the IMP to know the topology of the network. The IMP does not even need to know its neighbors' identities. Under existing loads in the ARPA network our current routing algorithm works very, very well. However, we must look to the future when loads are greater.



[I realize this figure is misleading - DCW]

(figure -- division of excess capacity)

We are presently thinking about changing the routing algorithm in the hope getting better global flow maximization, better load splitting, and better avoidance of congestion. Our thoughts follow these lines: the present routing algorithm has three faults: 1) packets are routed out one line or another over a given half second period, not a combination of lines. This leads to poor load splitting and some banging. 2) currently no attempt is made to quench flow in the face of congestion. The routing algorithm merely tries to route all traffic presented to it. 3) the current routing algorithm is too strongly biased towards the shortest path. This sometimes leads to congestion as packets can be routed directly into congestion if the congestion is on the shortest path and this does not lead to maximal global flow since that sometimes requires some packets to be routed far out of their way.

Fault 1 is quite easy to cure. A table of possible lines will be kept for each destination. Based on other considerations, the IMP will be able to route packets on one or all of the possible lines.

Fault 3 will be cured by no longer propagating least delay about the network. Instead, we will propagate available capacity about the network. Suppose for instance, a given Host can receive 100 kilobits per second. This 100 kilobits will be divided evenly and one share passed to each neighboring IMP. Each neighboring IMP will in turn divide the available capacity it was passed and pass some

to each of its neighbors. This will continue across the network.

Each time an IMP receives a packet, before the IMP acknowledges the packet, it will check if it has in its tables enough available capacity to accept the packet. If not, the packet is not acknowledged and is discarded. The fact that the IMP did not have enough capacity in its table for the packet is indicative that the IMP and line capacity ahead on this path to the destination is insufficient. If there is enough capacity, the packet is accepted and routed along the path of maximum available capacity and the available capacity is updated to show that some has been used. Of course, there will generally be traffic for several destination at a given IMP and therefore the available capacity passed on for a given destination must reflect that used for other destinations. This is just another way of saying that the sum of the capacity any node passes on can't be greater than the sum of its phone lines capacity and its buffering capacity. Although in the absence of traffic, excess capacity is passed out evenly to each neighboring IMP, once one neighbor begins to use capacity, the remaining capacity is again passed out evenly. This allows most of the capacity to be shifted to where it is needed. A little bit of capacity is always held for each other neighbor so that they can gain a foothold when they begin to need capacity. The shifting of excess capacity is severly smoothed. Thus traffic must push long and hard before it can capture a great deal of capacity. In other words, we are trying to optimize for the steady state

case. Still, we do let momentary transients through. However, they are noted and subsequent transients are prevented for a while. Notice that in curing fault 3 we have also cured fault 2. This new algorithm does quench flow.

One more point, we plan to use the above algorithm for non-interactive traffic. Interactive traffic will continue to be sent via the shortest path and the capacity it uses along this path noted.

You have now heard a summary of the design and operation of the IMP subnetwork. You have heard a description of the IMP software system. And you have bee led through the intricacies of the IMP's algorithms for transmission control, flow control and lockup prevention, and two routing algorithms. Surely that is enough for now. I will be glad to try to answer questions now or after the session.

Thank you.