NIC 50006

1

DEFENSE COMMUNICATIONS AGENCY

AD-A166 326

DDN PROTOCOL HANDBOOK

Volume Three

SUPPLEMENT

DECEMBER 1985

DDN

NCLAS[illegible]

SECURITY CLA[illegible]ON OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Distribution Statement A Approved for public release distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NIC 50004;NIC 50005; NIC 50006	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION SRI International DDN Network Information Ctr — 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Defense Data Network Program Management Office
6c. ADDRESS *(City, State, and ZIP Code)* Menlo Park, CA 94025	7b. ADDRESS *(City, State, and ZIP Code)* McLean, VA 22102
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. — PROJECT NO. — TASK NO. — WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
DDN Protocol Handbook (3 vols.) (Unclassified)

12. PERSONAL AUTHOR(S)
Feinler, E.; Jacobsen, O.; Stahl, M.; Ward, C., eds.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
	FROM ____ TO ____	851200	ca. 3000

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	Defense Data Network; DDN; DDN protocols; Network protocols; TCP/IP; Transmission Control Protocol/ Internet Protocol; Network subscriber access

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The DDN (Defense Data Network) Protocol Handbook is a three volume work which gathers together many documents of interest to those wishing to implement the Department of Defense suite of protocols on various computers to be attached to the DDN. The official Military Standard communication protocols in use on the DDN are included, as are several ARPANET (Advanced Research Projects Agency Network) research protocols which are currently in use, and some protocols currently undergoing review. Tutorial information and auxiliary documents are also included. In addition to its use as a sou guide for protocol implementation pruposes, the handbook can be used by vendors wishing to make their products compatible with DoD needs, by researchers wishing to improve the protocols, and by implementors of local area networks (LANs) wishing their networks to interact with the DDN.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL E. Redfield	22b. TELEPHONE *(Include Area Code)* (415) 859-6187 — 22c. OFFICE SYMBOL EJ 292

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

F. Redfield
SRI International
Network Information Center
333 Ravenswood Ave.
Menlo Park, CA 94025

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

DEFENSE COMMUNICATIONS AGENCY

DDN PROTOCOL HANDBOOK

Volume Three

SUPPLEMENT

DECEMBER 1985

Editors:

Elizabeth J. Feinler
Ole J. Jacobsen
Mary K. Stahl
Carol A. Ward

~~Additional copies of this document may be obtained from the DDN Network Information Center, SRI International, 333 Ravenswood Avenue, Room EJ291, Menlo~~ Park, CA 94025 or from the Defense Technical Information Center (DTIC), Cameron Station, Alexandria, VA 22314.

DDN Protocol Handbook. Third volume of three-volume set. Printed and bound in the United States of America. Published by the DDN Network Information Center, SRI International, Menlo Park, CA 94025.

Date: December 1985

ACKNOWLEDGEMENTS

The DDN Protocol Handbook was compiled by the DDN Network Information Center (NIC) for the Defense Data Network Program Management Office (DDN PMO) of the Defense Communications Agency (DCA) under contract number DCA-200-83-0025.

The editors are indebted to the authors of the many RFCs included in the body of this document. Special thanks goes to the following people for their invaluable support and contributions toward the production of the Handbook: Jonathan B. Postel from the University of Southern California Information Sciences Institute; Michael L. Corrigan, John Claitor, and John R. Walker from the DDN PMO; Edward Brady, Philip S. Selvaggi, Edward A. Cain from the Defense Communications Engineering Center; Chris J. Perry and Michael A. Padlipsky from the Mitre Corporation; and Diane Fountaine from the Office of the Assistant Secretary of Defense for Command, Control, Communications and Intelligence (C^3I).

TABLE OF CONTENTS - VOLUME THREE

SECTION 1. INTRODUCTION TO VOLUME 3

Volume Three of the 1985 DDN Protocol Handbook, a three-volume set, contains implementation guidelines and several auxiliary documents of use to protocol implementors in both the DoD and DARPA internet communities. Volumes One and Two contain the actual protocols as well as details about DoD and ARPANET Protocol review and acceptance policies. Volume Three should be used in conjunction with either or both of the other two volumes.

The price for the three-volume set is $110.00, prepaid, to cover the cost of reproduction and handling. Checks should be made payable to SRI International. Copies of the handbook will also be deposited at DTIC.

Additional copies of the 1985 DDN Protocol Handbook can be ordered from:

DDN Network Information Center
SRI International, Room EJ291
333 Ravenswood Avenue
Menlo Park, CA 94025
Telephone: (800) 235-3155

SECTION 2. PROTOCOL IMPLEMENTATION GUIDELINES

RFC: 813

WINDOW AND ACKNOWLEDGEMENT STRATEGY IN TCP

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. Introduction

This document describes implementation strategies to deal with two mechanisms in TCP, the window and the acknowledgement. These mechanisms are described in the specification document, but it is possible, while complying with the specification, to produce implementations which yield very bad performance. Happily, the pitfalls possible in window and acknowledgement strategies are very easy to avoid.

It is a much more difficult exercise to verify the performance of a specification than the correctness. Certainly, we have less experience in this area, and we certainly lack any useful formal technique. Nonetheless, it is important to attempt a specification in this area, because different implementors might otherwise choose superficially reasonable algorithms which interact poorly with each other. This document presents a particular set of algorithms which have received testing in the field, and which appear to work properly with each other. With more experience, these algorithms may become part of the formal specification: until such time their use is recommended.

2. The Mechanisms

The acknowledgement mechanism is at the heart of TCP. Very simply, when data arrives at the recipient, the protocol requires that it send back an acknowledgement of this data. The protocol specifies that the bytes of data are sequentially numbered, so that the recipient can acknowledge data by naming the highest numbered byte of data it has received, which also acknowledges the previous bytes (actually, it identifies the first byte of data which it has not yet received, but this is a small detail). The protocol contains only a general assertion that data should be acknowledged promptly, but gives no more specific indication as to how quickly an acknowledgement must be sent, or how much data should be acknowledged in each separate acknowledgement.

The window mechanism is a flow control tool. Whenever appropriate, the recipient of data returns to the sender a number, which is (more or less) the size of the buffer which the receiver currently has available for additional data. This number of bytes, called the window, is the maximum which the sender is permitted to transmit until the receiver returns some additional window. Sometimes, the receiver will have no buffer space available, and will return a window value of zero. Under these circumstances, the protocol requires the sender to send a small segment to the receiver now and then, to see if more data is accepted. If the window remains closed at zero for some substantial period, and the sender can obtain no response from the receiver, the protocol requires the sender to conclude that the receiver has failed, and to close the connection. Again, there is very little performance information in the specification, describing under what circumstances the window should be increased, and how the sender should respond to such revised information.

A bad implementation of the window algorithm can lead to extremely poor performance overall. The degradations which occur in throughput and CPU utilizations can easily be several factors of ten, not just a fractional increase. This particular phenomenon is specific enough that it has been given the name of Silly Window Syndrome, or SWS. Happily SWS is easy to avoid if a few simple rules are observed. The most important function of this memo is to describe SWS, so that implementors will understand the general nature of the problem, and to describe algorithms which will prevent its occurrence. This document also describes performance enhancing algorithms which relate to acknowledgement, and discusses the way acknowledgement and window algorithms interact as part of SWS.

3. SILLY WINDOW SYNDROME

In order to understand SWS, we must first define two new terms. Superficially, the window mechanism is very simple: there is a number, called "the window", which is returned from the receiver to the sender. However, we must have a more detailed way of talking about the meaning of this number. The receiver of data computes a value which we will call the "offered window". In a simple case, the offered window corresponds to the amount of buffer space available in the receiver. This correspondence is not necessarily exact, but is a suitable model for the discussion to follow. It is the offered window which is actually transmitted back from the receiver to the sender. The sender uses the offered window to compute a different value, the "usable window", which is the offered window minus the amount of outstanding unacknowledged data. The usable window is less than or equal to the offered window, and can be much smaller.

Consider the following simple example. The receiver initially provides an offered window of 1,000. The sender uses up this window by sending five segments of 200 bytes each. The receiver, on processing the first of these segments, returns an acknowledgement which also contains an updated window value. Let us assume that the receiver of the data has removed the first 200 bytes from the buffer, so that the receiver once again has 1,000 bytes of available buffer. Therefore, the receiver would return, as before, an offered window of 1,000 bytes. The sender, on receipt of this first acknowledgement, now computes the additional number of bytes which may be sent. In fact, of the 1,000 bytes which the recipient is prepared to receive at this time, 800 are already in transit, having been sent in response to the previous offered window. In this case, the usable window is only 200 bytes.

Let us now consider how SWS arises. To continue the previous example, assume that at some point, when the sender computes a useable window of 200 bytes, it has only 50 bytes to send until it reaches a "push" point. It thus sends 50 bytes in one segment, and 150 bytes in the next segment. Sometime later, this 50-byte segment will arrive at the recipient, which will process and remove the 50 bytes and once again return an offered window of 1,000 bytes. However, the sender will now compute that there are 950 bytes in transit in the network, so that the useable window is now only 50 bytes. Thus, the sender will once again send a 50 byte segment, even though there is no longer a natural boundary to force it.

In fact, whenever the acknowledgement of a small segment comes back, the useable window associated with that acknowledgement will cause another segment of the same small size to be sent, until some abnormality breaks the pattern. It is easy to see how small segments arise, because natural boundaries in the data occasionally cause the sender to take a computed useable window and divide it up between two segments. Once that division has occurred, there is no natural way for those useable window allocations to be recombined; thus the breaking up of the useable window into small pieces will persist.

Thus, SWS is a degeneration in the throughput which develops over time, during a long data transfer. If the sender ever stops, as for example when it runs out of data to send, the receiver will eventually acknowledge all the outstanding data, so that the useable window computed by the sender will equal the full offered window of the receiver. At this point the situation will have healed, and further data transmission over the link will occur efficiently. However, in large file transfers, which occur without interruption, SWS can cause appalling performance. The network between the sender and the receiver becomes clogged with many small segments, and an equal number of acknowledgements, which in turn causes lost segments, which triggers massive retransmission. Bad cases of SWS have been seen in which the average segment size was one-tenth of the size the sender and receiver were prepared to deal with, and the average number of retransmission per successful segments sent was five.

Happily, SWS is trivial to avoid. The following sections describe two algorithms, one executed by the sender, and one by the receiver, which appear to eliminate SWS completely. Actually, either algorithm by itself is sufficient to prevent SWS, and thus protect a host from a foreign implementation which has failed to deal properly with this problem. The two algorithms taken together produce an additional reduction in CPU consumption, observed in practice to be as high as a factor of four.

4. Improved Window Algorithms

The receiver of data can take a very simple step to eliminate SWS. When it disposes of a small amount of data, it can artificially reduce the offered window in subsequent acknowledgements, so that the useable window computed by the sender does not permit the sending of any further data. At some later time, when the receiver has processed a substantially larger amount of incoming data, the artificial limitation on the offered window can be removed all at once, so that the sender computes a sudden large jump rather than a sequence of small jumps in the useable window.

At this level, the algorithm is quite simple, but in order to determine exactly when the window should be opened up again, it is necessary to look at some of the other details of the implementation.

Depending on whether the window is held artificially closed for a short or long time, two problems will develop. The one we have already discussed -- never closing the window artificially -- will lead to SWS. On the other hand, if the window is only opened infrequently, the pipeline of data in the network between the sender and the receiver may have emptied out while the sender was being held off, so that a delay is introduced before additional data arrives from the sender. This delay does reduce throughput, but it does not consume network resources or CPU resources in the process, as does SWS. Thus, it is in this direction that one ought to overcompensate. For a simple implementation, a rule of thumb that seems to work in practice is to artificially reduce the offered window until the reduction constitutes one half of the available space, at which point increase the window to advertise the entire space again. In any event, one ought to make the chunk by which the window is opened at least permit one reasonably large segment. (If the receiver is so short of buffers that it can never advertise a large enough buffer to permit at least one large segment, it is hopeless to expect any sort of high throughput.)

There is an algorithm that the sender can use to achieve the same effect described above: a very simple and elegant rule first described by Michael Greenwald at MIT. The sender of the data uses the offered window to compute a useable window, and then compares the useable window to the offered window, and refrains from sending anything if the ratio of useable to offered is less than a certain fraction. Clearly, if the computed useable window is small compared to the offered window, this means that a substantial amount of previously sent information is still in the pipeline from the sender to the receiver, which in turn means that the sender can count on being granted a larger useable window in the future. Until the useable window reaches a certain amount, the sender should simply refuse to send anything.

Simple experiments suggest that the exact value of the ratio is not very important, but that a value of about 25 percent is sufficient to avoid SWS and achieve reasonable throughput, even for machines with a small offered window. An additional enhancement which might help throughput would be to attempt to hold off sending until one can send a maximum size segment. Another enhancement would be to send anyway, even if the ratio is small, if the useable window is sufficient to hold the data available up to the next "push point".

This algorithm at the sender end is very simple. Notice that it is not necessary to set a timer to protect against protocol lockup when postponing the send operation. Further acknowledgements, as they arrive, will inevitably change the ratio of offered to useable window. (To see this, note that when all the data in the catanet pipeline has arrived at the receiver, the resulting acknowledgement must yield an offered window and useable window that equal each other.) If the expected acknowledgements do not arrive, the retransmission mechanism will come into play to assure that something finally happens. Thus, to add this algorithm to an existing TCP implementation usually requires one line of code. As part of the send algorithm it is already necessary to compute the useable window from the offered window. It is a simple matter to add a line of code which, if the ratio is less than a certain percent, sets the useable window to zero. The results of SWS are so devastating that no sender should be without this simple piece of insurance.

5. Improved Acknowledgement Algorithms

In the beginning of this paper, an overly simplistic implementation of TCP was described, which led to SWS. One of the characteristics of this implementation was that the recipient of data sent a separate acknowledgement for every segment that it received. This compulsive acknowledgement was one of the causes of SWS, because each acknowledgement provided some new useable window, but even if one of the algorithms described above is used to eliminate SWS, overly frequent acknowledgement still has a substantial problem, which is that it greatly increases the processing time at the sender's end. Measurement of TCP implementations, especially on large operating systems, indicate that most of the overhead of dealing with a segment is not in the processing at the TCP or IP level, but simply in the scheduling of the handler which is required to deal with the segment. A steady dribble of acknowledgements causes a high overhead in scheduling, with very little to show for it. This waste is to be avoided if possible.

There are two reasons for prompt acknowledgement. One is to prevent retransmission. We will discuss later how to determine whether unnecessary retransmission is occurring. The other reason one acknowledges promptly is to permit further data to be sent. However, the previous section makes quite clear that it is not always desirable to send a little bit of data, even though the receiver may have room for it. Therefore, one can state a general rule that under normal operation, the receiver of data need not, and for efficiency reasons should not, acknowledge the data unless either the acknowledgement is intended to produce an increased useable window, is necessary in order to prevent retransmission or is being sent as part of a reverse direction segment being sent for some other reason. We will consider an algorithm to achieve these goals.

Only the recipient of the data can control the generation of acknowledgements. Once an acknowledgement has been sent from the receiver back to the sender, the sender must process it. Although the extra overhead is incurred at the sender's end, it is entirely under the receiver's control. Therefore, we must now describe an algorithm which occurs at the receiver's end. Obviously, the algorithm must have the following general form; sometimes the receiver of data, upon processing a segment, decides not to send an acknowledgement now, but to postpone the acknowledgement until some time in the future, perhaps by setting a timer. The peril of this approach is that on many large operating systems it is extremely costly to respond to a timer event, almost as costly as to respond to an incoming segment. Clearly, if the receiver of the data, in order to avoid extra overhead at the sender end, spends a great deal of time responding to timer interrupts, no overall benefit has been achieved, for efficiency at the sender end is achieved by great thrashing at the receiver end. We must find an algorithm that avoids both of these perils.

The following scheme seems a good compromise. The receiver of data will refrain from sending an acknowledgement under certain circumstances, in which case it must set a timer which will cause the acknowledgement to be sent later. However, the receiver should do this only where it is a reasonable guess that some other event will intervene and prevent the necessity of the timer interrupt. The most obvious event on which to depend is the arrival of another segment. So, if a segment arrives, postpone sending an acknowledgement if both of the following conditions hold. First, the push bit is not set in the segment, since it is a reasonable assumption that there is more data coming in a subsequent segment. Second, there is no revised window information to be sent back.

This algorithm will insure that the timer, although set, is seldom used. The interval of the timer is related to the expected inter-segment delay, which is in turn a function of the particular network through which the data is flowing. For the Arpanet, a reasonable interval seems to be 200 to 300 milliseconds. Appendix A describes an adaptive algorithm for measuring this delay.

The section on improved window algorithms described both a receiver algorithm and a sender algorithm, and suggested that both should be used. The reason for this is now clear. While the sender algorithm is extremely simple, and useful as insurance, the receiver algorithm is required in order that this improved acknowledgement strategy work. If the receipt of every segment causes a new window value to be returned, then of necessity an acknowledgement will be sent for every data segment. When, according to the strategy of the previous section, the receiver determines to artificially reduce the offered window, that is precisely the circumstance under which an acknowledgement need not be sent. When the receiver window algorithm and the receiver acknowledgement algorithm are used together, it will be seen that sending an acknowledgement will be triggered by one of the following events. First, a push bit has been received. Second, a temporary pause in the data stream is detected. Third, the offered window has been artificially reduced to one-half its actual value.

In the beginning of this section, it was pointed out that there are two reasons why one must acknowledge data. Our consideration at this point has been concerned only with the first, that an acknowledgement must be returned as part of triggering the sending of new data. It is also necessary to acknowledge whenever the failure to do so would trigger retransmission by the sender. Since the retransmission interval is selected by the sender, the receiver of the data cannot make a precise determination of when the acknowledgement must be sent. However, there is a rough rule the sender can use to avoid retransmission, provided that the receiver is reasonably well behaved.

We will assume that sender of the data uses the optional algorithm described in the TCP specification, in which the roundtrip delay is measured using an exponential decay smoothing algorithm. Retransmission of a segment occurs if the measured delay for that segment exceeds the smoothed average by some factor. To see how retransmission might be triggered, one must consider the pattern of segment arrivals at the receiver. The goal of our strategy was that the sender should send off a number of segments in close sequence, and receive one acknowledgement for the whole burst. The acknowledgement will be generated by the receiver at the time that the last segment in the burst arrives at the receiver. (To ensure the prompt return of the acknowledgement, the sender could turn on the "push" bit in the last segment of the burst.) The delay observed at the sender between the initial transmission of a segment and the receipt of the acknowledgement will include both the network transit time, plus the holding time at the receiver. The holding time will be greatest for the first segments in the burst, and smallest for the last segments in the burst. Thus, the smoothing algorithm will measure a delay which is roughly proportional to the average roundtrip delay for all the segments in the burst. Problems will arise if the average delay is substantially smaller than the maximum delay and the smoothing algorithm used has a very small threshold for triggering retransmission. The widest variation between average and maximum delay will occur when network transit time is negligible, and all delay is processing time. In this case, the maximum will be twice the average (by simple algebra) so the threshold that controls retransmission should be somewhat more than a factor of two.

In practice, retransmission of the first segments of a burst has not been a problem because the delay measured consists of the network roundtrip delay, as well as the delay due to withholding the acknowledgement, and the roundtrip tends to dominate except in very low roundtrip time situations (such as when sending to one's self for test purposes). This low roundtrip situation can be covered very simply by including a minimum value below which the roundtrip estimate is not permitted to drop.

In our experiments with this algorithm, retransmission due to faulty calculation of the roundtrip delay occurred only once, when the parameters of the exponential smoothing algorithm had been misadjusted so that they were only taking into account the last two or three segments sent. Clearly, this will cause trouble since the last two or three segments of any burst are the ones whose holding time at the receiver is minimal, so the resulting total estimate was much lower than appropriate. Once the parameters of the algorithm had been adjusted so that the number of segments taken into account was approximately twice the number of segments in a burst of average size, with a threshold factor of 1.5, no further retransmission has ever been identified due to this problem, including when sending to ourself and when sending over high delay nets.

6. Conservative Vs. Optimistic Windows

According to the TCP specification, the offered window is presumed to have some relationship to the amount of data which the receiver is actually prepared to receive. However, it is not necessarily an exact correspondence. We will use the term "conservative window" to describe the case where the offered window is precisely no larger than the actual buffering available. The drawback to conservative window algorithms is that they can produce very low throughput in long delay situations. It is easy to see that the maximum input of a conservative window algorithm is one bufferfull every roundtrip delay in the net, since the next bufferfull cannot be launched until the updated window/acknowledgement information from the previous transmission has made the roundtrip.

In certain cases, it may be possible to increase the overall throughput of the transmission by increasing the offered window over the actual buffer available at the receiver. Such a strategy we will call an "optimistic window" strategy. The optimistic strategy works if the network delivers the data to the recipient sufficiently slowly that it can process the data fast enough to keep the buffer from overflowing. If the receiver is faster than the sender, one could, with luck, permit an infinitely optimistic window, in which the sender is simply permitted to send full-speed. If the sender is faster than the receiver, however, and the window is too optimistic, then some segments will cause a buffer overflow, and will be discarded. Therefore, the correct strategy to implement an optimistic window is to increase the window size until segments start to be lost. This only works if it is possible to detect that the segment has been lost. In some cases, it is easy to do, because the segment is partially processed inside the receiving host before it is thrown away. In other cases, overflows may actually cause the network interface to be clogged, which will cause the segments to be lost elsewhere in the net. It is inadvisable to attempt an optimistic window strategy unless one is certain that the algorithm can detect the resulting lost segments. However, the increase in throughput which is possible from optimistic windows is quite substantial. Any systems with small buffer space should seriously consider the merit of optimistic windows.

The selection of an appropriate window algorithm is actually more complicated than even the above discussion suggests. The following considerations are not presented with the intention that they be incorporated in current implementations of TCP, but as background for the sophisticated designer who is attempting to understand how his TCP will respond to a variety of networks, with different speed and delay characteristics. The particular pattern of windows and acknowledgements sent from receiver to sender influences two characteristics of the data being sent. First, they control the average data rate. Clearly, the average rate of the sender cannot exceed the average rate of the receiver, or long-term buffer overflow will occur. Second, they influence the burstiness of the data coming from the sender. Burstiness has both advantages and disadvantages. The advantage of burstiness is that it reduces the CPU processing necessary to send the data. This follows from the observed fact, especially on large machines, that most of the cost of sending a segment is not the TCP or IP processing, but the scheduling overhead of getting started.

On the other hand, the disadvantage of burstiness is that it may cause buffers to overflow, either in the eventual recipient, which was discussed above, or in an intermediate gateway, a problem ignored in this paper. The algorithms described above attempts to strike a balance between excessive burstiness, which in the extreme cases can cause delays because a burst is not requested soon enough, and excessive fragmentation of the data stream into small segments, which we identified as Silly Window Syndrome.

Under conditions of extreme delay in the network, none of the algorithms described above will achieve adequate throughput. Conservative window algorithms have a predictable throughput limit, which is one windowfull per roundtrip delay. Attempts to solve this by optimistic window strategies may cause buffer overflows due to the bursty nature of the arriving data. A very sophisticated way to solve this is for the receiver, having measured by some means the roundtrip delay and intersegment arrival rate of the actual connection, to open his window, not in one optimistic increment of gigantic proportion, but in a number of smaller optimistic increments, which have been carefully spaced using a timer so that the resulting smaller bursts which arrive are each sufficiently small to fit into the existing buffers. One could visualize this as a number of requests flowing backwards through the net which trigger in return a number of bursts which flow back spaced evenly from the sender to the receiver. The overall result is that the receiver uses the window mechanism to control the burstiness of the arrivals, and the average rate.

To my knowledge, no such strategy has been implemented in any TCP. First, we do not normally have delays high enough to require this kind of treatment. Second, the strategy described above is probably not stable unless it is very carefully balanced. Just as buses on a single bus route tend to bunch up, bursts which start out equally spaced could well end up piling into each other, and forming the single large burst which the receiver was hoping to avoid. It is important to understand this extreme case, however, in order to understand the limits beyond which TCP, as normally implemented, with either conservative or simple optimistic windows can be expected to deliver throughput which is a reasonable percentage of the actual network capacity.

7. Conclusions

This paper describes three simple algorithms for performance enhancement in TCP, one at the sender end and two at the receiver. The sender algorithm is to refrain from sending if the useable window is smaller than 25 percent of the offered window. The receiver algorithms are first, to artificially reduce the offered window when processing new data if the resulting reduction does not represent more than some fraction, say 50 percent, of the actual space available, and second, to refrain from sending an acknowledgment at all if two simple conditions hold.

Either of these algorithms will prevent the worst aspects of Silly Window Syndrome, and when these algorithms are used together, they will produce substantial improvement in CPU utilization, by eliminating the process of excess acknowledgements.

Preliminary experiments with these algorithms suggest that they work, and work very well. Both the sender and receiver algorithms have been shown to eliminate SWS, even when talking to fairly silly algorithms at the other end. The Multics mailer, in particular, had suffered substantial attacks of SWS while sending large mail to a number of hosts. We believe that implementation of the sender side algorithm has eliminated every known case of SWS detected in our mailer. Implementation of the receiver side algorithm produced substantial improvements of CPU time when Multics was the sending system. Multics is a typical large operating system, with scheduling costs which are large compared to the actual processing time for protocol handlers.

Tests were done sending from Multics to a host which implemented the SWS suppression algorithm, and which could either refrain or not from sending acknowledgements on each segment. As predicted, suppressing the return acknowledgements did not influence the throughput for large data transfer at all, since the throttling effect was elsewhere. However, the CPU time required to process the data at the Multics end was cut by a factor of four (In this experiment, the bursts of data which were being sent were approximately eight segments. Thus, the number of acknowledgements in the two experiments differed by a factor of eight.)

An important consideration in evaluating these algorithms is that they must not cause the protocol implementations to deadlock. All of the recommendations in this document have the characteristic that they suggest one refrain from doing something even though the protocol specification permits one to do it. The possibility exists that if one refrains from doing something now one may never get to do it later, and both ends will halt, even though it would appear superficially that the transaction can continue.

Formally, the idea that things continue to work is referred to as "liveness". One of the defects of ad hoc solutions to performance problems is the possibility that two different approaches will interact to prevent liveness. It is believed that the algorithms described in this paper are always live, and that is one of the reasons why there is a strong advantage in uniform use of this particular proposal, except in cases where it is explicitly demonstrated not to work.

The argument for liveness in these solutions proceeds as follows.

First, the sender algorithm can only be stopped by one thing, a refusal of the receiver to acknowledge sent data. As long as the receiver continues to acknowledge data, the ratio of useable window to offered window will approach one, and eventually the sender must continue to send. However, notice that the receiver algorithm we have advocated involves refraining from acknowledging. Therefore, we certainly do have a situation where improper operation of this algorithm can prevent liveness.

What we must show is that the receiver of the data, if it chooses to refrain from acknowledging, will do so only for a short time, and not forever. The design of the algorithm described above was intended to achieve precisely this goal: whenever the receiver of data refrained from sending an acknowledgement it was required to set a timer. The only event that was permitted to clear that timer was the receipt of another segment, which essentially reset the timer, and started it going again. Thus, an acknowledgement will be sent as soon as no data has been received. This has precisely the effect desired: if the data flow appears to be disrupted for any reason, the receiver responds by sending an up-to-date acknowledgement. In fact, the receiver algorithm is designed to be more robust than this, for transmission of an acknowledgment is triggered by two events, either a cessation of data or a reduction in the amount of offered window to 50 percent of the actual value. This is the condition which will normally trigger the transmission of this acknowledgement.

APPENDIX A

Dynamic Calculation of Acknowledgement Delay

The text suggested that when setting a timer to postpone the sending of an acknowledgement, a fixed interval of 200 to 300 milliseconds would work properly in practice. This has not been verified over a wide variety of network delays, and clearly if there is a very slow net which stretches out the intersegment arrival time, a fixed interval will fail. In a sophisticated TCP, which is expected to adjust dynamically (rather than manually) to changing network conditions, it would be appropriate to measure this interval and respond dynamically. The following algorithm, which has been relegated to an Appendix, because it has not been tested, seems sensible. Whenever a segment arrives which does not have the push bit on in it, start a timer, which runs until the next segment arrives. Average these interarrival intervals, using an exponential decay smoothing function tuned to take into account perhaps the last ten or twenty segments that have come in. Occasionally, there will be a long interarrival period, even for a segment which is does not terminate a piece of data being pushed, perhaps because a window has gone to zero or some glitch in the sender or the network has held up the data. Therefore, examine each interarrival interval, and discard it from the smoothing algorithm if it exceeds the current estimate by some amount, perhaps a ratio of two or four times. By rejecting the larger intersegment arrival intervals, one should obtain a smoothed estimate of the interarrival of segments inside a burst. The number need not be exact, since the timer which triggers acknowledgement can add a fairly generous fudge factor to this without causing trouble with the sender's estimate of the retransmission interval, so long as the fudge factor is constant.

RFC: 814

NAME, ADDRESSES, PORTS, AND ROUTES

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. Introduction

It has been said that the principal function of an operating system is to define a number of different names for the same object, so that it can busy itself keeping track of the relationship between all of the different names. Network protocols seem to have somewhat the same characteristic. In TCP/IP, there are several ways of referring to things. At the human visible interface, there are character string "names" to identify networks, hosts, and services. Host names are translated into network "addresses", 32-bit values that identify the network to which a host is attached, and the location of the host on that net. Service names are translated into a "port identifier", which in TCP is a 16-bit value. Finally, addresses are translated into "routes", which are the sequence of steps a packet must take to reach the specified addresses. Routes show up explicitly in the form of the internet routing options, and also implicitly in the address to route translation tables which all hosts and gateways maintain.

This RFC gives suggestions and guidance for the design of the tables and algorithms necessary to keep track of these various sorts of identifiers inside a host implementation of TCP/IP.

2. The Scope of the Problem

One of the first questions one can ask about a naming mechanism is how many names one can expect to encounter. In order to answer this, it is necessary to know something about the expected maximum size of the internet. Currently, the internet is fairly small. It contains no more than 25 active networks, and no more than a few hundred hosts. This makes it possible to install tables which exhaustively list all of these elements. However, any implementation undertaken now should be based on an assumption of a much larger internet. The guidelines currently recommended are an upper limit of about 1,000 networks. If we imagine an average number of 25 hosts per net, this would suggest a maximum number of 25,000 hosts. It is quite unclear whether this host estimate is high or low, but even if it is off by several factors of two, the resulting number is still large enough to suggest that current table management strategies are unacceptable. Some fresh techniques will be required to deal with the internet of the future.

3. Names

As the previous section suggests, the internet will eventually have a sufficient number of names that a host cannot have a static table which provides a translation from every name to its associated address. There are several reasons other than sheer size why a host would not wish to have such a table. First, with that many names, we can expect names to be added and deleted at such a rate that an installer might spend all his time just revising the table. Second, most of the names will refer to addresses of machines with which nothing will ever be exchanged. In fact, there may be whole networks with which a particular host will never have any traffic.

To cope with this large and somewhat dynamic environment, the internet is moving from its current position in which a single name table is maintained by the NIC and distributed to all hosts, to a distributed approach in which each network (or group of networks) is responsible for maintaining its own names and providing a "name server" to translate between the names and the addresses in that network. Each host is assumed to store not a complete set of name-address translations, but only a cache of recently used names. When a name is provided by a user for translation to an address, the host will first examine its local cache, and if the name is not found there, will communicate with an appropriate name server to obtain the information, which it may then insert into its cache for future reference.

Unfortunately, the name server mechanism is not totally in place in the internet yet, so for the moment, it is necessary to continue to use the old strategy of maintaining a complete table of all names in every host. Implementors, however, should structure this table in such a way that it is easy to convert later to a name server approach. In particular, a reasonable programming strategy would be to make the name table accessible only through a subroutine interface, rather than by scattering direct references to the table all through the code. In this way, it will be possible, at a later date, to replace the subroutine with one capable of making calls on remote name servers.

A problem which occasionally arises in the ARPANET today is that the information in a local host table is out of date, because a host has moved, and a revision of the host table has not yet been installed from the NIC. In this case, one attempts to connect to a particular host and discovers an unexpected machine at the address obtained from the local table. If a human is directly observing the connection attempt, the error is usually detected immediately. However, for unattended operations such as the sending of queued mail, this sort of problem can lead to a great deal of confusion.

The nameserver scheme will only make this problem worse, if hosts cache locally the address associated with names that have been looked up, because the host has no way of knowing when the address has changed and the cache entry should be removed. To solve this problem, plans are currently under way to define a simple facility by which a host can query a foreign address to determine what name is actually associated with it. SMTP already defines a verification technique based on this approach.

4. Addresses

The IP layer must know something about addresses. In particular, when a datagram is being sent out from a host, the IP layer must decide where to send it on the immediately connected network, based on the internet address. Mechanically, the IP first tests the internet address to see whether the network number of the recipient is the same as the network number of the sender. If so, the packet can be sent directly to the final recipient. If not, the datagram must be sent to a gateway for further forwarding. In this latter case, a second decision must be made, as there may be more than one gateway available on the immediately attached network.

When the internet address format was first specified, 8 bits were reserved to identify the network. Early implementations thus implemented the above algorithm by means of a table with 256 entries, one for each possible net, that specified the gateway of choice for that net, with a special case entry for those nets to which the host was immediately connected. Such tables were sometimes statically filled in, which caused confusion and malfunctions when gateways and networks moved (or crashed).

The current definition of the internet address provides three different options for network numbering, with the goal of allowing a very large number of networks to be part of the internet. Thus, it is no longer possible to imagine having an exhaustive table to select a gateway for any foreign net. Again, current implementations must use a strategy based on a local cache of routing information for addresses currently being used.

The recommended strategy for address to route translation is as follows. When the IP layer receives an outbound datagram for transmission, it extracts the network number from the destination address, and queries its local table to determine whether it knows a suitable gateway to which to send the datagram. If it does, the job is done. (But see RFC 816 on Fault Isolation and Recovery, for recommendations on how to deal with the possible failure of the gateway.) If there is no such entry in the local table, then select any accessible gateway at random, insert that as an entry in the table, and use it to send the packet. Either the guess will be right or wrong. If it is wrong, the gateway to which the packet was sent will return an ICMP redirect message to report that there is a better gateway to reach the net in question. The arrival of this redirect should cause an update of the local table.

The number of entries in the local table should be determined by the maximum number of active connections which this particular host can support at any one time. For a large time sharing system, one might imagine a table with 100 or more entries. For a personal computer being used to support a single user telnet connection, only one address to gateway association need be maintained at once.

The above strategy actually does not completely solve the problem, but only pushes it down one level, where the problem then arises of how a new host, freshly arriving on the internet, finds all of its accessible gateways. Intentionally, this problem is not solved within the internetwork architecture. The reason is that different networks have drastically different strategies for allowing a host to find out about other hosts on its immediate network. Some nets permit a broadcast mechanism. In this case, a host can send out a message and expect an answer back from all of the attached gateways. In other cases, where a particular network is richly provided with tools to support the internet, there may be a special network mechanism which a host can invoke to determine where the gateways are. In other cases, it may be necessary for an installer to manually provide the name of at least one accessible gateway. Once a host has discovered the name of one gateway, it can build up a table of all other available gateways, by keeping track of every gateway that has been reported back to it in an ICMP message.

5. Advanced Topics in Addressing and Routing

The preceding discussion describes the mechanism required in a minimal implementation, an implementation intended only to provide operational service access today to the various networks that make up the internet. For any host which will participate in future research, as contrasted with service, some additional features are required. These features will also be helpful for service hosts if they wish to obtain access to some of the more exotic networks which will become part of the internet over the next few years. All implementors are urged to at least provide a structure into which these features could be later integrated.

There are several features, either already a part of the architecture or now under development, which are used to modify or expand the relationships between addresses and routes. The IP source route options allow a host to explicitly direct a datagram through a series of gateways to its foreign host. An alternative form of the ICMP redirect packet has been proposed, which would return information specific to a particular destination host, not a destination net. Finally, additional IP options have been proposed to identify particular routes within the internet that are unacceptable. The difficulty with implementing these new features is that the mechanisms do not lie entirely within the bounds of IP. All the mechanisms above are designed to apply to a particular connection, so that their use must be specified at the TCP level. Thus, the interface between IP and the layers above it must include mechanisms to allow passing this information back and forth, and TCP (or any other protocol at this level, such as UDP), must be prepared to store this information. The passing of information between IP and TCP is made more complicated by the fact that some of the information, in particular ICMP packets, may arrive at any time. The normal interface envisioned between TCP and IP is one across which packets can be sent or received. The existence of asynchronous ICMP messages implies that there must be an additional channel between the two, unrelated to the actual sending and receiving of data. (In fact, there are many other ICMP messages which arrive asynchronously and which must be passed from IP up to higher layers. See RFC 816, Fault Isolation and Recovery.)

Source routes are already in use in the internet, and many implementations will wish to be able to take advantage of them. The following sorts of usages should be permitted. First, a user, when initiating a TCP connection, should be able to hand a source route into TCP, which in turn must hand the source route to IP with every outgoing datagram. The user might initially obtain the source route by querying a different sort of name server, which would return a source route instead of an address, or the user may have fabricated the source route manually. A TCP which is listening for a connection, rather than attempting to open one, must be prepared to receive a datagram which contains a IP return route, in which case it must remember this return route, and use it as a source route on all returning datagrams.

6. Ports and Service Identifiers

The IP layer of the architecture contains the address information which specifies the destination host to which the datagram is being sent. In fact, datagrams are not intended just for particular hosts, but for particular agents within a host, processes or other entities that are the actual source and sink of the data. IP performs only a very simple dispatching once the datagram has arrived at the target host, it dispatches it to a particular protocol. It is the responsibility of that protocol handler, for example TCP, to finish dispatching the datagram to the particular connection for which it is destined. This next layer of dispatching is done using "port identifiers", which are a part of the header of the higher level protocol, and not the IP layer.

This two-layer dispatching architecture has caused a problem for certain implementations. In particular, some implementations have wished to put the IP layer within the kernel of the operating system, and the TCP layer as a user domain application program. Strict adherence to this partitioning can lead to grave performance problems, for the datagram must first be dispatched from the kernel to a TCP process, which then dispatches the datagram to its final destination process. The overhead of scheduling this dispatch process can severely limit the achievable throughput of the implementation.

As is discussed in RFC 817, Modularity and Efficiency in Protocol Implementations, this particular separation between kernel and user leads to other performance problems, even ignoring the issue of port level dispatching. However, there is an acceptable shortcut which can be taken to move the higher level dispatching function into the IP layer, if this makes the implementation substantially easier.

In principle, every higher level protocol could have a different dispatching algorithm. The reason for this is discussed below. However, for the protocols involved in the service offering being implemented today, TCP and UDP, the dispatching algorithm is exactly the same, and the port field is located in precisely the same place in the header. Therefore, unless one is interested in participating in further protocol research, there is only one higher level dispatch algorithm. This algorithm takes into account the internet level foreign address, the protocol number, and the local port and foreign port from the higher level protocol header. This algorithm can be implemented as a sort of adjunct to the IP layer implementation, as long as no other higher level protocols are to be implemented. (Actually, the above statement is only partially true, in that the UDP dispatch function is subset of the TCP dispatch function. UDP dispatch depends only protocol number and local port. However, there is an occasion within TCP when this exact same subset comes into play, when a process wishes to listen for a connection from any foreign host. Thus, the range of mechanisms necessary to support TCP dispatch are also sufficient to support precisely the UDP requirement.)

The decision to remove port level dispatching from IP to the higher level protocol has been questioned by some implementors. It has been argued that if all of the address structure were part of the IP layer, then IP could do all of the packet dispatching function within the host, which would lead to a simpler modularity. Three problems were identified with this. First, not all protocol implementors could agree on the size of the port identifier. TCP selected a fairly short port identifier, 16 bits, to reduce header size. Other protocols being designed, however, wanted a larger port identifier, perhaps 32 bits, so that the port identifier, if properly selected, could be considered probabilistically unique. Thus, constraining the port id to one particular IP level mechanism would prevent certain fruitful lines of research. Second, ports serve a special function in addition to datagram delivery: certain port numbers are reserved to identify particular services. Thus, TCP port 23 is the remote login service. If ports were implemented at the IP level, then the assignment of well known ports could not be done on a protocol basis, but would have to be done in a centralized manner for all of the IP architecture. Third, IP was designed with a very simple layering role: IP contained exactly those functions that the gateways must understand. If the port idea had been made a part of the IP layer, it would have suggested that gateways needed to know about ports, which is not the case.

There are, of course, other ways to avoid these problems. In particular, the "well-known port" problem can be solved by devising a second mechanism, distinct from port dispatching, to name well-known ports. Several protocols have settled on the idea of including, in the packet which sets up a connection to a particular service, a more general service descriptor, such as a character string field. These special packets, which are requesting connection to a particular service, are routed on arrival to a special server, sometimes called a "rendezvous server", which examines the service request, selects a random port which is to be used for this instance of the service, and then passes the packet along to the service itself to commence the interaction.

For the internet architecture, this strategy had the serious flaw that it presumed all protocols would fit into the same service paradigm: an initial setup phase, which might contain a certain overhead such as indirect routing through a rendezvous server, followed by the packets of the interaction itself, which would flow directly to the process providing the service. Unfortunately, not all high level protocols in internet were expected to fit this model. The best example of this is isolated datagram exchange using UDP. The simplest exchange in UDP is one process sending a single datagram to another. Especially on a local net, where the net related overhead is very low, this kind of simple single datagram interchange can be extremely efficient, with very low overhead in the hosts. However, since these individual packets would not be part of an established connection, if IP supported a strategy based on a rendezvous server and service descriptors, every isolated datagram would have to be routed indirectly in the receiving host through the rendezvous server, which would substantially increase the overhead of processing, and every datagram would have to carry the full service request field, which would increase the size of the packet header.

In general, if a network is intended for "virtual circuit service", or things similar to that, then using a special high overhead mechanism for circuit setup makes sense. However, current directions in research are leading away from this class of protocol, so once again the architecture was designed not to preclude alternative protocol structures. The only rational position was that the particular dispatching strategy used should be part of the higher level protocol design, not the IP layer.

This same argument about circuit setup mechanisms also applies to the design of the IP address structure. Many protocols do not transmit a full address field as part of every packet, but rather transmit a short identifier which is created as part of a circuit setup from source to destination. If the full address needs to be carried in only the first packet of a long exchange, then the overhead of carrying a very long address field can easily be justified. Under these circumstances, one can create truly extravagant address fields, which are capable of extending to address almost any conceivable entity. However, this strategy is useable only in a virtual circuit net, where the packets being transmitted are part of a established sequence, otherwise this large extravagant address must be transported on every packet. Since Internet explicitly rejected this restriction on the architecture, it was necessary to come up with an address field that was compact enough to be sent in every datagram, but general enough to correctly route the datagram through the catanet without a previous setup phase. The IP address of 32 bits is the compromise that results. Clearly it requires a substantial amount of shoehorning to address all of the interesting places in the universe with only 32 bits. On the other hand, had the address field become much bigger, IP would have been susceptible to another criticism, which is that the header had grown unworkably large. Again, the fundamental design decision was that the protocol be designed in such a way that it supported research in new and different sorts of protocol architectures.

There are some limited restrictions imposed by the IP design on the port mechanism selected by the higher level process. In particular, when a packet goes awry somewhere on the internet, the offending packet is returned, along with an error indication, as part of an ICMP packet. An ICMP packet returns only the IP layer, and the next 64 bits of the original datagram. Thus, any higher level protocol which wishes to sort out from which port a particular offending datagram came must make sure that the port information is contained within the first 64 bits of the next level header. This also means, in most cases, that it is possible to imagine, as part of the IP layer, a port dispatch mechanism which works by masking and matching on the first 64 bits of the incoming higher level header.

RFC: 815

IP DATAGRAM REASSEMBLY ALGORITHMS

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. Introduction

One of the mechanisms of IP is fragmentation and reassembly. Under certain circumstances, a datagram originally transmitted as a single unit will arrive at its final destination broken into several fragments. The IP layer at the receiving host must accumulate these fragments until enough have arrived to completely reconstitute the original datagram. The specification document for IP gives a complete description of the reassembly mechanism, and contains several examples. It also provides one possible algorithm for reassembly, based on keeping track of arriving fragments in a vector of bits. This document describes an alternate approach which should prove more suitable in some machines.

A superficial examination of the reassembly process may suggest that it is rather complicated. First, it is necessary to keep track of all the fragments, which suggests a small bookkeeping job. Second, when a new fragment arrives, it may combine with the existing fragments in a number of different ways. It may precisely fill the space between two fragments, or it may overlap with existing fragments, or completely duplicate existing fragments, or partially fill a space between two fragments without abutting either of them. Thus, it might seem that the reassembly process might involve designing a fairly complicated algorithm that tests for a number of different options.

In fact, the process of reassembly is extremely simple. This document describes a way of dealing with reassembly which reduces the bookkeeping problem to a minimum, which requires for storage only one buffer equal in size to the final datagram being reassembled, which can reassemble a datagram from any number of fragments arriving in any order with any possible pattern of overlap and duplication, and which is appropriate for almost any sort of operating system.

The reader should consult the IP specification document to be sure that he is completely familiar with the general concept of reassembly, and the particular header fields and vocabulary used to describe the process.

2. The Algorithm

In order to define this reassembly algorithm, it is necessary to define some terms. A partially reassembled datagram consists of certain sequences of octets that have already arrived, and certain areas still to come. We will refer to these missing areas as "holes". Each hole can be characterized by two numbers, hole.first, the number of the first octet in the hole, and hole.last, the number of the last octet in the hole. This pair of numbers we will call the "hole descriptor", and we will assume that all of the hole descriptors for a particular datagram are gathered together in the "hole descriptor list".

The general form of the algorithm is as follows. When a new fragment of the datagram arrives, it will possibly fill in one or more of the existing holes. We will examine each of the entries in the hole descriptor list to see whether the hole in question is eliminated by this incoming fragment. If so, we will delete that entry from the list. Eventually, a fragment will arrive which eliminates every entry from the list. At this point, the datagram has been completely reassembled and can be passed to higher protocol levels for further processing.

The algorithm will be described in two phases. In the first part, we will show the sequence of steps which are executed when a new fragment arrives, in order to determine whether or not any of the existing holes are filled by the new fragment. In the second part of this description, we will show a ridiculously simple algorithm for management of the hole descriptor list.

3. Fragment Processing Algorithm

An arriving fragment can fill any of the existing holes in a number of ways. Most simply, it can completely fill a hole. Alternatively, it may leave some remaining space at either the beginning or the end of an existing hole. Or finally, it can lie in the middle of an existing hole, breaking the hole in half and leaving a smaller hole at each end. Because of these possibilities, it might seem that a number of tests must be made when a new fragment arrives, leading to a rather complicated algorithm. In fact, if properly expressed, the algorithm can compare each hole to the arriving fragment in only four tests.

We start the algorithm when the earliest fragment of the datagram arrives. We begin by creating an empty data buffer area and putting one entry in its hole descriptor list, the entry which describes the datagram as being completely missing. In this case, hole.first equals zero, and hole.last equals infinity. (Infinity is presumably implemented by a very large integer, greater than 576, of the implementor's choice.) The following eight steps are then used to insert each of the arriving fragments into the buffer area where the complete datagram is being built up. The arriving fragment is described by fragment.first, the first octet of the fragment, and fragment.last, the last octet of the fragment.

1. Select the next hole descriptor from the hole descriptor list. If there are no more entries, go to step eight.

2. If fragment.first is greater than hole.last, go to step one.

3. If fragment.last is less than hole.first, go to step one.

   - (If either step two or step three is true, then the newly arrived fragment does not overlap with the hole in any way, so we need pay no further attention to this hole. We return to the beginning of the algorithm where we select the next hole for examination.)

4. Delete the current entry from the hole descriptor list.

   - (Since neither step two nor step three was true, the newly arrived fragment does interact with this hole in some way. Therefore, the current descriptor will no longer be valid. We will destroy it, and in the next two steps we will determine whether or not it is necessary to create any new hole descriptors.)

5. If fragment.first is greater than hole.first, then create a new hole descriptor "new_hole" with new_hole.first equal to hole.first, and new_hole.last equal to fragment.first minus one.

- (If the test in step five is true, then the first part of the original hole is not filled by this fragment. We create a new descriptor for this smaller hole.)

6. If fragment.last is less than hole.last and fragment.more fragments is true, then create a new hole descriptor "new_hole", with new_hole.first equal to fragment.last plus one and new_hole.last equal to hole.last.

    - (This test is the mirror of step five with one additional feature. Initially, we did not know how long the reassembled datagram would be, and therefore we created a hole reaching from zero to infinity. Eventually, we will receive the last fragment of the datagram. At this point, that hole descriptor which reaches from the last octet of the buffer to infinity can be discarded. The fragment which contains the last fragment indicates this fact by a flag in the internet header called "more fragments". The test of this bit in this statement prevents us from creating a descriptor for the unneeded hole which describes the space from the end of the datagram to infinity.)

7. Go to step one.

8. If the hole descriptor list is now empty, the datagram is now complete. Pass it on to the higher level protocol processor for further handling. Otherwise, return.

4. Part Two: Managing the Hole Descriptor List

The main complexity in the eight step algorithm above is not performing the arithmetical tests, but in adding and deleting entries from the hole descriptor list. One could imagine an implementation in which the storage management package was many times more complicated than the rest of the algorithm, since there is no specified upper limit on the number of hole descriptors which will exist for a datagram during reassembly. There is a very simple way to deal with the hole descriptors, however. Just put each hole descriptor in the first octets of the hole itself. Note that by the definition of the reassembly algorithm, the minimum size of a hole is eight octets. To store hole.first and hole.last will presumably require two octets each. An additional two octets will be required to thread together the entries on the hole descriptor list. This leaves at least two more octets to deal with implementation idiosyncrasies.

There is only one obvious pitfall to this storage strategy. One must execute the eight step algorithm above before copying the data from the fragment into the reassembly buffer. If one were to copy the data first, it might smash one or more hole descriptors. Once the algorithm above has been run, any hole descriptors which are about to be smashed have already been rendered obsolete.

5. Loose Ends

Scattering the hole descriptors throughout the reassembly buffer itself requires that they be threaded onto some sort of list so that they can be found. This in turn implies that there must be a pointer to the head of the list. In many cases, this pointer can be stored in some sort of descriptor block which the implementation associates with each reassembly buffer. If no such storage is available, a dirty but effective trick is to store the head of the list in a part of the internet header in the reassembly buffer which is no longer needed. An obvious location is the checksum field.

When the final fragment of the datagram arrives, the packet length field in the internet header should be filled in.

6. Options

The preceding description made one unacceptable simplification. It assumed that there were no internet options associated with the datagram being reassembled. The difficulty with options is that until one receives the first fragment of the datagram, one cannot tell how big the internet header will be. This is because, while certain options are copied identically into every fragment of a datagram, other options, such as "record route", are put in the first fragment only. (The "first fragment" is the fragment containing octet zero of the original datagram.)

Until one knows how big the internet header is, one does not know where to copy the data from each fragment into the reassembly buffer. If the earliest fragment to arrive happens to be the first fragment, then this is no problem. Otherwise, there are two solutions. First, one can leave space in the reassembly buffer for the maximum possible internet header. In fact, the maximum size is not very large, 64 octets. Alternatively, one can simply gamble that the first fragment will contain no options. If, when the first fragment finally arrives, there are options, one can then shift the data in the buffer a sufficient distance for allow for them. The only peril in copying the data is that one will trash the pointers that thread the hole descriptors together. It is easy to see how to untrash the pointers.

The source and record route options have the interesting feature that, since different fragments can follow different paths, they may arrive with different return routes recorded in different fragments.

Normally, this is more information than the receiving Internet module needs. The specified procedure is to take the return route recorded in the first fragment and ignore the other versions.

7. The Complete Algorithm

In addition to the algorithm described above there are two parts to the reassembly process. First, when a fragment arrives, it is necessary to find the reassembly buffer associated with that fragment. This requires some mechanism for searching all the existing reassembly buffers. The correct reassembly buffer is identified by an equality of the following fields: the foreign and local internet address, the protocol ID, and the identification field.

The final part of the algorithm is some sort of timer based mechanism which decrements the time to live field of each partially reassembled datagram, so that incomplete datagrams which have outlived their usefulness can be detected and deleted. One can either create a demon which comes alive once a second and decrements the field in each datagram by one, or one can read the clock when each first fragment arrives, and queue some sort of timer call, using whatever system mechanism is appropriate, to reap the datagram when its time has come.

An implementation of the complete algorithm comprising all these parts was constructed in BCPL as a test. The complete algorithm took less than one and one-half pages of listing, and generated approximately 400 nova machine instructions. That portion of the algorithm actually involved with management of hole descriptors is about 20 lines of code.

9

The version of the algorithm described here is actually a simplification of the author's original version, thanks to an insightful observation by Elizabeth Martin at MIT.

RFC: 816

FAULT ISOLATION AND RECOVERY

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. Introduction

Occasionally, a network or a gateway will go down, and the sequence of hops which the packet takes from source to destination must change. Fault isolation is that action which hosts and gateways collectively take to determine that something is wrong; fault recovery is the identification and selection of an alternative route which will serve to reconnect the source to the destination. In fact, the gateways perform most of the functions of fault isolation and recovery. There are, however, a few actions which hosts must take if they wish to provide a reasonable level of service. This document describes the portion of fault isolation and recovery which is the responsibility of the host.

2. What Gateways Do

Gateways collectively implement an algorithm which identifies the best route between all pairs of networks. They do this by exchanging packets which contain each gateway's latest opinion about the operational status of its neighbor networks and gateways. Assuming that this algorithm is operating properly, one can expect the gateways to go through a period of confusion immediately after some network or gateway has failed, but one can assume that once a period of negotiation has passed, the gateways are equipped with a consistent and correct model of the connectivity of the internet. At present this period of negotiation may actually take several minutes, and many TCP implementations time out within that period, but it is a design goal of the eventual algorithm that the gateway should be able to reconstruct the topology quickly enough that a TCP connection should be able to survive a failure of the route.

3. Host Algorithm for Fault Recovery

Since the gateways always attempt to have a consistent and correct model of the internetwork topology, the host strategy for fault recovery is very simple. Whenever the host feels that something is wrong, it asks the gateway for advice, and, assuming the advice is forthcoming, it believes the advice completely. The advice will be wrong only during the transient period of negotiation, which immediately follows an outage, but will otherwise be reliably correct.

In fact, it is never necessary for a host to explicitly ask a gateway for advice, because the gateway will provide it as appropriate. When a host sends a datagram to some distant net, the host should be prepared to receive back either of two advisory messages which the gateway may send. The ICMP "redirect" message indicates that the gateway to which the host sent the datagram is not longer the best gateway to reach the net in question. The gateway will have forwarded the datagram, but the host should revise its routing table to have a different immediate address for this net. The ICMP "destination unreachable" message indicates that as a result of an outage, it is currently impossible to reach the addressed net or host in any manner. On receipt of this message, a host can either abandon the connection immediately without any further retransmission, or resend slowly to see if the fault is corrected in reasonable time.

If a host could assume that these two ICMP messages would always arrive when something was amiss in the network, then no other action on the part of the host would be required in order maintain its tables in an optimal condition. Unfortunately, there are two circumstances under which the messages will not arrive properly. First, during the transient following a failure, error messages may arrive that do not correctly represent the state of the world. Thus, hosts must take an isolated error message with some scepticism. (This transient period is discussed more fully below.) Second, if the host has been sending datagrams to a particular gateway, and that gateway itself crashes, then all the other gateways in the internet will reconstruct the topology, but the gateway in question will still be down, and therefore cannot provide any advice back to the host. As long as the host continues to direct datagrams at this dead gateway, the datagrams will simply vanish off the face of the earth, and nothing will come back in return. Hosts must detect this failure.

If some gateway many hops away fails, this is not of concern to the host, for then the discovery of the failure is the responsibility of the immediate neighbor gateways, which will perform this action in a manner invisible to the host. The problem only arises if the very first gateway, the one to which the host is immediately sending the datagrams, fails. We thus identify one single task which the host must perform as its part of fault isolation in the internet: the host must use some strategy to detect that a gateway to which it is sending datagrams is dead.

Let us assume for the moment that the host implements some algorithm to detect failed gateways; we will return later to discuss what this algorithm might be. First, let us consider what the host should do when it has determined that a gateway is down. In fact, with the exception of one small problem, the action the host should take is extremely simple. The host should select some other gateway, and try sending the datagram to it. Assuming that gateway is up, this will either produce correct results, or some ICMP advice. Since we assume that, ignoring temporary periods immediately following an outage, any gateway is capable of giving correct advice, once the host has received advice from any gateway, that host is in as good a condition as it can hope to be.

There is always the unpleasant possibility that when the host tries a different gateway, that gateway too will be down. Therefore, whatever algorithm the host uses to detect a dead gateway must continuously be applied, as the host tries every gateway in turn that it knows about.

The only difficult part of this algorithm is to specify the means by which the host maintains the table of all of the gateways to which it has immediate access. Currently, the specification of the internet protocol does not architect any message by which a host can ask to be supplied with such a table. The reason is that different networks may provide very different mechanisms by which this table can be filled in. For example, if the net is a broadcast net, such as an ethernet or a ringnet, every gateway may simply broadcast such a table from time to time, and the host need do nothing but listen to obtain the required information. Alternatively, the network may provide the mechanism of logical addressing, by which a whole set of machines can be provided with a single group address, to which a request can be sent for assistance. Failing those two schemes, the host can build up its table of neighbor gateways by remembering all the gateways from which it has ever received a message. Finally, in certain cases, it may be necessary for this table, or at least the initial entries in the table, to be constructed manually by a manager or operator at the site. In cases where the network in question provides absolutely no support for this kind of host query, at least some manual intervention will be required to get started, so that the host can find out about at least one gateway.

4. Host Algorithms for Fault Isolation

We now return to the question raised above. What strategy should the host use to detect that it is talking to a dead gateway, so that it can know to switch to some other gateway in the list. In fact, there are several algorithms which can be used. All are reasonably simple to implement, but they have very different implications for the overhead on the host, the gateway, and the network. Thus, to a certain extent, the algorithm picked must depend on the details of the network and of the host.

1. NETWORK LEVEL DETECTION

Many networks, particularly the Arpanet, perform precisely the required function internal to the network. If a host sends a datagram to a dead gateway on the Arpanet, the network will return a "host dead" message, which is precisely the information the host needs to know in order to switch to another gateway. Some early implementations of Internet on the Arpanet threw these messages away. That is an exceedingly poor idea.

2. CONTINUOUS POLLING

The ICMP protocol provides an echo mechanism by which a host may solicit a response from a gateway. A host could simply send this message at a reasonable rate, to assure itself continuously that the gateway was still up. This works, but, since the message must be sent fairly often to detect a fault in a reasonable time, it can imply an unbearable overhead on the host itself, the network, and the gateway. This strategy is prohibited except where a specific analysis has indicated that the overhead is tolerable.

3. TRIGGERED POLLING

If the use of polling could be restricted to only those times when something seemed to be wrong, then the overhead would be bearable. Provided that one can get the proper advice from one's higher level protocols, it is possible to implement such a strategy. For example, one could program the TCP level so that whenever it retransmitted a segment more than once, it sent a hint down to the IP layer which triggered polling. This strategy does not have excessive overhead, but does have the problem that the host may be somewhat slow to respond to an error, since only after polling has started will the host be able to confirm that something has gone wrong, and by then the TCP above may have already timed out.

Both forms of polling suffer from a minor flaw. Hosts as well as gateways respond to ICMP echo messages. Thus, polling cannot be used to detect the error that a foreign address thought to be a gateway is actually a host. Such a confusion can arise if the physical addresses of machines are rearranged.

4. TRIGGERED RESELECTION

There is a strategy which makes use of a hint from a higher level, as did the previous strategy, but which avoids polling altogether. Whenever a higher level complains that the service seems to be defective, the Internet layer can pick the next gateway from the list of available gateways, and switch to it. Assuming that this gateway is up, no real harm can come of this decision, even if it was wrong, for the worst that will happen is a redirect message which instructs the host to return to the gateway originally being used. If, on the other hand, the original gateway was indeed down, then this immediately provides a new route, so the period of time until recovery is shortened. This last strategy seems particularly clever, and is probably the most generally suitable for those cases where the network itself does not provide fault isolation. (Regretably, I have forgotten who suggested this idea to me. It is not my invention.)

5. Higher Level Fault Detection

The previous discussion has concentrated on fault detection and recovery at the IP layer. This section considers what the higher layers such as TCP should do.

TCP has a single fault recovery action; it repeatedly retransmits a segment until either it gets an acknowledgement or its connection timer expires. As discussed above, it may use retransmission as an event to trigger a request for fault recovery to the IP layer. In the other direction, information may flow up from IP, reporting such things as ICMP Destination Unreachable or error messages from the attached network. The only subtle question about TCP and faults is what TCP should do when such an error message arrives or its connection timer expires.

The TCP specification discusses the timer. In the description of the open call, the timeout is described as an optional value that the client of TCP may specify; if any segment remains unacknowledged for this period, TCP should abort the connection. The default for the timeout is 30 seconds. Early TCPs were often implemented with a fixed timeout interval, but this did not work well in practice, as the following discussion may suggest.

Clients of TCP can be divided into two classes: those running on immediate behalf of a human, such as Telnet, and those supporting a program, such as a mail sender. Humans require a sophisticated response to errors. Depending on exactly what went wrong, they may want to abandon the connection at once, or wait for a long time to see if things get better. Programs do not have this human impatience, but also lack the power to make complex decisions based on details of the exact error condition. For them, a simple timeout is reasonable.

Based on these considerations, at least two modes of operation are needed in TCP. One, for programs, abandons the connection without exception if the TCP timer expires. The other mode, suitable for people, never abandons the connection on its own initiative, but reports to the layer above when the timer expires. Thus, the human user can see error messages coming from all the relevant layers, TCP and ICMP, and can request TCP to abort as appropriate. This second mode requires that TCP be able to send an asynchronous message up to its client to report the timeout, and it requires that error messages arriving at lower layers similarly flow up through TCP.

At levels above TCP, fault detection is also required. Either of the following can happen. First, the foreign client of TCP can fail, even though TCP is still running, so data is still acknowledged and the timer never expires. Alternatively, the communication path can fail, without the TCP timer going off, because the local client has no data to send. Both of these have caused trouble.

Sending mail provides an example of the first case. When sending mail using SMTP, there is an SMTP level acknowledgement that is returned when a piece of mail is successfully delivered. Several early mail receiving programs would crash just at the point where they had received all of the mail text (so TCP did not detect a timeout due to outstanding unacknowledged data) but before the mail was acknowledged at the SMTP level. This failure would cause early mail senders to wait forever for the SMTP level acknowledgement. The obvious cure was to set a timer at the SMTP level, but the first attempt to do this did not work, for there was no simple way to select the timer interval. If the interval selected was short, it expired in normal operational when sending a large file to a slow host. An interval of many minutes was needed to prevent false timeouts, but that meant that failures were detected only very slowly. The current solution in several mailers is to pick a timeout interval proportional to the size of the message.

Server telnet provides an example of the other kind of failure. It can easily happen that the communications link can fail while there is no traffic flowing, perhaps because the user is thinking. Eventually, the user will attempt to type something, at which time he will discover that the connection is dead and abort it. But the host end of the connection, having nothing to send, will not discover anything wrong, and will remain waiting forever. In some systems there is no way for a user in a different process to destroy or take over such a hanging process, so there is no way to recover.

One solution to this would be to have the host server telnet query the user end now and then, to see if it is still up. (Telnet does not have an explicit query feature, but the host could negotiate some unimportant option, which should produce either agreement or disagreement in return.) The only problem with this is that a reasonable sample interval, if applied to every user on a large system, can generate an unacceptable amount of traffic and system overhead. A smart server telnet would use this query only when something seems wrong, perhaps when there had been no user activity for some time.

In both these cases, the general conclusion is that client level error detection is needed, and that the details of the mechanism are very dependent on the application. Application programmers must be made aware of the problem of failures, and must understand that error detection at the TCP or lower level cannot solve the whole problem for them.

6. Knowing When to Give Up

It is not obvious, when error messages such as ICMP Destination Unreachable arrive, whether TCP should abandon the connection. The reason that error messages are difficult to interpret is that, as discussed above, after a failure of a gateway or network, there is a transient period during which the gateways may have incorrect information, so that irrelevant or incorrect error messages may sometimes return. An isolated ICMP Destination Unreachable may arrive at a host, for example, if a packet is sent during the period when the gateways are trying to find a new route. To abandon a TCP connection based on such a message arriving would be to ignore the valuable feature of the Internet that for many internal failures it reconstructs its function without any disruption of the end points.

But if failure messages do not imply a failure, what are they for? In fact, error messages serve several important purposes. First, if they arrive in response to opening a new connection, they probably are caused by opening the connection improperly (e.g., to a non-existent address) rather than by a transient network failure. Second, they provide valuable information, after the TCP timeout has occurred, as to the probable cause of the failure. Finally, certain messages, such as ICMP Parameter Problem, imply a possible implementation problem. In general, error messages give valuable information about what went wrong, but are not to be taken as absolutely reliable. A general alerting mechanism, such as the TCP timeout discussed above, provides a good indication that whatever is wrong is a serious condition, but without the advisory messages to augment the timer, there is no way for the client to know how to respond to the error. The combination of the timer and the advice from the error messages provide a reasonable set of facts for the client layer to have. It is important that error messages from all layers be passed up to the client module in a useful and consistent way.

RFC: 817

MODULARITY AND EFFICIENCY IN PROTOCOL IMPLEMENTATION

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. Introduction

Many protocol implementers have made the unpleasant discovery that their packages do not run quite as fast as they had hoped. The blame for this widely observed problem has been attributed to a variety of causes, ranging from details in the design of the protocol to the underlying structure of the host operating system. This RFC will discuss some of the commonly encountered reasons why protocol implementations seem to run slowly.

Experience suggests that one of the most important factors in determining the performance of an implementation is the manner in which that implementation is modularized and integrated into the host operating system. For this reason, it is useful to discuss the question of how an implementation is structured at the same time that we consider how it will perform. In fact, this RFC will argue that modularity is one of the chief villains in attempting to obtain good performance, so that the designer is faced with a delicate and inevitable tradeoff between good structure and good performance. Further, the single factor which most strongly determines how well this conflict can be resolved is not the protocol but the operating system.

2. Efficiency Considerations

There are many aspects to efficiency. One aspect is sending data at minimum transmission cost, which is a critical aspect of common carrier communications, if not in local area network communications. Another aspect is sending data at a high rate, which may not be possible at all if the net is very slow, but which may be the one central design constraint when taking advantage of a local net with high raw bandwidth. The final consideration is doing the above with minimum expenditure of computer resources. This last may be necessary to achieve high speed, but in the case of the slow net may be important only in that the resources used up, for example cpu cycles, are costly or otherwise needed. It is worth pointing out that these different goals often conflict; for example it is often possible to trade off efficient use of the computer against efficient use of the network. Thus, there may be no such thing as a successful general purpose protocol implementation.

The simplest measure of performance is throughput, measured in bits per second. It is worth doing a few simple computations in order to get a feeling for the magnitude of the problems involved. Assume that data is being sent from one machine to another in packets of 576 bytes, the maximum generally acceptable internet packet size. Allowing for header overhead, this packet size permits 4288 bits in each packet. If a useful throughput of 10,000 bits per second is desired, then a data bearing packet must leave the sending host about every 430 milliseconds, a little over two per second. This is clearly not difficult to achieve. However, if one wishes to achieve 100 kilobits per second throughput, the packet must leave the host every 43 milliseconds, and to achieve one megabit per second, which is not at all unreasonable on a high-speed local net, the packets must be spaced no more than 4.3 milliseconds.

These latter numbers are a slightly more alarming goal for which to set one's sights. Many operating systems take a substantial fraction of a millisecond just to service an interrupt. If the protocol has been structured as a process, it is necessary to go through a process scheduling before the protocol code can even begin to run. If any piece of a protocol package or its data must be fetched from disk, real time delays of between 30 to 100 milliseconds can be expected. If the protocol must compete for cpu resources with other processes of the system, it may be necessary to wait a scheduling quantum before the protocol can run. Many systems have a scheduling quantum of 100 milliseconds or more. Considering these sorts of numbers, it becomes immediately clear that the protocol must be fitted into the operating system in a thorough and effective manner if any like reasonable throughput is to be achieved.

There is one obvious conclusion immediately suggested by even this simple analysis. Except in very special circumstances, when many packets are being processed at once, the cost of processing a packet is dominated by factors, such as cpu scheduling, which are independent of the packet size. This suggests two general rules which any implementation ought to obey. First, send data in large packets. Obviously, if processing time per packet is a constant, then throughput will be directly proportional to the packet size. Second, never send an unneeded packet. Unneeded packets use up just as many resources as a packet full of data, but perform no useful function. RFC 813, "Window and Acknowledgement Strategy in TCP", discusses one aspect of reducing the number of packets sent per useful data byte. This document will mention other attacks on the same problem.

The above analysis suggests that there are two main parts to the problem of achieving good protocol performance. The first has to do with how the protocol implementation is integrated into the host operating system. The second has to do with how the protocol package itself is organized internally. This document will consider each of these topics in turn.

3. The Protocol vs. the Operating System

There are normally three reasonable ways in which to add a protocol to an operating system. The protocol can be in a process that is provided by the operating system, or it can be part of the kernel of the operating system itself, or it can be put in a separate communications processor or front end machine. This decision is strongly influenced by details of hardware architecture and operating system design; each of these three approaches has its own advantages and disadvantages.

The "process" is the abstraction which most operating systems use to provide the execution environment for user programs. A very simple path for implementing a protocol is to obtain a process from the operating system and implement the protocol to run in it. Superficially, this approach has a number of advantages. Since modifications to the kernel are not required, the job can be done by someone who is not an expert in the kernel structure. Since it is often impossible to find somebody who is experienced both in the structure of the operating system and the structure of the protocol, this path, from a management point of view, is often extremely appealing. Unfortunately, putting a protocol in a process has a number of disadvantages, related to both structure and performance. First, as was discussed above, process scheduling can be a significant source of real-time delay. There is not only the actual cost of going through the scheduler, but the problem that the operating system may not have the right sort of priority tools to bring the process into execution quickly whenever there is work to be done.

Structurally, the difficulty with putting a protocol in a process is that the protocol may be providing services, for example support of data streams, which are normally obtained by going to special kernel entry points. Depending on the generality of the operating system, it may be impossible to take a program which is accustomed to reading through a kernel entry point, and redirect it so it is reading the data from a process. The most extreme example of this problem occurs when implementing server telnet. In almost all systems, the device handler for the locally attached teletypes is located inside the kernel, and programs read and write from their teletype by making kernel calls. If server telnet is implemented in a process, it is then necessary to take the data streams provided by server telnet and somehow get them back down inside the kernel so that they mimic the interface provided by local teletypes. It is usually the case that special kernel modification is necessary to achieve this structure, which somewhat defeats the benefit of having removed the protocol from the kernel in the first place.

Clearly, then, there are advantages to putting the protocol package in the kernel. Structurally, it is reasonable to view the network as a device, and device drivers are traditionally contained in the kernel. Presumably, the problems associated with process scheduling can be sidesteped, at least to a certain extent, by placing the code inside the kernel. And it is obviously easier to make the server telnet channels mimic the local teletype channels if they are both realized in the same level in the kernel.

However, implementation of protocols in the kernel has its own set of pitfalls. First, network protocols have a characteristic which is shared by almost no other device: they require rather complex actions to be performed as a result of a timeout. The problem with this requirement is that the kernel often has no facility by which a program can be brought into execution as a result of the timer event. What is really needed, of course, is a special sort of process inside the kernel. Most systems lack this mechanism. Failing that, the only execution mechanism available is to run at interrupt time.

There are substantial drawbacks to implementing a protocol to run at interrupt time. First, the actions performed may be somewhat complex and time consuming, compared to the maximum amount of time that the operating system is prepared to spend servicing an interrupt. Problems can arise if interrupts are masked for too long. This is particularly bad when running as a result of a clock interrupt, which can imply that the clock interrupt is masked. Second, the environment provided by an interrupt handler is usually extremely primitive compared to the environment of a process. There are usually a variety of system facilities which are unavailable while running in an interrupt handler. The most important of these is the ability to suspend execution pending the arrival of some event or message. It is a cardinal rule of almost every known operating system that one must not invoke the scheduler while running in an interrupt handler. Thus, the programmer who is forced to implement all or part of his protocol package as an interrupt handler must be the best sort of expert in the operating system involved, and must be prepared for development sessions filled with obscure bugs which crash not just the protocol package but the entire operating system.

A final problem with processing at interrupt time is that the system scheduler has no control over the percentage of system time used by the protocol handler. If a large number of packets arrive, from a foreign host that is either malfunctioning or fast, all of the time may be spent in the interrupt handler, effectively killing the system.

There are other problems associated with putting protocols into an operating system kernel. The simplest problem often encountered is that the kernel address space is simply too small to hold the piece of code in question. This is a rather artificial sort of problem, but it is a severe problem none the less in many machines. It is an appallingly unpleasant experience to do an implementation with the knowledge that for every byte of new feature put in one must find some other byte of old feature to throw out. It is hopeless to expect an effective and general implementation under this kind of constraint. Another problem is that the protocol package, once it is thoroughly entwined in the operating system, may need to be redone every time the operating system changes. If the protocol and the operating system are not maintained by the same group, this makes maintenance of the protocol package a perpetual headache.

The third option for protocol implementation is to take the protocol package and move it outside the machine entirely, on to a separate processor dedicated to this kind of task. Such a machine is often described as a communications processor or a front-end processor. There are several advantages to this approach. First, the operating system on the communications processor can be tailored for precisely this kind of task. This makes the job of implementation much easier. Second, one does not need to redo the task for every machine to which the protocol is to be added. It may be possible to reuse the same front-end machine on different host computers. Since the task need not be done as many times, one might hope that more attention could be paid to doing it right. Given a careful implementation in an environment which is optimized for this kind of task, the resulting package should turn out to be very efficient. Unfortunately, there are also problems with this approach. There is, of course, a financial problem associated with buying an additional computer. In many cases, this is not a problem at all since the cost is negligible compared to what the programmer would cost to do the job in the mainframe itself. More fundamentally, the communications processor approach does not completely sidestep any of the problems raised above. The reason is that the communications processor, since it is a separate machine, must be attached to the mainframe by some mechanism. Whatever that mechanism, code is required in the mainframe to deal with it. It can be argued that the program to deal with the communications processor is simpler than the program to implement the entire protocol package. Even if that is so, the communications processor interface package is still a protocol in nature, with all of the same structural problems. Thus, all of the issues raised above must still be faced. In addition to those problems, there are some other, more subtle problems associated with an outboard implementation of a protocol. We will return to these problems later.

There is a way of attaching a communications processor to a mainframe host which sidesteps all of the mainframe implementation problems, which is to use some preexisting interface on the host machine as the port by which a communications processor is attached. This strategy is often used as a last stage of desperation when the software on the host computer is so intractable that it cannot be changed in any way. Unfortunately, it is almost inevitably the case that all of the available interfaces are totally unsuitable for this purpose, so the result is unsatisfactory at best. The most common way in which this form of attachment occurs is when a network connection is being used to mimic local teletypes. In this case, the front-end processor can be attached to the mainframe by simply providing a number of wires out of the front-end processor, each corresponding to a connection, which are plugged into teletype ports on the mainframe computer. (Because of the appearance of the physical configuration which results from this arrangement, Michael Padlipsky has described this as the "milking machine" approach to computer networking.) This strategy solves the immediate problem of providing remote access to a host, but it is extremely inflexible. The channels being provided to the host are restricted by the host software to one purpose only, remote login. It is impossible to use them for any other purpose, such as file transfer or sending mail, so the host is integrated into the network environment in an extremely limited and inflexible manner. If this is the best that can be done, then it should be tolerated. Otherwise, implementors should be strongly encouraged to take a more flexible approach.

4. Protocol Layering

The previous discussion suggested that there was a decision to be made as to where a protocol ought to be implemented. In fact, the decision is much more complicated than that, for the goal is not to implement a single protocol, but to implement a whole family of protocol layers, starting with a device driver or local network driver at the bottom, then IP and TCP, and eventually reaching the application specific protocol, such as Telnet, FTP and SMTP on the top. Clearly, the bottommost of these layers is somewhere within the kernel, since the physical device driver for the net is almost inevitably located there. Equally clearly, the top layers of this package, which provide the user his ability to perform the remote login function or to send mail, are not entirely contained within the kernel. Thus, the question is not whether the protocol family shall be inside or outside the kernel, but how it shall be sliced in two between that part inside and that part outside.

Since protocols come nicely layered, an obvious proposal is that one of the layer interfaces should be the point at which the inside and outside components are sliced apart. Most systems have been implemented in this way, and many have been made to work quite effectively. One obvious place to slice is at the upper interface of TCP. Since TCP provides a bidirectional byte stream, which is somewhat similar to the I/O facility provided by most operating systems, it is possible to make the interface to TCP almost mimic the interface to other existing devices. Except in the matter of opening a connection, and dealing with peculiar failures, the software using TCP need not know that it is a network connection, rather than a local I/O stream that is providing the communications function. This approach does put TCP inside the kernel, which raises all the problems addressed above. It also raises the problem that the interface to the IP layer can, if the programmer is not careful, become excessively buried inside the kernel. It must be remembered that things other than TCP are expected to run on top of IP. The IP interface must be made accessible, even if TCP sits on top of it inside the kernel.

Another obvious place to slice is above Telnet. The advantage of slicing above Telnet is that it solves the problem of having remote login channels emulate local teletype channels. The disadvantage of putting Telnet into the kernel is that the amount of code which has now been included there is getting remarkably large. In some early implementations, the size of the network package, when one includes protocols at the level of Telnet, rivals the size of the rest of the supervisor. This leads to vague feelings that all is not right.

Any attempt to slice through a lower layer boundary, for example between internet and TCP, reveals one fundamental problem. The TCP layer, as well as the IP layer, performs a demultiplexing function on incoming datagrams. Until the TCP header has been examined, it is not possible to know for which user the packet is ultimately destined. Therefore, if TCP, as a whole, is moved outside the kernel, it is necessary to create one separate process called the TCP process, which performs the TCP multiplexing function, and probably all of the rest of TCP processing as well. This means that incoming data destined for a user process involves not just a scheduling of the user process, but scheduling the TCP process first.

This suggests an alternative structuring strategy which slices through the protocols, not along an established layer boundary, but along a functional boundary having to do with demultiplexing. In this approach, certain parts of IP and certain parts of TCP are placed in the kernel. The amount of code placed there is sufficient so that when an incoming datagram arrives, it is possible to know for which process that datagram is ultimately destined. The datagram is then routed directly to the final process, where additional IP and TCP processing is performed on it. This removes from the kernel any requirement for timer based actions, since they can be done by the process provided by the user. This structure has the additional advantage of reducing the amount of code required in the kernel, so that it is suitable for systems where kernel space is at a premium. The RFC 814, titled "Names, Addresses, Ports, and Routes," discusses this rather orthogonal slicing strategy in more detail.

A related discussion of protocol layering and multiplexing can be found in Cohen and Postel [1].

5. Breaking Down the Barriers

In fact, the implementor should be sensitive to the possibility of even more peculiar slicing strategies in dividing up the various protocol layers between the kernel and the one or more user processes. The result of the strategy proposed above was that part of TCP should execute in the process of the user. In other words, instead of having one TCP process for the system, there is one TCP process per connection. Given this architecture, it is not longer necessary to imagine that all of the TCPs are identical. One TCP could be optimized for high throughput applications, such as file transfer. Another TCP could be optimized for small low delay applications such as Telnet. In fact, it would be possible to produce a TCP which was somewhat integrated with the Telnet or FTP on top of it. Such an integration is extremely important, for it can lead to a kind of efficiency which more traditional structures are incapable of producing. Earlier, this paper pointed out that one of the important rules to achieving efficiency was to send the minimum number of packets for a given amount of data. The idea of protocol layering interacts very strongly (and poorly) with this goal, because independent layers have independent ideas about when packets should be sent, and unless these layers can somehow be brought into cooperation, additional packets will flow. The best example of this is the operation of server telnet in a character at a time remote echo mode on top of TCP. When a packet containing a character arrives at a server host, each layer has a different response to that packet. TCP has an obligation to acknowledge the packet. Either server telnet or the application layer above has an obligation to echo the character received in the packet. If the character is a Telnet control sequence, then Telnet has additional actions which it must perform in response to the packet. The result of this, in most implementations, is that several packets are sent back in response to the one arriving packet. Combining all of these return messages into one packet is important for several reasons. First, of course, it reduces the number of packets being sent over the net, which directly reduces the charges incurred for many common carrier tariff structures. Second, it reduces the number of scheduling actions which will occur inside both hosts, which, as was discussed above, is extremely important in improving throughput.

The way to achieve this goal of packet sharing is to break down the barrier between the layers of the protocols, in a very restrained and careful manner, so that a limited amount of information can leak across the barrier to enable one layer to optimize its behavior with respect to the desires of the layers above and below it. For example, it would represent an improvement if TCP, when it received a packet, could ask the layer above whether or not it would be worth pausing for a few milliseconds before sending an acknowledgement in order to see if the upper layer would have any outgoing data to send. Dallying before sending the acknowledgement produces precisely the right sort of optimization if the client of TCP is server Telnet. However, dallying before sending an acknowledgement is absolutely unacceptable if TCP is being used for file transfer, for in file transfer there is almost never data flowing in the reverse direction, and the delay in sending the acknowledgement probably translates directly into a delay in obtaining the next packets. Thus, TCP must know a little about the layers above it to adjust its performance as needed.

It would be possible to imagine a general purpose TCP which was equipped with all sorts of special mechanisms by which it would query the layer above and modify its behavior accordingly. In the structures suggested above, in which there is not one but several TCPs, the TCP can simply be modified so that it produces the correct behavior as a matter of course. This structure has the disadvantage that there will be several implementations of TCP existing on a single machine, which can mean more maintenance headaches if a problem is found where TCP needs to be changed. However, it is probably the case that each of the TCPs will be substantially simpler than the general purpose TCP which would otherwise have been built. There are some experimental projects currently under way which suggest that this approach may make designing of a TCP, or almost any other layer, substantially easier, so that the total effort involved in bringing up a complete package is actually less if this approach is followed. This approach is by no means generally accepted, but deserves some consideration.

The general conclusion to be drawn from this sort of consideration is that a layer boundary has both a benefit and a penalty. A visible layer boundary, with a well specified interface, provides a form of isolation between two layers which allows one to be changed with the confidence that the other one will not stop working as a result. However, a firm layer boundary almost inevitably leads to inefficient operation. This can easily be seen by analogy with other aspects of operating systems. Consider, for example, file systems. A typical operating system provides a file system, which is a highly abstracted representation of a disk. The interface is highly formalized, and presumed to be highly stable. This makes it very easy for naive users to have access to disks without having to write a great deal of software. The existence of a file system is clearly beneficial. On the other hand, it is clear that the restricted interface to a file system almost inevitably leads to inefficiency. If the interface is organized as a sequential read and write of bytes, then there will be people who wish to do high throughput transfers who cannot achieve their goal. If the interface is a virtual memory interface, then other users will regret the necessity of building a byte stream interface on top of the memory mapped file. The most objectionable inefficiency results when a highly sophisticated package, such as a data base management package, must be built on top of an existing operating system. Almost inevitably, the implementors of the database system attempt to reject the file system and obtain direct access to the disks. They have sacrificed modularity for efficiency.

The same conflict appears in networking, in a rather extreme form.

The concept of a protocol is still unknown and frightening to most naive programmers. The idea that they might have to implement a protocol, or even part of a protocol, as part of some application package, is a dreadful thought. And thus there is great pressure to hide the function of the net behind a very hard barrier. On the other hand, the kind of inefficiency which results from this is a particularly undesirable sort of inefficiency, for it shows up, among other things, in increasing the cost of the communications resource used up to achieve the application goal. In cases where one must pay for one's communications costs, they usually turn out to be the dominant cost within the system. Thus, doing an excessively good job of packaging up the protocols in an inflexible manner has a direct impact on increasing the cost of the critical resource within the system. This is a dilemma which will probably only be solved when programmers become somewhat less alarmed about protocols, so that they are willing to weave a certain amount of protocol structure into their application program, much as application programs today weave parts of database management systems into the structure of their application program.

An extreme example of putting the protocol package behind a firm layer boundary occurs when the protocol package is relegated to a front-end processor. In this case the interface to the protocol is some other protocol. It is difficult to imagine how to build close cooperation between layers when they are that far separated. Realistically, one of the prices which must be associated with an implementation so physically modularized is that the performance will suffer as a result. Of course, a separate processor for protocols could be very closely integrated into the mainframe architecture, with interprocessor co-ordination signals, shared memory, and similar features. Such a physical modularity might work very well, but there is little documented experience with this closely coupled architecture for protocol support.

6. Efficiency of Protocol Processing

To this point, this document has considered how a protocol package should be broken into modules, and how those modules should be distributed between free standing machines, the operating system kernel, and one or more user processes. It is now time to consider the other half of the efficiency question, which is what can be done to speed the execution of those programs that actually implement the protocols. We will make some specific observations about TCP and IP, and then conclude with a few generalities.

IP is a simple protocol, especially with respect to the processing of normal packets, so it should be easy to get it to perform efficiently. The only area of any complexity related to actual packet processing has to do with fragmentation and reassembly. The reader is referred to RFC 815, titled "IP Datagram Reassembly Algorithms", for specific consideration of this point.

Most costs in the IP layer come from table look up functions, as opposed to packet processing functions. An outgoing packet requires two translation functions to be performed. The internet address must be translated to a target gateway, and a gateway address must be translated to a local network number (if the host is attached to more than one network). It is easy to build a simple implementation of these table look up functions that in fact performs very poorly. The programmer should keep in mind that there may be as many as a thousand network numbers in a typical configuration. Linear searching of a thousand entry table on every packet is extremely unsuitable. In fact, it may be worth asking TCP to cache a hint for each connection, which can be handed down to IP each time a packet is sent, to try to avoid the overhead of a table look up.

TCP is a more complex protocol, and presents many more opportunities for getting things wrong. There is one area which is generally accepted as causing noticeable and substantial overhead as part of TCP processing. This is computation of the checksum. It would be nice if this cost could be avoided somehow, but the idea of an end-to-end checksum is absolutely central to the functioning of TCP. No host implementor should think of omitting the validation of a checksum on incoming data.

Various clever tricks have been used to try to minimize the cost of computing the checksum. If it is possible to add additional microcoded instructions to the machine, a checksum instruction is the most obvious candidate. Since computing the checksum involves picking up every byte of the segment and examining it, it is possible to combine the operation of computing the checksum with the operation of copying the segment from one location to another. Since a number of data copies are probably already required as part of the processing structure, this kind of sharing might conceivably pay off if it didn't cause too much trouble to the modularity of the program. Finally, computation of the checksum seems to be one place where careful attention to the details of the algorithm used can make a drastic difference in the throughput of the program. The Multics system provides one of the best case studies of this, since Multics is about as poorly organized to perform this function as any machine implementing TCP. Multics is a 36-bit word machine, with four 9-bit bytes per word. The eight-bit bytes of a TCP segment are laid down packed in memory, ignoring word boundaries. This means that when it is necessary to pick up the data as a set of 16-bit units for the purpose of adding them to compute checksums, horrible masking and shifting is required for each 16-bit value. An early version of a program using this strategy required 6 milliseconds to checksum a 576-byte segment. Obviously, at this point, checksum computation was becoming the central bottleneck to throughput. A more careful recoding of this algorithm reduced the checksum processing time to less than one millisecond. The strategy used was extremely dirty. It involved adding up carefully selected words of the area in which the data lay, knowing that for those particular words, the 16-bit values were properly aligned inside the words. Only after the addition had been done were the various sums shifted, and finally added to produce the eventual checksum. This kind of highly specialized programming is probably not acceptable if used everywhere within an operating system. It is clearly appropriate for one highly localized function which can be clearly identified as an extreme performance bottleneck.

Another area of TCP processing which may cause performance problems is the overhead of examining all of the possible flags and options which occur in each incoming packet. One paper, by Bunch and Day [2], asserts that the overhead of packet header processing is actually an important limiting factor in throughput computation. Not all measurement experiments have tended to support this result. To whatever extent it is true, however, there is an obvious strategy which the implementor ought to use in designing his program. He should build his program to optimize the expected case. It is easy, especially when first designing a program, to pay equal attention to all of the possible outcomes of every test. In practice, however, few of these will ever happen. A TCP should be built on the assumption that the next packet to arrive will have absolutely nothing special about it, and will be the next one expected in the sequence space. One or two tests are sufficient to determine that the expected set of control flags are on. (The ACK flag should be on; the Push flag may or may not be on. No other flags should be on.) One test is sufficient to determine that the sequence number of the incoming packet is one greater than the last sequence number received. In almost every case, that will be the actual result. Again, using the Multics system as an example, failure to optimize the case of receiving the expected sequence number had a detectable effect on the performance of the system. The particular problem arose when a number of packets arrived at once. TCP attempted to process all of these packets before awaking the user. As a result, by the time the last packet arrived, there was a threaded list of packets which had several items on it. When a new packet arrived, the list was searched to find the location into which the packet should be inserted. Obviously, the list should be searched from highest sequence number to lowest sequence number, because one is expecting to receive a packet which comes after those already received. By mistake, the list was searched from front to back, starting with the packets with the lowest sequence number. The amount of time spent searching this list backwards was easily detectable in the metering measurements.

Other data structures can be organized to optimize the action which is normally taken on them. For example, the retransmission queue is very seldom actually used for retransmission, so it should not be organized to optimize that action. In fact, it should be organized to optimized the discarding of things from it when the acknowledgement arrives. In many cases, the easiest way to do this is not to save the packet at all, but to reconstruct it only if it needs to be retransmitted, starting from the data as it was originally buffered by the user.

There is another generality, at least as important as optimizing the common case, which is to avoid copying data any more times than necessary. One more result from the Multics TCP may prove enlightening here. Multics takes between two and three milliseconds within the TCP layer to process an incoming packet, depending on its size. For a 576-byte packet, the three milliseconds is used up approximately as follows. One millisecond is used computing the checksum. Six hundred microseconds is spent copying the data. (The data is copied twice, at .3 milliseconds a copy.) One of those copy operations could correctly be included as part of the checksum cost, since it is done to get the data on a known word boundary to optimize the checksum algorithm.

However, the copy also performs another necessary transfer at the same time. Header processing and packet resequencing takes .7 milliseconds. The rest of the time is used in miscellaneous processing, such as removing packets from the retransmission queue which are acknowledged by this packet. Data copying is the second most expensive single operation after data checksuming. Some implementations, often because of an excessively layered modularity, end up copying the data around a great deal. Other implementations end up copying the data because there is no shared memory between processes, and the data must be moved from process to process via a kernel operation. Unless the amount of this activity is kept strictly under control, it will quickly become the major performance bottleneck.

7. Conclusions

This document has addressed two aspects of obtaining performance from a protocol implementation, the way in which the protocol is layered and integrated into the operating system, and the way in which the detailed handling of the packet is optimized. It would be nice if one or the other of these costs would completely dominate, so that all of one's attention could be concentrated there. Regrettably, this is not so. Depending on the particular sort of traffic one is getting, for example, whether Telnet one-byte packets or file transfer maximum size packets at maximum speed, one can expect to see one or the other cost being the major bottleneck to throughput. Most implementors who have studied their programs in an attempt to find out where the time was going have reached the unsatisfactory conclusion that it is going equally to all parts of their program. With the possible exception of checksum processing, very few people have ever found that their performance problems were due to a single, horrible bottleneck which they could fix by a single stroke of inventive programming. Rather, the performance was something which was improved by painstaking tuning of the entire program.

Most discussions of protocols begin by introducing the concept of layering, which tends to suggest that layering is a fundamentally wonderful idea which should be a part of every consideration of protocols. In fact, layering is a mixed blessing. Clearly, a layer interface is necessary whenever more than one client of a particular layer is to be allowed to use that same layer. But an interface, precisely because it is fixed, inevitably leads to a lack of complete understanding as to what one layer wishes to obtain from another. This has to lead to inefficiency. Furthermore, layering is a potential snare in that one is tempted to think that a layer boundary, which was an artifact of the specification procedure, is in fact the proper boundary to use in modularizing the implementation. Again, in certain cases, an architected layer must correspond to an implemented layer, precisely so that several clients can have access to that layer in a reasonably straightforward manner. In other cases, cunning rearrangement of the implemented module boundaries to match with various functions, such as the demultiplexing of incoming packets, or the sending of asynchronous outgoing packets, can lead to unexpected performance improvements compared to more traditional implementation strategies. Finally, good performance is something which is difficult to retrofit onto an existing program. Since performance is influenced, not just by the fine detail, but by the gross structure, it is sometimes the case that in order to obtain a substantial performance improvement, it is necessary to completely redo the program from the bottom up. This is a great disappointment to programmers, especially those doing a protocol implementation for the first time. Programmers who are somewhat inexperienced and unfamiliar with protocols are sufficiently concerned with getting their program logically correct that they do not have the capacity to think at the same time about the performance of the structure they are building. Only after they have achieved a logically correct program do they discover that they have done so in a way which has precluded real performance. Clearly, it is more difficult to design a program thinking from the start about both logical correctness and performance. With time, as implementors as a group learn more about the appropriate structures to use for building protocols, it will be possible to proceed with an implementation project having more confidence that the structure is rational, that the program will work, and that the program will work well. Those of us now implementing protocols have the privilege of being on the forefront of this learning process. It should be no surprise that our programs sometimes suffer from the uncertainty we bring to bear on them.

Citations

[1] Cohen and Postel, "On Protocol Multiplexing", Sixth Data Communications Symposium, ACM/IEEE, November 1979.

[2] Bunch and Day, "Control Structure Overhead in TCP", Trends and Applications: Computer Networking, NBS Symposium, May 1980.

IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. COM-22, NO. 5, MAY 1974

A Protocol for Packet Network Intercommunication

VINTON G. CERF AND ROBERT E. KAHN, MEMBER, IEEE

Abstract—A protocol that supports the sharing of resources that exist in different packet switching networks is presented. The protocol provides for variation in individual network packet sizes, transmission failures, sequencing, flow control, end-to-end error checking, and the creation and destruction of logical process-to-process connections. Some implementation issues are considered, and problems such as internetwork routing, accounting, and timeouts are exposed.

INTRODUCTION

IN THE LAST few years considerable effort has been expended on the design and implementation of packet switching networks [1]–[7],[14],[17]. A principle reason for developing such networks has been to facilitate the sharing of computer resources. A packet communication network includes a transportation mechanism for delivering data between computers or between computers and terminals. To make the data meaningful, computers and terminals share a common protocol (i.e., a set of agreed upon conventions). Several protocols have already been developed for this purpose [8]–[12],[16]. However, these protocols have addressed only the problem of communication on the same network. In this paper we present a protocol design and philosophy that supports the sharing of resources that exist in different packet switching networks.

After a brief introduction to internetwork protocol issues, we describe the function of a GATEWAY as an interface between networks and discuss its role in the protocol. We then consider the various details of the protocol, including addressing, formatting, buffering, sequencing, flow control, error control, and so forth. We close with a description of an interprocess communication mechanism and show how it can be supported by the internetwork protocol.

Even though many different and complex problems must be solved in the design of an individual packet switching network, these problems are manifestly compounded when dissimilar networks are interconnected. Issues arise which may have no direct counterpart in an individual network and which strongly influence the way in which internetwork communication can take place.

A typical packet switching network is composed of a set of computer resources called HOSTS, a set of one or more *packet switches*, and a collection of communication media that interconnect the packet switches. Within each HOST, we assume that there exist *processes* which must communicate with processes in their own or other HOSTS. Any current definition of a process will be adequate for our purposes [13]. These processes are generally the ultimate source and destination of data in the network. Typically, within an individual network, there exists a protocol for communication between any source and destination process. Only the source and destination processes require knowledge of this convention for communication to take place. Processes in two distinct networks would ordinarily use different protocols for this purpose. The ensemble of packet switches and communication media is called the *packet switching subnet*. Fig. 1 illustrates these ideas.

In a typical packet switching subnet, data of a fixed maximum size are accepted from a source HOST, together with a formatted destination address which is used to route the data in a store and forward fashion. The transmit time for this data is usually dependent upon internal network parameters such as communication media data rates, buffering and signaling strategies, routing, propagation delays, etc. In addition, some mechanism is generally present for error handling and determination of status of the networks components.

Individual packet switching networks may differ in their implementations as follows.

1) Each network may have distinct ways of addressing the receiver, thus requiring that a uniform addressing scheme be created which can be understood by each individual network.

2) Each network may accept data of different maximum size, thus requiring networks to deal in units of the smallest maximum size (which may be impractically small) or requiring procedures which allow data crossing a network boundary to be reformatted into smaller pieces.

3) The success or failure of a transmission and its performance in each network is governed by different time delays in accepting, delivering, and transporting the data. This requires careful development of internetwork timing procedures to insure that data can be successfully delivered through the various networks.

4) Within each network, communication may be disrupted due to unrecoverable mutation of the data or missing data. End-to-end restoration procedures are desirable to allow complete recovery from these conditions.

Paper approved by the Associate Editor for Data Communications of the IEEE Communications Society for publication without oral presentation. Manuscript received November 5, 1973. The research reported in this paper was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC 15-73-C-0370.

V. G. Cerf is with the Department of Computer Science and Electrical Engineering, Stanford University, Stanford, Calif.
R. E. Kahn is with the Information Processing Technology Office, Advanced Research Projects Agency, Department of Defense, Arlington, Va.

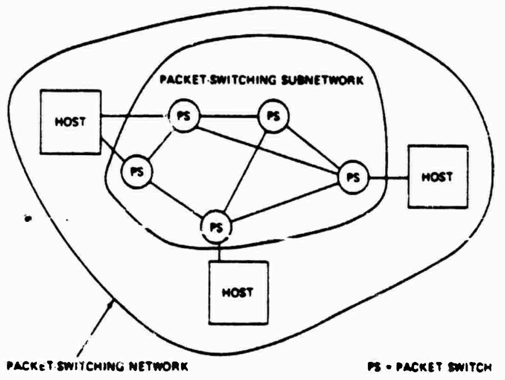


Fig. 1. Typical packet switching network.

5) Status information, routing, fault detection, and isolation are typically different in each network. Thus, to obtain verification of certain conditions, such as an inaccessible or dead destination, various kinds of coordination must be invoked between the communicating networks.

It would be extremely convenient if all the differences between networks could be economically resolved by suitable interfacing at the network boundaries. For many of the differences, this objective can be achieved. However, both economic and technical considerations lead us to prefer that the interface be as simple and reliable as possible and deal primarily with passing data between networks that use different packet switching strategies.

The question now arises as to whether the interface ought to account for differences in HOST or process level protocols by transforming the source conventions into the corresponding destination conventions. We obviously want to allow conversion between packet switching strategies at the interface, to permit interconnection of existing and planned networks. However, the complexity and dissimilarity of the HOST or process level protocols makes it desirable to avoid having to transform between them at the interface, even if this transformation were always possible. Rather, compatible HOST and process level protocols must be developed to achieve effective internetwork resource sharing. The unacceptable alternative is for every HOST or process to implement every protocol (a potentially unbounded number) that may be needed to communicate with other networks. We therefore assume that a common protocol is to be used between HOST's or processes in different networks and that the interface between networks should take as small a role as possible in this protocol.

To allow networks under different ownership to interconnect, some accounting will undoubtedly be needed for traffic that passes across the interface. In its simplest terms, this involves an accounting of packets handled by each net for which charges are passed from net to net until the buck finally stops at the user or his representative. Furthermore, the interconnection must preserve intact the internal operation of each individual network. This is easily achieved if two networks interconnect as if each were a HOST to the other network, but without utilizing or indeed incorporating any elaborate HOST protocol transformations.

It is thus apparent that the interface between networks must play a central role in the development of any network interconnection strategy. We give a special name to this interface that performs these functions and call it a GATEWAY.

THE GATEWAY NOTION

In Fig. 2 we illustrate three individual networks labeled A, B, and C which are joined by GATEWAYS M and N. GATEWAY M interfaces network A with network B, and GATEWAY N interfaces network B to network C. We assume that an individual network may have more than one GATEWAY (e.g., network B) and that there may be more than one GATEWAY path to use in going between a pair of networks. The responsibility for properly routing data resides in the GATEWAY.

In practice, a GATEWAY between two networks may be composed of two halves, each associated with its own network. It is possible to implement each half of a GATEWAY so it need only embed internetwork packets in local packet format or extract them. We propose that the GATEWAYS handle internetwork packets in a standard format, but we are not proposing any particular transmission procedure between GATEWAY halves.

Let us now trace the flow of data through the interconnected networks. We assume a packet of data from process X enters network A destined for process Y in network C. The address of Y is initially specified by process X and the address of GATEWAY M is derived from the address of process Y. We make no attempt to specify whether the choice of GATEWAY is made by process X, its HOST, or one of the packet switches in network A. The packet traverses network A until it reaches GATEWAY M. At the GATEWAY, the packet is reformatted to meet the requirements of network B, account is taken of this unit of flow between A and B, and the GATEWAY delivers the packet to network B. Again the derivation of the next GATEWAY address is accomplished based on the address of the destination Y. In this case, GATEWAY N is the next one. The packet traverses network B until it finally reaches GATEWAY N where it is formatted to meet the requirements of network C. Account is again taken of this unit of flow between networks B and C. Upon entering network C, the packet is routed to the HOST in which process Y resides and there it is delivered to its ultimate destination.

Since the GATEWAY must understand the address of the source and destination HOSTS, this information must be available in a standard format in every packet which arrives at the GATEWAY. This information is contained in an *internetwork header* prefixed to the packet by the source HOST. The packet format, including the internet-

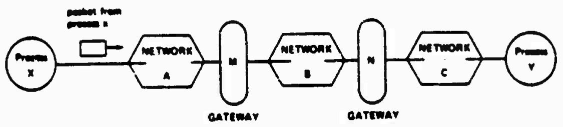


Fig. 2. Three networks interconnected by two GATEWAYS.

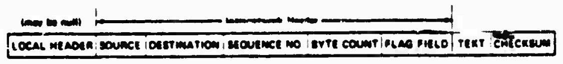


Fig. 3. Internetwork packet format (fields not shown to scale).

work header, is illustrated in Fig. 3. The source and destination entries uniformly and uniquely identify the address of every HOST in the composite network. Addressing is a subject of considerable complexity which is discussed in greater detail in the next section. The next two entries in the header provide a sequence number and a byte count that may be used to properly sequence the packets upon delivery to the destination and may also enable the GATEWAYS to detect fault conditions affecting the packet. The flag field is used to convey specific control information and is discussed in the section on retransmission and duplicate detection later. The remainder of the packet consists of text for delivery to the destination and a trailing check sum used for end-to-end software verification. The GATEWAY does *not* modify the text and merely forwards the check sum along without computing or recomputing it.

Each network may need to augment the packet format before it can pass through the individual network. We have indicated a *local header* in the figure which is prefixed to the beginning of the packet. This local header is introduced merely to illustrate the concept of embedding an internetwork packet in the format of the individual network through which the packet must pass. It will obviously vary in its exact form from network to network and may even be unnecessary in some cases. Although not explicitly indicated in the figure, it is also possible that a local trailer may be appended to the end of the packet.

Unless all transmitted packets are legislatively restricted to be small enough to be accepted by every individual network, the GATEWAY may be forced to split a packet into two or more smaller packets. This action is called fragmentation and must be done in such a way that the destination is able to piece together the fragmented packet. It is clear that the internetwork header format imposes a minimum packet size which all networks must carry (obviously all networks will want to carry packets larger than this minimum). We believe the long range growth and development of internetwork communication would be seriously inhibited by specifying how much larger than the minimum a packet size can be, for the following reasons.

1) If a maximum permitted packet size is specified then it becomes impossible to completely isolate the internal packet size parameters of one network from the internal packet size parameters of all other networks.

2) It would be very difficult to increase the maximum permitted packet size in response to new technology (e.g., large memory systems, higher data rate communication facilities, etc.) since this would require the agreement and then implementation by all participating networks.

3) Associative addressing and packet encryption may require the size of a particular packet to expand during transit for incorporation of new information.

Provision for fragmentation (regardless of where it is performed) permits packet size variations to be handled on an individual network basis without global administration and also permits HOSTS and processes to be insulated from changes in the packet sizes permitted in any networks through which their data must pass.

If fragmentation must be done, it appears best to do it upon entering the next network at the GATEWAY since only this GATEWAY (and not the other networks) must be aware of the internal packet size parameters which made the fragmentation necessary.

If a GATEWAY fragments an incoming packet into two or more packets, they must eventually be passed along to the destination HOST as fragments or reassembled for the HOST. It is conceivable that one might desire the GATEWAY to perform the reassembly to simplify the task of the destination HOST (or process) and or to take advantage of a larger packet size. We take the position that GATEWAYS should not perform this function since GATEWAY reassembly can lead to serious buffering problems, potential deadlocks, the necessity for all fragments of a packet to pass through the same GATEWAY, and increased delay in transmission. Furthermore, it is not sufficient for the GATEWAYS to provide this function since the final GATEWAY may also have to fragment a packet for transmission. Thus the destination HOST must be prepared to do this task.

Let us now turn briefly to the somewhat unusual accounting effect which arises when a packet may be fragmented by one or more GATEWAYS. We assume, for simplicity, that each network initially charges a fixed rate per packet transmitted, regardless of distance, and if one network can handle a larger packet size than another, it charges a proportionally larger price per packet. We also assume that a subsequent increase in any network's packet size does not result in additional cost per packet to its users. The charge to a user thus remains basically constant through any net which must fragment a packet. The unusual effect occurs when a packet is fragmented into smaller packets which must individually pass through a subsequent network with a larger packet size than the original unfragmented packet. We expect that most networks will naturally select packet sizes close to one another, but in any case, an increase in packet size in one net, even when it causes fragmentation, will not increase the cost of transmission and may actually decrease it. In the event that any other packet charging policies (than the one we suggest) are adopted, differences in cost can be used as an economic lever toward optimization of individual network performance.

PROCESS LEVEL COMMUNICATION

We suppose that processes wish to communicate in full duplex with their correspondents using unbounded but finite length messages. A single character might constitute the text of a message from a process to a terminal or vice versa. An entire page of characters might constitute the text of a message from a file to a process. A data stream (e.g., a continuously generated bit string) can be represented as a sequence of finite length messages.

Within a HOST we assume the existence of a transmission control program (TCP) which handles the transmission and acceptance of messages on behalf of the processes it serves. The TCP is in turn served by one or more packet switches connected to the HOST in which the TCP resides. Processes that want to communicate present messages to the TCP for transmission, and TCP's deliver incoming messages to the appropriate destination processes. We allow the TCP to break up messages into segments because the destination may restrict the amount of data that may arrive, because the local network may limit the maximum transmission size, or because the TCP may need to share its resources among many processes concurrently. Furthermore, we constrain the length of a segment to an integral number of 8-bit bytes. This uniformity is most helpful in simplifying the software needed with HOST machines of different natural word lengths. Provision at the process level can be made for padding a message that is not an integral number of bytes and for identifying which of the arriving bytes of text contain information of interest to the receiving process.

Multiplexing and demultiplexing of segments among processes are fundamental tasks of the TCP. On transmission, a TCP must multiplex together segments from different source processes and produce internetwork packets for delivery to one of its serving packet switches. On reception, a TCP will accept a sequence of packets from its serving packet switch(es). From this sequence of arriving packets (generally from different HOSTS), the TCP must be able to reconstruct and deliver messages to the proper destination processes.

We assume that every segment is augmented with additional information that allows transmitting and receiving TCP's to identify destination and source processes, respectively. At this point, we must face a major issue. How should the source TCP format segments destined for the same destination TCP? We consider two cases.

Case 1) If we take the position that segment boundaries are immaterial and that a byte stream can be formed of segments destined for the same TCP, then we may gain improved transmission efficiency and resource sharing by arbitrarily parceling the stream into packets, permitting many segments to share a single internetwork packet header. However, this position results in the need to reconstruct exactly, and in order, the stream of text bytes produced by the source TCP. At the destination, this stream must first be parsed into segments and these in turn must be used to reconstruct messages for delivery to the appropriate processes.

There are fundamental problems associated with this strategy due to the possible arrival of packets out of order at the destination. The most critical problem appears to be the amount of interference that processes sharing the same TCP-TCP byte stream may cause among themselves. This is especially so at the receiving end. First, the TCP may be put to some trouble to parse the stream back into segments and then distribute them to buffers where messages are reassembled. If it is not readily apparent that all of a segment has arrived (remember, it may come as several packets), the receiving TCP may have to suspend parsing temporarily until more packets have arrived. Second, if a packet is missing, it may not be clear whether succeeding segments, even if they are identifiable, can be passed on to the receiving process, unless the TCP has knowledge of some process level sequencing scheme. Such knowledge would permit the TCP to decide whether a succeeding segment could be delivered to its waiting process. Finding the beginning of a segment when there are gaps in the byte stream may also be hard.

Case 2): Alternatively, we might take the position that the destination TCP should be able to determine, upon its arrival and without additional information, for which process or processes a received packet is intended, and if so, whether it should be delivered then.

If the TCP is to determine for which process an arriving packet is intended, every packet must contain a *process header* (distinct from the internetwork header) that completely identifies the destination process. For simplicity, we assume that each packet contains text from a single process which is destined for a single process. Thus each packet need contain only one process header. To decide whether the arriving data is deliverable to the destination process, the TCP must be able to determine whether the data is in the proper sequence (we can make provision for the destination process to instruct its TCP to ignore sequencing, but this is considered a special case). With the assumption that each arriving packet contains a process header, the necessary sequencing and destination process identification is immediately available to the destination TCP.

Both Cases 1) and 2) provide for the demultiplexing and delivery of segments to destination processes, but only Case 2) does so without the introduction of potential interprocess interference. Furthermore, Case 1) introduces extra machinery to handle flow control on a HOST-to-HOST basis, since there must also be some provision for process level control, and this machinery is little used since the probability is small that within a given HOST, two processes will be coincidentally scheduled to send messages to the same destination HOST. For this reason, we select the method of Case 2) as a part of the *internetwork transmission protocol.*

ADDRESS FORMATS

The selection of address formats is a problem between networks because the local network addresses of TCP's may vary substantially in format and size. A uniform internetwork TCP address space, understood by each GATEWAY and TCP, is essential to routing and delivery of internetwork packets.

Similar troubles are encountered when we deal with process addressing and, more generally, port addressing. We introduce the notion of *ports* in order to permit a process to distinguish between multiple message streams. The port is simply a designator of one such message stream associated with a process. The means for identifying a port are generally different in different operating systems, and therefore, to obtain uniform addressing, a standard port address format is also required. A port address designates a full duplex message stream.

TCP ADDRESSING

TCP addressing is intimately bound up in routing issues, since a HOST or GATEWAY must choose a suitable destination HOST or GATEWAY for an outgoing internetwork packet. Let us postulate the following address format for the TCP address (Fig. 4). The choice for network identification (8 bits) allows up to 256 distinct networks. This size seems sufficient for the forseeable future. Similarly, the TCP identifier field permits up to 65 536 distinct TCP's to be addressed, which seems more than sufficient for any given network.

As each packet passes through a GATEWAY, the GATEWAY observes the destination network ID to determine how to route the packet. If the destination network is connected to the GATEWAY, the lower 16 bits of the TCP address are used to produce a local TCP address in the destination network. If the destination network is not connected to the GATEWAY, the upper 8 bits are used to select a subsequent GATEWAY. We make no effort to specify how each individual network shall associate the internetwork TCP identifier with its local TCP address. We also do not rule out the possibility that the local network understands the internetwork addressing scheme and thus alleviates the GATEWAY of the routing responsibility.

PORT ADDRESSING

A receiving TCP is faced with the task of demultiplexing the stream of internetwork packets it receives and reconstructing the original messages for each destination process. Each operating system has its own internal means of identifying processes and ports. We assume that 16 bits are sufficient to serve as internetwork port identifiers. A sending process need not know how the destination port identification will be used. The destination TCP will be able to parse this number appropriately to find the proper buffer into which it will place arriving packets. We permit a large port number field to support processes which want to distinguish between many different messages streams concurrently. In reality, we do not care how the 16 bits are sliced up by the TCP's involved.

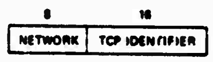


Fig. 4. TCP address.

Even though the transmitted port name field is large, it is still a compact external name for the internal representation of the port. The use of short names for port identifiers is often desirable to reduce transmission overhead and possibly reduce packet processing time at the destination TCP. Assigning short names to each port, however, requires an initial negotiation between source and destination to agree on a suitable short name assignment, the subsequent maintenance of conversion tables at both the source and the destination, and a final transaction to release the short name. For dynamic assignment of port names, this negotiation is generally necessary in any case.

SEGMENT AND PACKET FORMATS

As shown in Fig. 5, messages are broken by the TCP into segments whose format is shown in more detail in Fig. 6. The field lengths illustrated are merely suggestive. The first two fields (source port and destination port in the figure) have already been discussed in the preceding section on addressing. The uses of the third and fourth fields (window and acknowledgment in the figure) will be discussed later in the section on retransmission and duplicate detection.

We recall from Fig. 3 that an internetwork header contains both a sequence number and a byte count, as well as a flag field and a check sum. The uses of these fields are explained in the following section.

REASSEMBLY AND SEQUENCING

The reconstruction of a message at the receiving TCP clearly requires[1] that each internetwork packet carry a sequence number which is unique to its particular destination port message stream. The sequence numbers must be monotonic increasing (or decreasing) since they are used to reorder and reassemble arriving packets into a message. If the space of sequence numbers were infinite, we could simply assign the next one to each new packet. Clearly, this space cannot be infinite, and we will consider what problems a finite sequence number space will cause when we discuss retransmission and duplicate detection in the next section. We propose the following scheme for performing the sequencing of packets and hence the reconstruction of messages by the destination TCP.

A pair of ports will exchange one or more messages over a period of time. We could view the sequence of messages produced by one port as if it were embedded in an infinitely long stream of bytes. Each byte of the message has a unique sequence number which we take to be its byte location relative to the beginning of the stream. When a

[1] In the case of encrypted packets, a preliminary stage of reassembly may be required prior to decryption.

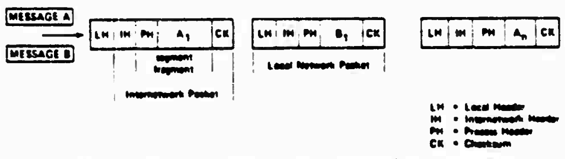


Fig. 5. Creation of segments and packets from messages.

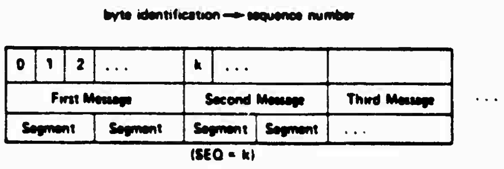


Fig. 7. Assignment of sequence numbers.

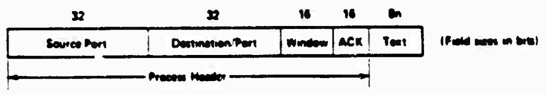


Fig. 6. Segment format (process header and text).

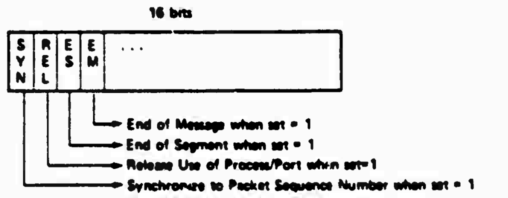

16 bits
S Y N
R E L
E S
E M
...
End of Message when set = 1
End of Segment when set = 1
Release Use of Process/Port when set=1
Synchronize to Packet Sequence Number when set = 1

Fig. 8. Internetwork header flag field.

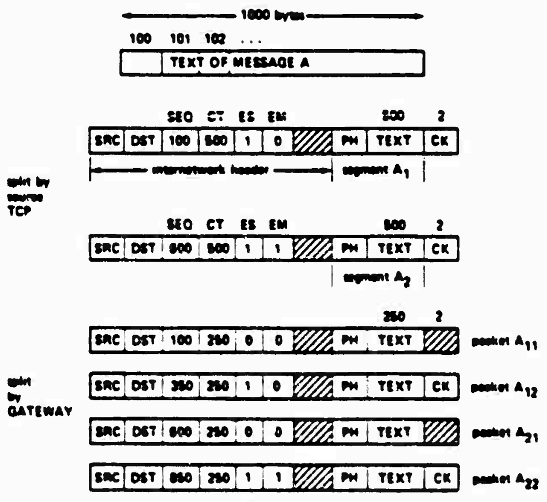


Fig. 9. Message splitting and packet splitting.

segment is extracted from the message by the source TCP and formatted for internetwork transmission, the relative location of the first byte of segment text is used as the sequence number for the packet. The byte count field in the internetwork header accounts for all the text in the segment (but does not include the check-sum bytes or the bytes in either internetwork or process header). We emphasize that the sequence number associated with a given packet is unique only to the pair of ports that are communicating (see Fig. 7). Arriving packets are examined to determine for which port they are intended. The sequence numbers on each arriving packet are then used to determine the relative location of the packet text in the messages under reconstruction. We note that this allows the exact position of the data in the reconstructed message to be determined even when pieces are still missing.

Every segment produced by a source TCP is packaged in a single internetwork packet and a check sum is computed over the text and process header associated with the segment.

The splitting of messages into segments by the TCP and the potential splitting of segments into smaller pieces by GATEWAYS creates the necessity for indicating to the destination TCP when the end of a segment (ES) has arrived and when the end of a message (EM) has arrived. The flag field of the internetwork header is used for this purpose (see Fig. 8).

The ES flag is set by the source TCP each time it prepares a segment for transmission. If it should happen that the message is completely contained in the segment, then the EM flag would also be set. The EM flag is also set on the last segment of a message, if the message could not be contained in one segment. These two flags are used by the destination TCP, respectively, to discover the presence of a check sum for a given segment and to discover that a complete message has arrived.

The ES and EM flags in the internetwork header are known to the GATEWAY and are of special importance when packets must be split apart for propagation through the next local network. We illustrate their use with an example in Fig. 9.

The original message A in Fig. 9 is shown split into two segments A_1 and A_2 and formatted by the TCP into a pair of internetwork packets. Packets A_1 and A_2 have their ES bits set, and A_2 has its EM bit set as well. When packet A_1 passes through the GATEWAY, it is split into two pieces: packet A_{11} for which neither EM nor ES bits are set, and packet A_{12} whose ES bit is set. Similarly, packet A_2 is split such that the first piece, packet A_{21}, has neither bit set, but packet A_{22} has both bits set. The sequence number field (SEQ) and the byte count field (CT) of each packet is modified by the GATEWAY to properly identify the text bytes of each packet. The GATEWAY need only examine the internetwork header to do fragmentation.

The destination TCP, upon reassembling segment A_1, will detect the ES flag and will verify the check sum it knows is contained in packet A_{12}. Upon receipt of packet A_{22}, assuming all other packets have arrived, the destination TCP detects that it has reassembled a complete message and can now advise the destination process of its receipt.

RETRANSMISSION AND DUPLICATE DETECTION

No transmission can be 100 percent reliable. We propose a timeout and positive acknowledgment mechanism which will allow TCP's to recover from packet losses from one HOST to another. A TCP transmits packets and waits for replies (acknowledgements) that are carried in the reverse packet stream. If no acknowledgment for a particular packet is received, the TCP will retransmit. It is our expectation that the HOST level retransmission mechanism, which is described in the following paragraphs, will not be called upon very often in practice. Evidence already exists[2] that individual networks can be effectively constructed without this feature. However, the inclusion of a HOST retransmission capability makes it possible to recover from occasional network problems and allows a wide range of HOST protocol strategies to be incorporated. We envision it will occasionally be invoked to allow HOST accommodation to infrequent overdemands for limited buffer resources, and otherwise not used much.

Any retransmission policy requires some means by which the receiver can detect duplicate arrivals. Even if an infinite number of distinct packet sequence numbers were available, the receiver would still have the problem of knowing how long to remember previously received packets in order to detect duplicates. Matters are complicated by the fact that only a finite number of distinct sequence numbers are in fact available, and if they are reused, the receiver must be able to distinguish between new transmissions and retransmissions.

A *window* strategy, similar to that used by the French CYCLADES system (voie virtuelle transmission mode [8]) and the ARPANET very distant HOST connection [18], is proposed here (see Fig. 10).

Suppose that the sequence number field in the internetwork header permits sequence numbers to range from 0 to $n - 1$. We assume that the sender will not transmit more than w bytes without receiving an acknowledgment. The w bytes serve as the window (see Fig. 11). Clearly, w must be less than n. The rules for sender and receiver are as follows.

Sender: Let L be the sequence number associated with the left window edge.

1) The sender transmits bytes from segments whose text lies between L and up to $L + w - 1$.

2) On timeout (duration unspecified), the sender retransmits unacknowledged bytes.

3) On receipt of acknowledgment consisting of the receiver's current left window edge, the sender's left window edge is advanced over the acknowledged bytes (advancing the right window edge implicitly).

Receiver:

1) Arriving packets whose sequence numbers coincide with the receiver's current left window edge are acknowledged by sending to the source the next sequence number expected. This effectively acknowledges bytes in between. The left window edge is advanced to the next sequence number expected.

2) Packets arriving with a sequence number to the left of the window edge (or, in fact, outside of the window) are discarded, and the current left window edge is returned as acknowledgment.

3) Packets whose sequence numbers lie within the receiver's window but do not coinicide with the receiver's left window edge are optionally kept or discarded, but are not acknowledged. This is the case when packets arrive out of order.

We make some observations on this strategy. First, all computations with sequence numbers and window edges must be made modulo n (e.g., byte 0 follows byte $n - 1$). Second, w must be less than $n/2$[3]; otherwise a retransmission may appear to the receiver to be a new transmission in the case that the receiver has accepted a window's worth of incoming packets, but all acknowledgments have been lost. Third, the receiver can either save or discard arriving packets whose sequence numbers do not coincide with the receiver's left window. Thus, in the simplest implementation, the receiver need not buffer more than one packet per message stream if space is critical. Fourth, multiple packets can be acknowledged simultaneously. Fifth, the receiver is able to deliver messages to processes in their proper order as a natural result of the reassembly mechanism. Sixth, when duplicates are detected, the acknowledgment method used naturally works to resynchronize sender and receiver. Furthermore, if the receiver accepts packets whose sequence numbers lie within the current window but

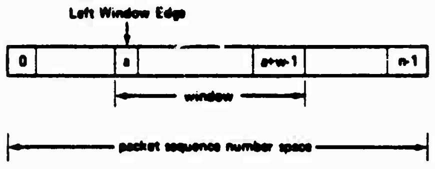


Fig. 10. The window concept.

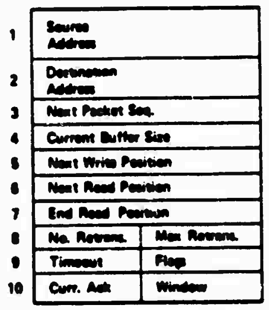


Fig. 11. Conceptual TCB format.

[2] The ARPANET is one such example.

[3] Actually $n/2$ is merely a convenient number to use; it is only required that a retransmission not appear to be a new transmission.

which are not coincident with the left window edge, an acknowledgment consisting of the current left window edge would act as a stimulus to cause retransmission of the unacknowledged bytes. Finally, we mention an overlap problem which results from retransmission, packet splitting, and alternate routing of packets through different GATEWAYS.

A 600-byte packet might pass through one GATEWAY and be broken into two 300-byte packets. On retransmission, the same packet might be broken into three 200-byte packets going through a different GATEWAY. Since each byte has a sequence number, there is no confusion at the receiving TCP. We leave for later the issue of initially synchronizing the sender and receiver left window edges and the window size.

FLOW CONTROL

Every segment that arrives at the destination TCP is ultimately acknowledged by returning the sequence number of the next segment which must be passed to the process (it may not yet have arrived).

Earlier we described the use of a sequence number space and window to aid in duplicate detection. Acknowledgments are carried in the process header (see Fig. 6) and along with them there is provision for a "suggested window" which the receiver can use to control the flow of data from the sender. This is intended to be the main component of the process flow control mechanism. The receiver is free to vary the window size according to any algorithm it desires so long as the window size never exceeds half the sequence number space.[3]

This flow control mechanism is exceedingly powerful and flexible and does not suffer from synchronization troubles that may be encountered by incremental buffer allocation schemes [9],[10]. However, it relies heavily on an effective retransmission strategy. The receiver can reduce the window even while packets are en route from the sender whose window is presently larger. The net effect of this reduction will be that the receiver may discard incoming packets (they may be outside the window) and reiterate the current window size along with a current window edge as acknowledgment. By the same token, the sender can, upon occasion, choose to send more than a window's worth of data on the possibility that the receiver will expand the window to accept it (of course, the sender must not send more than half the sequence number space at any time). Normally, we would expect the sender to abide by the window limitation. Expansion of the window by the receiver merely allows more data to be accepted. For the receiving HOST with a small amount of buffer space, a strategy of discarding all packets whose sequence numbers do not coincide with the current left edge of the window is probably necessary, but it will incur the expense of extra delay and overhead for retransmission.

TCP INPUT/OUTPUT HANDLING

The TCP has a component which handles input/output (I/O) to and from the network.[4] When a packet has arrived, it validates the addresses and places the packet on a queue. A pool of buffers can be set up to handle arrivals, and if all available buffers are used up, succeeding arrivals can be discarded since unacknowledged packets will be retransmitted.

On output, a smaller amount of buffering is needed, since process buffers can hold the data to be transmitted. Perhaps double buffering will be adequate. We make no attempt to specify how the buffering should be done, except to require that it be able to service the network with as little overhead as possible. Packet sized buffers, one or more ring buffers, or any other combination are possible candidates.

When a packet arrives at the destination TCP, it is placed on a queue which the TCP services frequently. For example, the TCP could be interrupted when a queue placement occurs. The TCP then attempts to place the packet text into the proper place in the appropriate process receive buffer. If the packet terminates a segment, then it can be checksummed and acknowledged. Placement may fail for several reasons.

1) The destination process may not be prepared to receive from the stated source, or the destination port ID may not exist.

2) There may be insufficient buffer space for the text.

3) The beginning sequence number of the text may not coincide with the next sequence number to be delivered to the process (e.g., the packet has arrived out of order).

In the first case, the TCP should simply discard the packet (thus far, no provision has been made for error acknowledgments). In the second and third cases, the packet sequence number can be inspected to determine whether the packet text lies within the legitimate window for reception. If it does, the TCP may optionally keep the packet queued for later processing. If not, the TCP can discard the packet. In either case the TCP can optionally acknowledge with the current left window edge.

It may happen that the process receive buffer is not present in the active memory of the HOST, but is stored on secondary storage. If this is the case, the TCP can prompt the scheduler to bring in the appropriate buffer and the packet can be queued for later processing.

If there are no more input buffers available to the TCP for temporary queueing of incoming packets, and if the TCP cannot quickly use the arriving data (e.g., a TCP to TCP message), then the packet is discarded. Assuming a sensibly functioning system, no other processes than the one for which the packet was intended should be affected by this discarding. If the delayed processing queue grows

[4] This component can serve to handle other protocols whose associated control programs are designated by internetwork destination address.

excessively long, any packets in it can be safely discarded since none of them have yet been acknowledged. Congestion at the TCP level is flexibly handled owing to the robust retransmission and duplicate detection strategy.

TCP/PROCESS COMMUNICATION

In order to send a message, a process sets up its text in a buffer region in its own address space, inserts the requisite control information (described in the following list) in a transmit control block (TCB) and passes control to the TCP. The exact form of a TCB is not specified here, but it might take the form of a passed pointer, a pseudointerrupt, or various other forms. To receive a message in its address space, a process sets up a receive buffer, inserts the requisite control information in a receive control block (RCB) and again passes control to the TCP.

In some simple systems, the buffer space may in fact be provided by the TCP. For simplicity we assume that a ring buffer is used by each process, but other structures (e.g., buffer chaining) are not ruled out.

A possible format for the TCB is shown in Fig. 11. The TCB contains information necessary to allow the TCP to extract and send the process data. Some of the information might be implicitly known, but we are not concerned with that level of detail. The various fields in the TCB are described as follows.

1) *Source Address*: This is the full net/HOST/TCP/port address of the transmitter.

2) *Destination Address*: This is the full net/HOST/TCP/port of the receiver.

3) *Next Packet Sequence Number*: This is the sequence number to be used for the next packet the TCP will transmit from this port.

4) *Current Buffer Size*: This is the present size of the process transmit buffer.

5) *Next Write Position*: This is the address of the next position in the buffer at which the process can place new data for transmission.

6) *Next Read Position*: This is the address at which the TCP should begin reading to build the next segment for output.

7) *End Read Position*: This is the address at which the TCP should halt transmission. Initially 6) and 7) bound the message which the process wishes to transmit.

8) *Number of Retransmissions/Maximum Retransmissions*: These fields enable the TCP to keep track of the number of times it has retransmitted the data and could be omitted if the TCP is not to give up.

9) *Timeout/Flags*: The timeout field specifies the delay after which unacknowledged data should be retransmitted. The flag field is used for semaphores and other TCP/process synchronization, status reporting, etc.

10) *Current Acknowledgment/Window*: The current acknowledgment field identifies the first byte of data still unacknowledged by the destination TCP.

The read and write positions move circularly around the transmit buffer, with the write position always to the left (module the buffer size) of the read position.

The next packet sequence number should be constrained to be less than or equal to the sum of the current acknowledgment and the window fields. In any event, the next sequence number should not exceed the sum of the current acknowledgment and half of the maximum possible sequence number (to avoid confusing the receiver's duplicate detection algorithm). A possible buffer layout is shown in Fig. 12.

The RCB is substantially the same, except that the end read field is replaced by a partial segment check-sum register which permits the receiving TCP to compute and remember partial check sums in the event that a segment arrives in several packets. When the final packet of the segment arrives, the TCP can verify the check sum and if successful, acknowledge the segment.

CONNECTIONS AND ASSOCIATIONS

Much of the thinking about process-to-process communication in packet switched networks has been influenced by the ubiquitous telephone system. The HOST–HOST protocol for the ARPANET deals explicitly with the opening and closing of simplex connections between processes [9],[10]. Evidence has been presented that message-based "connection-free" protocols can be constructed [12], and this leads us to carefully examine the notion of a connection.

The term *connection* has a wide variety of meanings. It can refer to a physical or logical path between two entities, it can refer to the flow over the path, it can inferentially refer to an action associated with the setting up of a path, or it can refer to an association between two or more entities, with or without regard to any path between them. In this paper, we do not explicitly reject the term connection, since it is in such widespread use, and does connote a meaningful relation, but consider it exclusively in the sense of an association between two or more entities without regard to a path. To be more precise about our intent, we shall define the relationship between two or more ports that are in communication, or are prepared to communicate to be an *association*. Ports that are associated with each other are called *associates*.

It is clear that for any communication to take place between two processes, one must be able to address the other. The two important cases here are that the destination port may have a global and unchanging address or that it may be globally unique but dynamically reassigned. While in either case the sender may have to learn the destination address, given the destination name, only in the second instance is there a requirement for learning the address from the destination (or its representative) each time an association is desired. Only after the source has learned how to address the destination can an association be said to have occurred. But this is not yet sufficient. If

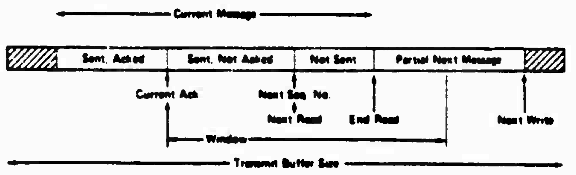


Fig. 12. Transmit buffer layout.

ordering of delivered messages is also desired, both TCP's must maintain sufficient information to allow proper sequencing. When this information is also present at both ends, then an association is said to have occurred.

Note that we have not said anything about a path, nor anything which implies that either end be aware of the condition of the other. Only when both partners are prepared to communicate with each other has an association occurred, and it is possible that neither partner may be able to verify that an association exists until some data flows between them.

CONNECTION-FREE PROTOCOLS WITH ASSOCIATIONS

In the ARPANET, the interface message processors (IMP's) do not have to open and close connections from source to destination. The reason for this is that connections are, in effect, always open, since the address of every source and destination is never[4] reassigned. When the name and the place are static and unchanging, it is only necessary to label a packet with source and destination to transmit it through the network. In our parlance, every source and destination forms an association.

In the case of processes, however, we find that port addresses are continually being used and reused. Some ever-present processes could be assigned fixed addresses which do not change (e.g., the logger process). If we supposed, however, that every TCP had an infinite supply of port addresses so that no old address would ever be reused, then any dynamically created port would be assigned the next unused address. In such an environment, there could never be any confusion by source and destination TCP as to the intended recipient or implied source of each message, and all ports would be associates.

Unfortunately, TCP's (or more properly, operating systems) tend not to have an infinite supply of internal port addresses. These internal addresses are reassigned after the demise of each port. Walden [12] suggests that a set of unique uniform external port addresses could be supplied by a central registry. A newly created port could apply to the central registry for an address which the central registry would guarantee to be unused by any HOST system in the network. Each TCP could maintain tables matching external names with internal ones, and use the external ones for communication with other processes. This idea violates the premise that interprocess communication should not require centralized control. One would have to extend the central registry service to include all HOST's in all the interconnected networks to apply this idea to our situation, and we therefore do not attempt to adopt it.

[4] Unless the IMP is physically moved to another site, or the HOST is connected to a different IMP.

Let us consider the situation from the standpoint of the TCP. In order to send or receive data for a given port, the TCP needs to set up a TCB and RCB and initialize the window size and left window edge for both. On the receive side, this task might even be delayed until the first packet destined for a given port arrives. By convention, the first packet should be marked so that the receiver will synchronize to the received sequence number.

On the send side, the first request to transmit could cause a TCB to be set up with some initial sequence number (say, zero) and an assumed window size. The receiving TCP can reject the packet if it wishes and notify the sending TCP of the correct window size via the acknowledgment mechanism, but only if either

1) we insist that the first packet be a complete segment;
2) an acknowledgment can be sent for the first packet (even if not a segment, as long as the acknowledgment specifies the next sequence number such that the source also understands that no bytes have been accepted).

It is apparent, therefore, that the synchronizing of window size and left window edge can be accomplished without what would ordinarily be called a connection setup.

The first packet referencing a newly created RCB sent from one associate to another can be marked with a bit which requests that the receiver synchronize his left window edge with the sequence number of the arriving packet (see SYN bit in Fig. 8). The TCP can examine the source and destination port addresses in the packet and in the RCB to decide whether to accept or ignore the request.

Provision should be made for a destination process to specify that it is willing to LISTEN to a specific port or "any" port. This last idea permits processes such as the logger process to accept data arriving from unspecified sources. This is purely a HOST matter, however.

The initial packet may contain data which can be stored or discarded by the destination, depending on the availability of destination buffer space at the time. In the other direction, acknowledgment is returned for receipt of data which also specifies the receiver's window size.

If the receiving TCP should want to reject the synchronization request, it merely transmits an acknowledgment carrying a release (REL) bit (see Fig. 8) indicating that the destination port address is unknown or inaccessible. The sending HOST waits for the acknowledgment (after accepting or rejecting the synchronization request) before sending the next message or segment. This rejection is quite different from a negative data acknowledgment. We do not have explicit negative acknowledgments. If no acknowledgment is returned, the sending HOST may retransmit without introducing confusion if, for example, the left window edge is not changed on the retransmission.

Because messages may be broken up into many packets for transmission or during transmission, it will be necessary to ignore the REL flag except in the case that the EM flag is also set. This could be accomplished either by the TCP or by the GATEWAY which could reset the flag off all but the packet containing the set EM flag (see Fig. 9).

At the end of an association, the TCP sends a packet with ES, EM, and REL flags set. The packet sequence number scheme will alert the receiving TCP if there are still outstanding packets in transit which have not yet arrived, so a premature dissociation cannot occur.

To assure that both TCP's are aware that the association has ended, we insist that the receiving TCP respond to the REL by sending a REL acknowledgment of its own.

Suppose now that a process sends a single message to an associate including an REL along with the data. Assuming an RCB has been prepared for the receiving TCP to accept the data, the TCP will accumulate the incoming packets until the one marked ES, EM, REL arrives, at which point a REL is returned to the sender. The association is thereby terminated and the appropriate TCB and RCB are destroyed. If the first packet of a message contains a SYN request bit and the last packet contains ES, EM, and REL bits, then data will flow "one message at a time." This mode is very similar to the scheme described by Walden [12], since each succeeding message can only be accepted at the receiver after a new LISTEN (like Walden's RECEIVE) command is issued by the receiving process to its serving TCP. Note that only if the acknowledgment is received by the sender can the association be terminated properly. It has been pointed out[4] that the receiver may erroneously accept duplicate transmissions if the sender does not receive the acknowledgment. This may happen if the sender transmits a duplicate message with the SYN and REL bits set and the destination has already destroyed any record of the previous transmission. One way of preventing this problem is to destroy the record of the association at the destination only after some known and suitably chosen timeout. However, this implies that a new association with the same source and destination port identifiers could not be established until this timeout had expired. This problem can occur even with sequences of messages whose SYN and REL bits are separated into different internetwork packets. We recognize that this problem must be solved, but do not go into further detail here.

Alternatively, both processes can send one message, causing the respective TCP's to allocate RCB TCB pairs at both ends which rendezvous with the exchanged data and then disappear. If the overhead of creating and destroying RCB's and TCB's is small, such a protocol might be adequate for most low-bandwidth uses. This idea might also form the basis for a relatively secure transmission system. If the communicating processes agree to change their external port addresses in some way known only to each other (i.e., pseudorandom), then each message will appear to the outside world as if it is part of a different association message stream. Even if the data is intercepted by a third party, he will have no way of knowing that the data should in fact be considered part of a sequence of messages.

[4] S. Crocker of ARPA IPT.

We have described the way in which processes develop associations with each other, thereby becoming associates for possible exchange of data. These associations need not involve the transmission of data prior to their formation and indeed two associates need not be able to determine that they are associates until they attempt to communicate.

CONCLUSIONS

We have discussed some fundamental issues related to the interconnection of packet switching networks. In particular, we have described a simple but very powerful and flexible protocol which provides for variation in individual network packet sizes, transmission failures, sequencing, flow control, and the creation and destruction of process-to-process associations. We have considered some of the implementation issues that arise and found that the proposed protocol is implementable by HOST's of widely varying capacity.

The next important step is to produce a detailed specification of the protocol so that some initial experiments with it can be performed. These experiments are needed to determine some of the operational parameters (e.g., how often and how far out of order do packets actually arrive; what sort of delay is there between segment acknowledgments; what should be retransmission timeouts be?) of the proposed protocol.

ACKNOWLEDGMENT

The authors wish to thank a number of colleagues for helpful comments during early discussions of international network protocols, especially R. Metcalfe, R. Scantlebury, D. Walden, and H. Zimmerman; D. Davies and L. Pouzin who constructively commented on the fragmentation and accounting issues; and S. Crocker who commented on the creation and destruction of associations.

REFERENCES

[1] L. Roberts and B. Wessler, "Computer network development to achieve resource sharing," in *1970 Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 543–549.
[2] L. Pouzin, "Presentation and major design aspects of the CYCLADES computer network," in *Proc. 3rd Data Communications Symp.*, 1973.
[3] F. R. E. Dell, "Features of a proposed synchronous data network," in *Proc. 2nd Symp. Problems in the Optimization of Data Communications Systems*, 1971, pp. 50–57.

[4] R. A. Scantlebury and P. T. Wilkinson, "The design of a switching system to allow remote access to computer services by other computers and terminal devices," in *Proc. 2nd Symp. Problems in the Optimization of Data Communications Systems*, 1971, pp. 160-167.
[5] D. L. A. Barber, "The European computer network project," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 192-200.
[6] R. Despres, "A packet switching network with graceful saturated operation," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 345-351.
[7] R. E. Kahn and W. R. Crowther, "Flow control in a resource-sharing computer network," *IEEE Trans. Commun.*, vol. COM-20, pp. 539-546, June 1972.
[8] J. F. Chambon, M. Elie, J. Le Bihan, G. LeLann, and H. Zimmerman, "Functional specification of transmission station in the CYCLADES network. ST-ST protocol" (in French), I.R.I.A. Tech. Rep. SCH502.3, May 1973.
[9] S. Carr, S. Crocker, and V. Cerf, "HOST-HOST Communication Protocol In the ARPA Network," in *Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 589-597.
[10] A. McKenzie, "HOST/HOST protocol for the ARPA network," in *Current Network Protocols*, Network Information Cen., Menlo Park, Calif., NIC 8246, Jan. 1972.
[11] L. Pouzin, "Address format in Mitranet," NIC 14497, INWG 20, Jan. 1973.
[12] D. Walden, "A system for interprocess communication in a resource sharing computer network," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 221-230, Apr. 1972.
[13] B. Lampson, "A scheduling philosophy for multiprocessing systems," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 347-360, May 1968.
[14] F. E. Heart, R. E. Kahn, S. Ornstein, W. Crowther, and D. Walden, "The interface message processor for the ARPA computer network," in *Proc. Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970, pp. 551-567.
[15] N. G. Anslow and J. Hanscoff, "Implementation of international data exchange networks," in *Computer Communications: Impacts and Implications*, S. Winkler, Ed. Washington, D. C., 1972, pp. 181-184
[16] A. McKenzie, "HOST/HOST protocol design considerations," INWG Note 16, NIC 13879, Jan. 1973.
[17] R. E. Kahn, "Resource-sharing computer communication networks," *Proc. IEEE*, vol. 60, pp. 1397-1407, Nov. 1972.
[18] Bolt, Beranek, and Newman, "Specification for the interconnection of a host and an IMP," Bolt Beranek and Newman, Inc., Cambridge, Mass., BBN Rep. 1822 (revised), Apr. 1973.

Vinton G. Cerf was born in New Haven, Conn., in 1943. He did undergraduate work in mathematics at Stanford University, Stanford, Calif., and received the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, Calif., in 1972.

He was with IBM in Los Angeles from 1965 through 1967 and consulted and/or worked part time at UCLA from 1967 through 1972. Currently he is Assistant Professor of Computer Science and Electrical Engineering at Stanford University, and consultant to Cabledata Associates. Most of his current research is supported by the Defense Advanced Research Projects Agency and by the National Science Foundation on the technology and economics of computer networking. He is Chairman of IFIP TC6.1, an international network working group which is studying the problem of packet network interconnection.

★

Robert E. Kahn (M'65) was born in Brooklyn, N. Y., on December 23, 1938. He received the B.E.E. degree from the City College of New York, New York, in 1960, and the M.A. and Ph.D. degrees from Princeton University, Princeton, N. J., in 1962 and 1964, respectively.

From 1960 to 1962 he was a Member of the Technical Staff of Bell Telephone Laboratories, Murray Hill, N. J., engaged in traffic and communication studies. From 1964 to 1966 he was a Ford Postdoctoral Fellow and an Assistant Professor of Electrical Engineering at the Massachusetts Institute of Technology, Cambridge, where he worked on communications and information theory. From 1966 to 1972 he was a Senior Scientist at Bolt Beranek and Newman, Inc., Cambridge, Mass., where he worked on computer communications network design and techniques for distributed computation. Since 1972 he has been with the Advanced Research Projects Agency, Department of Defense, Arlington, Va.

Dr. Kahn is a member of Tau Beta Pi, Sigma Xi, Eta Kappa Nu, the Institute of Mathematical Statistics, and the Mathematical Association of America. He was selected to serve as a National Lecturer for the Association for Computing Machinery in 1972.

Reprinted by permission from
IEEE TRANSACTIONS ON COMMUNICATIONS
Vol. COM-22, No. 5, May 1974

Issues in Packet-Network Interconnection

VINTON G. CERF AND PETER T. KIRSTEIN

Invited Paper

Abstract—This paper introduces the wide range of technical, legal, and political issues associated with the interconnection of packet-switched data communication networks. Motivations for interconnection are given, desired user services are described, and a range of technical choices for achieving interconnection are compared. Issues such as the level of interconnection, the role of gateways, naming and addressing, flow and congestion control, accounting and access control, and basic internet services are discussed in detail. The CCITT X.25/X.75 packet-network interface recommendations are evaluated in terms of their applicability to network interconnection. Alternatives such as datagram operation and general host gateways are compared with the virtual circuit methods. Some observations on the regulatory aspects of interconnection are offered and the paper concludes with a statement of open research problems and some tentative conclusions.

I. Introduction

IT IS THE THEME of many papers in this issue, that people need access to data resources. In many cases this access must be over large distances, in others it may be local to a building or a single site. Data networks have been set up to meet many user needs—often, but not necessarily, using packet-switching technology. For single organizations, these data networks are often private ones, built with a technology optimized to the specific application. For communication between organizations, these networks are being set up by licensed carriers. In North America, there are many such licensed carriers, e.g., TELENET [1], DATAPAC [2], and TYMNET [3]. In the rest of the world, the Post, Telegraph, and Telephone Authority (PTT) in each country has a near monopoly on such services, special public data networks being set up in these countries include TRANSPAC [5] in France, EURONET [6] for inter-European traffic, DDX [7] in Japan, EDS [8] in the Federal Republic of Germany, and the Nordic Public Data Network (NPDN, [9]) in Scandinavia. These public data networks are considered in greater detail in other references (e.g., [10]-[12]). Most of the above networks use packet-switching technology, some of them, e.g., EDS and the NPDN, do not do so yet, but may do so in the future. In some cases special data networks have been authorized for specific communities, e.g., SITA [13] for the airlines, and SWIFT [14] for the banks. In addition many private networks have been set up among individual organizations, and experimental networks of different technologies have been developed also, e.g., ARPANET [15], [16], CYCLADES [17], ETHERNET [18], SPYDER [19], PRNET [20], [21] and SATNET [22].

Manuscript received June 30, 1978, revised July 31, 1978.

V. G. Cerf is with the Advanced Research Projects Agency, U.S. Department of Defense, Arlington, VA 22209.
P. T. Kirstein is with the Department of Statistics and Computer Science, University College, London, England.

It is a common user requirement that a single terminal and access port should be able to access any computing resource the user may desire—even if the resource is on another data network. From this requirement, there is a clear user need to have data networks connected together. By the same token, the providers of data network services would like to have their networks used as intensively as possible; thus they also have a strong motivation to connect their data networks to others. As a result of these considerations, there has been a high recent interest in the issues arising in the connection of data networks [23]-[26], [32].

From the user viewpoint, the requirement for interconnection of data networks is independent of the network technology. From the implementation viewpoint, there can be some considerable complications in connecting networks of widely different technologies—such as circuit-switched and datagram packet-switched networks (these terms are explained below). On the whole we will consider only, in this paper, the interconnection of packet-switched data networks. In many cases, however, the arguments will be equally valid for the interconnection of packet-switched to circuit-switched networks.

Network interconnection raises a great many technical, legal, and political questions and issues. The technical issues generally revolve around mechanisms for achieving interconnection and their performance. How can networks be interconnected so that packets can flow in a controllable way from one net to another? Should all computer systems on all nets be able to communicate with each other? How can this be achieved? What kind of performance can be achieved with a set of interconnected networks of widely varying internal design and operating characteristics? How are terminals to be given access to resources in other networks? What protocols are required to achieve this? Should the protocols of one net be translated into those of another, or should common protocols be defined? What kinds of communication protocol standards are needed to support efficient and useful interconnection? Who should take responsibility for setting standards?

The legal and political issues are at least as complex as the technical ones. Can private networks interconnect to each other or must they do so through the mediation of a public network? How is privacy to be protected? Should there be control over the kinds of data which move from one net to another? Are there international agreements and conventions which might be affected by international interconnection of data networks? What kinds of charging and accounting policies should apply to multinetwork traffic? How can faults and errors be diagnosed in a multinet environment? Who should be responsible for correcting such faults? Who should be responsible for maintaining the gateways which connect nets together?

We cannot possibly answer all of these questions in this paper, but we deal with many of them in the sections below.

This paper is divided into eleven sections. In the next section we provide some definitions, and in Section III we explore some of the motivations for network interconnection. In Section IV we discuss the range of end-user service requirements and choices for providing multinetwork service. Section V reviews the concept of computer-communication protocol layering. Section VI reviews the basic interconnection choices and introduces the concept of gateways between nets, protocol translation and the impact of common protocols; it elaborates also on the function of gateways. Section VII discusses the CCITT recommendations X.25 and X.75 and their role in network interconnection. Section VIII describes some of the network interconnections achieved and some of the experiments in progress. Section IX outlines regulatory issues raised by network interconnection alternatives. Section X mentions some unresolved research questions, and the final section offers some tentative conclusions on network interconnection issues.

II. The Definition of Terms

The vocabulary of networking is extensive and not always consistent. We introduce some generic terms below which we will use in this paper for purposes of discussion. It is important for the reader not to make any *a priori* assumptions about the physical realization of the objects named or of the boundary of jurisdictions owning or managing them. For instance, a gateway (see below) might be implemented to share the hardware of a packet switch and be owned by a packet-switching service carrier; alternatively it might be embedded in a host computer which subscribes to service on two or more computer networks. Roughly speaking, we are assigning names to groups of functions which may or may not be realized as physically distinct entities.

Packet: A packet of information is a finite sequence of bits, divided into a control header part and a data part. The header will contain enough information for the packet to be routed to its destination. There will usually be some checks on each such packet, so that any switch through which the packet passes may exercise error control. Packets are generally associated with internal packet-network operation and are not necessarily visible to host computers attached to the network.

Datagram: A finite length packet of data together with destination host address information (and, usually, source address) which can be exchanged in its entirety between hosts, independent of all other datagrams sent through a packet switched network. Typically, the maximum length of a datagram lies between 1000 and 8000 bits.

Gateway: The collection of hardware and software required to effect the interconnection of two or more data networks, enabling the passage of user data from one to another.

Host: The collection of hardware and software which utilizes the basic packet-switching service to support end-to-end interprocess communication and user services.

Packet Switch: The collection of hardware and software resources which implements all intranetwork procedures such as routing, resource allocation, and error control and provides access to network packet-switching services through a host/network interface.

Protocol: A set of communication conventions, including formats and procedures which allow two or more end points to communicate. The end points may be packet switches, hosts, terminals, people, file systems, etc.

Protocol Translator: A collection of software, and possibly hardware, required to convert the high level protocols used in one network to those used in another.

Terminal: A collection of hardware and possibly software which may be as simple as a character-mode teletype or as complex as a full scale computer system. As terminals increase in capability, the distinction between "host" and "terminal" may become a matter of nomenclature without technical substance.

Virtual Circuit: A logical channel between source and destination packet switches in a packet-switched network. A virtual circuit requires some form of "setup" which may or may not be visible to the subscriber. Packets sent on a virtual circuit are delivered in the order sent, but with varying delay.

PTT: Technically PTT stands for Post, Telegraph, and Telephone Authority; this authority has a different form in different countries. In this paper, by PTT we mean merely the authority (or authorities) licensed in each country to offer public data transmission services.

We have attempted to make these definitions as noncontroversial as possible. For example, in the definition of packet switch, we alluded to a host/network interface. The reader should not assume that subscriber services are limited to those offered through the host/network interface. The packet-switching carrier might also offer host-based services and terminal access mechanisms as additional subscriber services.

III. The Motivating Forces in the Interconnection of Data Networks

In the introduction, we mentioned that there was a strong interest, among both the users and suppliers of data serivces, in the interconnection of data networks. However, the technical interests of the different parties are not identical. The end user would merely like to be able to access any resources from a single terminal, with a single access port, as economically as possible according to his own performance criteria. A Public Carrier, or PTT, has a strong motivation to connect its network to other PTT's. As in the telephone system, the concept of all subscribers being accessible through a single Public Data Service, is considered highly desirable; however the different PTT's may have restricted geographic coverage, or only a specific market penetration.

The motivation of the PTT's to interface to private networks is weaker and more complex. They always provide facilities to attach single terminals, where a terminal may be a complex computer system; they are often not interested, at present, in making any special arrangements when the "terminal" is a whole computer network. The operators of private networks often have a vital interest in connecting their networks to other private networks and to the public ones. Even though in many cases the bulk of its traffic is internal to the private network, which is why it was set up in the first place, there is usually a vital need to access resources not available on that network. The regulatory limitations often imposed on the method of interconnection of private networks are discussed in Section IX. In some countries, it is not permitted to build private networks using leased line services, but intrabuilding networks may be permitted. Interconnection of such local networks to public networks may play a crucial role in making the local network useful.

To date the PTT's have tried to standardize on access procedures for their Public-Packet Data Services. The standardization has taken place in the International Consultative Committee on Telegraphy and Telephony (called CCITT) in a set of recommendations called X.3, X.25, X.28, and X.29 ([27]-[29]). Not all PTT's have such forms of access yet, but most of the industrialized nations in the West are moving in this direction. This series of recommendations is discussed in much more detail in Section VI; it does not pay special attention to the attachment of private networks ([31], [32]), but the recommendations are themselves expected to change to meet this requirement. The PTT's are agreeing on a set of interface recommendations and procedures called X.75 [33], to connect their networks to each other; so far this interface procedure (and its corresponding hardware) is not intended to be provided to private networks.

While most PTT's have preferred to ignore the technical implications of the attachment of private networks to the public ones, most private network operators cannot ignore this requirement. They are often motivated to add some extra "Foreign Exchange" capability as an afterthought, with minimum change to their intranetwork procedures; this approach can be successful up to a point, but will usually be limited by the lack of high-level procedures between the different networks. These high-level procedures have not yet been considered by CCITT, but it has been proposed that CCITT Study Group VII investigate high-level procedures and architectural models, in cooperation with the investigation of "open system architectures" by Technical Committee 97, Sub-Committee 16 of the International Standards Organisation (ISO). This subject is also considered later in this paper, in Section VI.

An aim of these standardization exercises is to ensure that both manufacturer and user implementations of network resources can communicate with each other through single private or public data networks. A consequence should be that the resources are also compatibly accessible over connected data networks.

Depending on the applications and spatial distribution of subscribers, the preferred choice of packet-switching medium will vary. Intrabuilding applications such as electronic office services may be most economically provided through the use of a coaxial-packet cable system such as the Xerox ETHERNET [18] and LCSNET [64], or twisted pair rings such as DCS [34], coupled with a mix of self-contained user computers (e.g., intelligent terminals with substantial computing and memory capacity) and shared computing, storage, and input-output facilities. Larger area regional applications might best employ shared video cables [35] or packet radios [20], [21] for mobile use. National systems might be composed of a mixture of domestic satellite channels and conventional leased-line services. International systems might use point-to-point links plus a shared communication satellite channel and multiple ground stations to achieve the most cost-effective service.

A consequence of the wide range of technologies which are optimum for different packet-switching applications is that many different networks, both private and public, may co-exist. A network interconnection strategy, if properly designed, will permit local networks to be optimized without sacrificing the possibility of providing effective internetwork services. The potential economic and functional advantages of local networks such as ETHERNET or DCS will lead naturally to private user networks. Such private network developments are analogous to telephone network private automated branch exchanges (PABX) and represent a natural consequence of the marriage of computer and telecommunication technology.

Two further developments can be expected. First, organizations which are dispersed geographically, nationally, or internationally, will want to interconnect these private networks both to share centralized resources and to effect intraorganization electronic mail and other automated office services. Second, there will be an increasing interest in interorganization interconnections to allow automated procurement and financial transaction services, for example, to be applied to interorganization affairs.

In most countries where private networks are permitted, interorganization telecommunication requires the involvement of a PTT. Hence the most typical network interconnection scenarios will involve three or four networks. Within one national administration the private nets of different organizations will be interconnected through a public network. International interconnections will involve at least two public networks. We will return to this topic in Section VI.

In addition to permitting locally optimized networks to be interconnected, a network interconnection strategy should also support the gradual introduction of new networking technology into existing systems without requiring simultaneous global change throughout. This consideration leads to the conclusion that the public data networks should support the most important user requirements for internet service from the outset. If this were the case, then changes in network technology which require a multinetwork system during phased transition would not, *a priori*, have to affect user services.

IV. Provision of End-User Multinetwork Services

The ultimate choice of a network interconnection strategy will be strongly affected by the types of user services which must be supported. It is useful to consider the range of existing and foreseeable user service requirements without regard for the precise means by which these requirements are to be met. We will leave for discussion in subsequent sections the choice of supporting the various services within or external to the packet-switched network. The types of service discussed below are general requirements for network facilities. For this reason they also should be supported across interconnected networks.

Most of the currently prevalent computer-communication services fall into four categories:

1) terminal access to time-shared host computers;
2) remote job entry services (RJE);
3) bulk data transfer;
4) transaction processing.

The time-sharing and transaction services typically demand short network and host response times but modest bandwidth. The RJE and file transfer services more often require high amounts of data transfer, but can tolerate longer delay. Some networks were designed to support primarily terminal service, leaving RJE or file transfer services to be supported by dedicated leased lines. Packet-switching techniques permit both types of service to be supported with common network resources, leading to verifiable economies. However, bulk data transfer requires increasingly higher throughput rates if delivery delays are to be kept constant as the amount of data to be transferred increases.

As distributed operating systems become more prevalent, there will be an increased need for host-to-host transaction services. A prototypical example of such a system is found in the DARPA National Software Works [4], [36]. In such a system, small quantities of control information must be exchanged quickly to coordinate the activity of the distributed components. Broadcast or multidestination services will be needed to support distributed file systems in which information can be stored redundantly to improve the reliability of access and to protect against catastrophic failures.

Transaction services are also finding application in reservation systems, credit verification, point of sale, and electronic funds-transfer systems in which hundreds or thousands of terminals supply to, or request of, hosts small amounts of information at random intervals. Real-time data collection for weather analysis, ground and air traffic control, and meter reading, for example, also fall into this category.

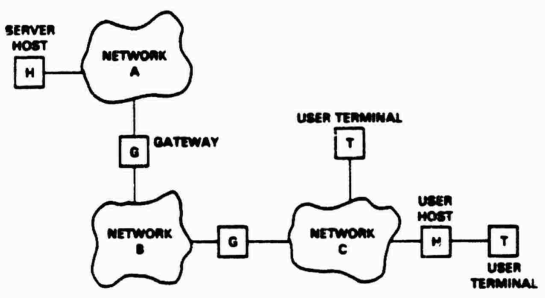


Fig. 1. Network concatenation.

More elaborate user requirements can be foreseen as electronic mail facilities propagate. Multiple destination addressing and end-to-end encryption for the protection of privacy as well as support for text, digitized voice, and facsimile message transmission are all likely requirements. Electronic teleconferencing using mixtures of compressed digital packet speech, videographics, real-time cursors (for pointing at video images under discussion), and text display will give rise to requirements for closed user groups and time-synchronized mixes of transaction-like (e.g., for cursor tracking and packet speech) and reliable circuit-like services (e.g., for display management).

Reliability and rapid response will be increasingly important as more and more computer-based applications requiring telecommunications are integrated into the business, government, military, and social fabric of the world economy. The more such systems are incorporated into their daily activities, the more vulnerable the subscribers are to failures. Reliability concerns lead to the requirement for redundant alternatives such as distributed file systems, richly connected networks, and substantial local processing and storage capability. These trends increase the need for networking to share common hardware and software resources (and thus reduce their marginal cost), to support remote software maintenance and debugging, and to support intra- and inter-organizational information exchange.

We have described the end-user services required across one or more data networks. We have carefully refrained from discussing which services should be provided in the data network, and which should be provided in the hosts. Here the choice in single networks will depend on the network technology and the application requirements. For example, in a network using a broadcast technology such as ETHERNET or the SATNET, multidestination facilities may well be incorporated in the data network itself. In typical store-and-forward networks, this feature might be provided at the host level by the transmission of multiple copies of packets. This example highlights immediately the difficulty of using sophisticated services at the data network level across concatenated networks. If *A*, *B*, and *C* are data networks connected as in Fig. 1, and *A* and *C* but not *B* support broadcast or real-time features, it is very difficult to provide them across the concatenation of *A*, *B*, and *C*.

The problem of achieving a useful set of internetwork services might be approached in several ways, as follows.

1) Require all networks to implement the entire range of desired services (e.g., datagram, virtual circuit, broadcast, real-

time, etc.), and then attempt to support these services across the gateways between the networks.

2) Require all networks to implement only the most basic services (e.g., datagram or virtual circuit), support these services across gateways, and rely on the subscriber to implement all other services end-to-end.

3) Allow the subscriber to identify the services which he desires and provide error indications if the networks involved, or the gateways between them, cannot provide the desired services.

4) Allow the subscriber to specify the internetwork route to be followed and depend on the subscriber to decide which concatenation of services are appropriate and what end-to-end protocols are needed to achieve the ultimately preferred class of service.

5) Provide one set of services for local use within each network and another, possibly different set for internetwork use.

The five choices above are by no means exhaustive, and, in fact, only scratch the surface of possibilities. Nothing has been said, thus far, about the compatibility of various levels of communication protocols which exist within each network, within subscriber equipments, and within the logical gateway between networks. To explore these issues further, it will be helpful to have a model of internetwork architecture, taking into account the common principle of protocol layering and the various possible choices of interconnection strategy which depend upon the protocol layer at which the networks are interfaced. We consider this in the next section.

V. Layered Protocol Concepts

Both to provide services in single networks, and to compare the capabilities of different networks, a very useful concept in networking is protocol layering. Various services of increasing capability can be built one on top of the other, each using the facilities of the service layer below and supporting the facilities of the layer above. A thorough tutorial on this concept can be found in the paper by Pouzin and Zimmermann in this issue [37]. We give some specific examples below of layering as a means of illustrating the scope of services and interfaces to be found in packet networks today—and some of the problems encountered in offering services across multiple networks.

Table I offers a very generic view of a typical protocol hierarchy in a store-and-forward computer network, including layers usually found outside of the communication network itself. There are several complications to the use of generic protocol layering to study network interconnection issues. Chief among these is that networks do not all contain the same elements of the generic hierarchy. A second complication is that some networks implement service functions at different protocol layers. For instance, virtual circuit networks implement an end/end subscriber virtual circuit in their intranet, end/end level protocol. Finally, the hierarchical ordering of functions is not always the same in all networks. For instance, TYMNET places a terminal handling protocol within the network access layer, so that hosts look to each other like one or more terminals. Figs. 2-7 illustrate the functional layering of some different networks. It is important to note how the functions vary with the choice of transmission medium.

TABLE I
Generic Protocol Layers

PROTOCOL LAYER	FUNCTIONS
6. APPLICATION	FUNDS TRANSFER, INFORMATION RETRIEVAL, ELECTRONIC MAIL, TEXT EDITING . . .
5. UTILITY	FILE TRANSFER, VIRTUAL TERMINAL SUPPORT
4. END/END SUBSCRIBER	INTERPROCESS COMMUNICATION (E.G. VIRTUAL CIRCUIT, DATAGRAM, REAL-TIME, BROADCAST)
3. NETWORK ACCESS	NETWORK ACCESS SERVICES (E.G. VIRTUAL CIRCUIT, DATAGRAM . . .)
2. INTRANET, END-TO-END	FLOW CONTROL, SEQUENCING
1. INTRANET, NODE-TO-NODE	CONGESTION CONTROL, ROUTING
0. LINK CONTROL	ERROR HANDLING, LINK FLOW CONTROL

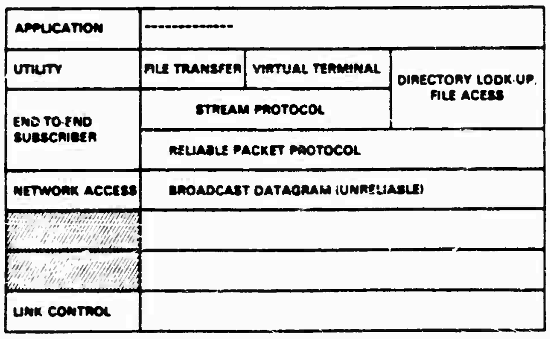


Fig. 2. ETHERNET protocol layering.

A. ETHERNET

In Fig. 2, we represent the Xerox ETHERNET protocol hierarchy. The basic link control mechanism is the ability of the interface device to detect conflict on a shared coaxial cable. If a transmitting interface detects that another interface is also transmitting, it immediately aborts the transmission. Hosts attached to the network interface present datagrams to be transmitted and are told if the datagram was aborted. Datagrams can be addressed to specific interfaces or to all of them. The end/end subscriber layer of protocol is split into two parts: a reliable datagram protocol in which each datagram is reliably delivered and separately acknowledged, and a stream protocol which can be thought of as a virtual circuit. This split is possible, in part, because there is a fairly large maximum datagram size (about 500 bytes) so that user applications can send datagrams without having to fragment and reassemble them. This makes the datagram service useful for many applications which might otherwise have to use the stream protocol. All higher level protocols, such as Virtual Terminal and File Transfer, are carried out in the hosts.

B. ARPANET

The ARPANET protocol hierarchy is shown in Fig. 3. The basic link control between packet switches treats the physical link as eight independent virtual links. This increases effective throughput, but does not necessarily preserve the order in which packets were originally introduced into the network. The intranet node-to-node protocols deal with adaptive routing decisions, store-and-forward service and congestion control. Hosts have the option of either passing messages (up to

APPLICATION	RJE	ELECTRONIC MAIL	
UTILITY	TELNET	FTP	
END/END SUBSCRIBER	NCP	TCP	NVP/NVCP
NETWORK ACCESS	PERMANENT VIRTUAL CIRCUIT		DATAGRAM
INTRANET, END/END	FLOW CONTROL, SEQUENCING, MESSAGE REASSEMBLY		▨
INTRANET, NODE/NODE	ADAPTIVE ROUTING, STORE AND FORWARD, CONGESTION CONTROL		
LINK CONTROL	NON-SEQUENCED, MULTI-CHANNEL ERROR CONTROL		

Fig. 3. ARPANET protocol layering.

END/END SUBSCRIBER	TERMINAL-TO-HOST
NETWORK ACCESS	VIRTUAL CIRCUIT
INTRANET END-END	▨
INTRANET NODE-NODE	FRAME DISASSEMBLY, REASSEMBLY, ROUTING, STORE/FORWARD, CONGESTION CONTROL
LINK CONTROL	FRAME-BASED ERROR CONTROL, RETRANSMISSION, SEQUENCING

Fig. 4. TYMNET protocol layering.

8063 bits of text) across the host/network interface, which will be delivered in sequence to the destination, or passing datagrams (up to 1008 bits of text) which are not necessarily delivered in sequence. The user's network access interface is datagram-like in the sense that no circuit setup exchange is needed even to activate the sequenced message service. In effect, this service acts like a permanent virtual circuit over which a sequence of discrete messages are sent. For the sequenced messages, there is exactly one virtual circuit maintained for each host/host pair. In fact, these virtual circuits are set up dynamically and terminated by the source/destination packet switches so as to improve resource utilization [38], [62].

The end/end subscriber layer of ARPANET contains two main protocols: Network Control Protocol (NCP, [39], [40]) and Transmission Control Protocol (TCP, [25]). NCP was the first interprocess communication protocol built for ARPANET. It relies on the sequenced message service provided by the network and derives multiple virtual circuits between pairs of hosts by multiplexing. The TCP can use either the sequenced message service or the datagram service. It does its own sequencing and end/end error control and derives multiple virtual circuits through extended addressing and multiplexing. TCP was designed for operation in a multinet environment in which the only service which reasonably could be expected was an unreliable, unsequenced datagram service.

To support experiments in packetized voice communication, two protocols were developed for use on the ARPANET. The Network Voice Protocol (NVP) and Network Voice Conferencing Protocol (NVCP) use the datagram service to achieve very low delay and interarrival time variance in support of digital, compressed packet speech (more on these protocols may be found in [41]). The NVP could be considered the basis for a generic protocol which could support a variety of real-time, end/end user applications.

The higher level utility protocols such as terminal/host protocol (TELNET, [40], [42]) and file transfer protocol (FTP, [40], [42]) use virtual circuits provided by NCP or TCP. The FTP requires one live interactive stream to control the data transfer, and a second for the data stream itself. Yet higher level applications such as electronic mail and remote job entry (RJE, [40], [42]) use mixtures of TELNET and FTP to effect the service desired. These protocols are usually put into the hosts. There is one anomaly, which occurs in many networks. Because terminal handling is required so frequently, a Terminal Interface Message Processor (TIP, [43]) was built. This device is physically integrated with the packet switch (IMP, [38]); it includes also the NCP and TELNET protocols.

C. TYMNET

TYMNET (see Fig. 4) is one of the oldest of the networks in the collection described here [3]. Strictly speaking, it operates rather differently than other packet-switched networks, because the frames of data that move from switch to switch are disassembled and reassembled in each switch as an integral part of the store-and-forward operation. Nevertheless, the network benefits from the asynchronous sharing of the circuits between the switches in much the same way that more typical packet-switched networks do. The network was designed to support remote terminal access to time-shared computer resources. The basic service is the transmission of a stream of characters between the terminal and the serving host. A frame is made up of one or more blocks of characters, each block labeled with its source terminal identifier and length. The switch-to-switch layer of protocol disassembles each frame into its constituent blocks and uses a routing table to determine to which next switch the block should be sent. Blocks destined for the same next switch are batched together in a frame which is checksummed and sent via the link control procedure to the next switch. Batching the blocks reduces line overhead (the blocks share the frame checksum) at the expense of more CPU cycles in the switch for frame disassembly and reassembly.

The protocol between TYMNET switches also includes a flow control mechanism which, because of the fixed routes, can be used to apply back pressure all the way back to the traffic source. This is not precisely an end-to-end flow control mechanism, but a hop-by-hop back pressure strategy. Character blocks are kept in sequence along the fixed routes so that no resequencing is required as they exit from the network at their destinations. The network interface is basically a virtual circuit designed to transport character streams between a host and a terminal. The same virtual circuits can be used to transport character streams between hosts, which look to each other like a collection of terminals. Above the basic virtual circuit service, is a special echo-handling protocol which allows the host and the terminal handler in the "remote TYMSAT" to coordinate the echoing of the characters typed by a user.

D. PTT Networks

Many PTT networks, e.g., TELNET, TRANSPAC, DATAPAC, and EURONET use a particular network-access protocol, X.25 [28], [29] (see Fig. 5). This protocol has been recommended by the CCITT for public packet-switched data networks. X.25 is a three-part protocol consisting of a hardware electrical interface, X.21 [44], the digital equivalent of the usual V.24 or EIA-RS232C modem interface [45], a link control procedure, High Level Data Link Control (HDLC, [46]), and a packet-level protocol for effecting the setup, use, termination, flow, and error control of virtual circuits.

UTILITY	TERMINAL HANDLING X.28, X.29
END/END SUBSCRIBER	▨
NETWORK ACCESS	X.25, PERMANENT OR TEMPORARY VIRTUAL CIRCUITS
INTRANET, END END	MULTIPLE VIRTUAL CIRCUITS, FLOW CONTROL
INTRANET NODE NODE	ROUTING, STORE/FORWARD, CONGESTION CONTROL
LINK CONTROL	HDLC OR EQUIVALENT

Fig. 5. PTT protocol layering.

In all but the DATAPAC network, a fixed route for routing packets through the network is selected at the time the virtual circuit is created. "Permanent" virtual circuits are a customer option; if used, the setup phase is invoked only in the case of a network failure. Between source and destination packet switches, a virtual circuit protocol is operated which implements end-to-end flow control on multiple virtual circuits between pairs of packet switches. Up to 4096 virtual circuits between pairs of host ports can be maintained by each packet switch, as compared to the single virtual circuit provided by ARPANET (on which hosts can multiplex their own virtual circuits). This choice has a noticeable impact on the subscriber interface protocol which becomes complicated because the subscriber host and the packet switch to which it attaches must maintain a consistent view of the state of each virtual circuit in use.

To provide for echo control, user commands, code conversion, and other terminal-related services, these networks implement CCITT Recommendations X.28 [29] and X.29 [29] in a PAD (Packet Assembly and Disassembly unit). These protocols sit atop the virtual circuit X.25 protocol. In order to serve customers desiring a terminal-to-host service with character terminals, such as is provided by TYMNET or by the ARPANET (through the TIP), most of the PTT networks mentioned are developing a PAD unit. A matching X.29 (PAD control protocol) layer must be provided in hosts offering to service terminals connected to PAD's.

E. High Level Protocols

The X.25/X.28/X.29 protocol hierarchy does not include an end/end subscriber or high-level protocol layer. Some customers will, in fact, implement end-to-end protocols on top of the virtual circuit protocol, but others may not. Several attempts are being made to standardize protocols above the network access level. The ARPANET community has developed a Transmission Control Protocol [25] for internetwork operation to replace the Network Control Program (NCP) developed early in the ARPANET project. The International Federation of Information Processing (IFIP) has proposed a Transport Station through its Working Group 6.1 on Network Interconnection [47], the proposal has been submitted to the International Standards Organisation (ISO) as a draft standard. In addition, other communities, e.g., the High Level Protocol Working Group in the UK, have devised protocols for Virtual Packet Terminals (VPT, [48]) and File Transport Protocol (FTP, [49]) which are intended to be network independent and which may be submitted to CCITT. The ISO study on "open systems architecture" and the proposed similar study by CCITT Study Group VII will attempt to evolve higher level protocol recommendations for existing and future data networks.

This brief summary of different network-protocol layerings is in no way comprehensive, but illustrates the diversity of protocol designs which can be found on nets providing different types of services to subscribers.

VI. Technical Interconnection Choices

A. The Issues

Beginning with the earliest papers dealing with strategies for packet-network interconnection [23]-[26], [32], the common objective of all the proposed methods is to provide the physical means to access the services of a host on one network to all subscribers (including hosts) of all the interconnected networks. Of course, limitations to this accessibility are envisaged, imposed either for administrative reasons or by the scarcity of resources. The achievement of this objective invariably requires that data produced at a source in one net be delivered and correctly interpreted at the destination(s) in another network. In an abstract sense, this boils down to providing interprocess communication across network boundaries. Even if a person is the ultimate source of the data, packet-switching networks must interpose some degree of software processing between the person and the destination service, even if only to assemble or disassemble packets produced by a computer terminal.

A fundamental aspect of interprocess communication is that no communication can take place without some agreed conventions. The communicating processes must share some physical transmission medium (wire, shared memory, radio spectrum, etc.), and they must use common conventions or agreed upon translation methods in order to successfully exchange and interpret the data they wish to communicate. One of the key elements in any network interconnection strategy is therefore how the required commonality is to be obtained. In some cases, it is enough to translate one protocol into another. In others, protocols can be held in common among the communicating parties.

In any real network interconnection, of course, a number of secondary objectives will affect the choice of interconnection strategy. For example achievable bandwidth, reliability, robustness (i.e., resistance to failures), security, flexibility, accountability, access control, resource allocation options, and the like can separately and jointly influence the choice of interconnection strategy. Combinations of strategies employing protocol standards and protocol translations at various levels of the layered protocol hierarchy are also likely possibilities.

There are a number of issues which must be resolved before a coherent network interconnection strategy can be defined. A list of some of these issues, which will be treated in more detail in succeeding sections, is:

1) level of interconnection;
2) naming, addressing, and routing;
3) flow and congestion control;
4) accounting;
5) access control;
6) internet services.

B. Gateways and Levels of Network Interconnection

The concept of a gateway is common to all network interconnection strategies. The fundamental role of the gateway is to terminate the internal protocols of each network to which it is attached while, at the same time, providing a common

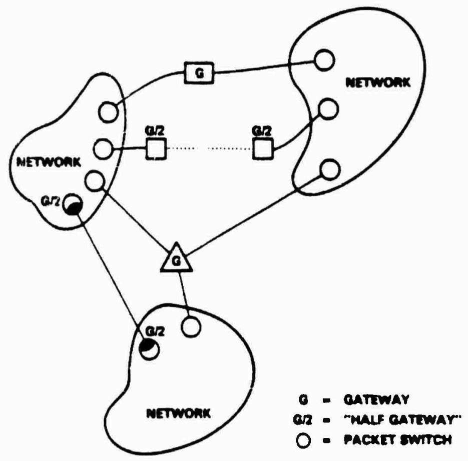


Fig. 6. Various gateway configurations.

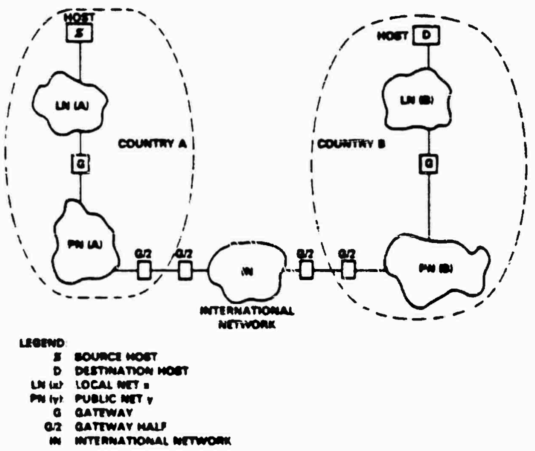


Fig. 7. International packet-networking model.

ground across which data from one network can pass into another. However, the choice of functions to be performed in the gateway varies considerably among different interconnection strategies (see Fig. 6). The term "gateway" need not imply a monolithic device which joins a pair of networks. Indeed, the gateway may merely be software in a pair of packet switches in different networks, or it may be made up of two parts, one in each network (a sort of "gateway half"). In the latter case, the two halves might be devices separate and distinct from the network packet switches or might be integrated with them. Furthermore, a gateway might interconnect more than two networks. In the material which follows, every attempt has been made to avoid any implicit choice of gateway implementation. It is worth pointing out, however, that the "half gateway" concept is highly attractive from both a technical and a purely administrative point of view. Technically, each half could terminate certain levels of protocol of the net to which it is attached. Administratively each half could be the responsibility of the network to which it belongs. Then the only matters for jurisdictional negotiation are the physical medium by which the half-gateways exchange data, and the format and protocol of the exchange.

It is important to realize that typical applications may involve three or more networks. Where local networks are used, they will usually need to be interconnected to realize the benefits of interorganizational data exchange. In most countries, such interconnections will only be permitted through a public network. Thus for a typical national situation, three networks and two gateways will be involved in providing the desired host-to-host communication.

The international picture is similar, except that more networks are likely to be involved. Shown in Fig. 7, the path from a host, S, on local network $LN(A)$ in country A, passes through a public network, $PN(A)$ in country A, through an international network IN, through a public network $PN(B)$ in country B, and finally through a local network, $LN(B)$, to the destination host, D. There are four internetwork gateways involved. It is this model involving multiple gateways that guides us away from network interconnection methods which rely on the source and destination hosts being in adjacent networks connected by the mediation of a single gateway.

1) Common Subnet Technology (Packet Level Interconnection): The level at which networks are interconnected can be determined by the protocol layers terminated by the gateway. For example, if a pair of identical networks were to be interconnected at the interpacket-switch level of protocol, we might illustrate the gateway placement as shown in Fig. 8. Here the "gateway" may consist only of software routines in the adjacent packet switches, e.g., $P(A)$ and $P(B)$, which provide accounting, and possibly readdressing functions. The contour model of protocol layer is useful here since it shows which levels are common to the two networks and which levels could be different. In essence, those layers which are terminated by the gateways could be different in each net, while those which are passed transparently through the gateway are assumed to be common in both networks. This network interconnection strategy requires that the internal address structure of all the interconnected networks be common. If, for example, addresses were composed of a network identifier, concatenated with a packet-switch identifier and a host identifier, then addressing of objects in each of the networks would be straightforward and routing could be performed on a regional basis with the network identifiers acting as the regional identifiers, if desired. Alternatively, two identical networks could adopt a common network name and assign nonduplicative addresses to each of the packet switches in both networks. This may require that addresses in one network be changed.

The strategy described above might be called the "common subnetwork strategy," since, in the end, subscribers of the newly formed joint network would essentially see a single network. This strategy does not rule out the provision of special access control mechanisms in the gateway nodes which could filter traffic flowing from one network into the other. Similarly, the gateway nodes could perform special internetwork traffic accounting which might not normally be performed in a subnet switching node. This network interconnection method is limited to those cases in which the nets to be connected are virtually identical, since the gateways must participate directly in all the subnet protocols. The end-to-end subnet protocols (e.g. source/destination packet-switch protocols) must pass transparently through the gateways to permit interactions between a source packet switch in one net and a destination packet switch in another. The resulting network presents the same network access interface to all

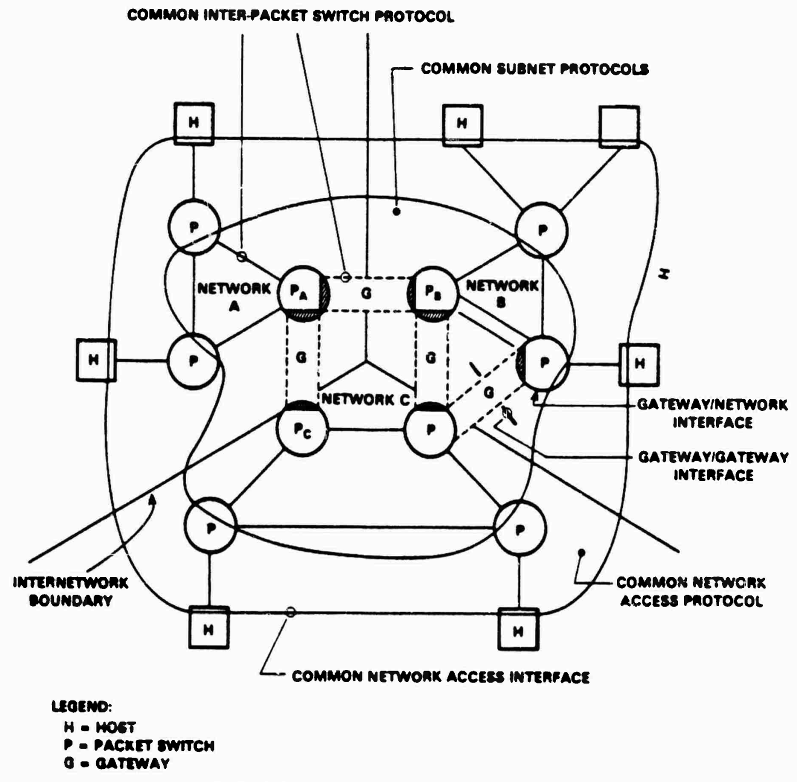

COMMON INTER-PACKET SWITCH PROTOCOL
COMMON SUBNET PROTOCOLS
H
P
NETWORK A
NETWORK B
NETWORK C
P_A
P_B
P_C
G
GATEWAY/NETWORK INTERFACE
GATEWAY/GATEWAY INTERFACE
INTERNETWORK BOUNDARY
COMMON NETWORK ACCESS PROTOCOL
COMMON NETWORK ACCESS INTERFACE
LEGEND:
H = HOST
P = PACKET SWITCH
G = GATEWAY

Fig. 8. Interconnection of common subnetworks.

subscribers, and this leads us to the next example which is based on the concept of a common network access interface.

2) Common Network Access Interfaces: If the subnetwork protocols are not identical, the next opportunity to establish internetwork commonality is at the network access interface. This is illustrated in Fig. 9. Each network is assumed to have its own intranet protocols. However, each network presents the same external interface to subscribers. This is illustrated by showing a common interface passing through all hosts, marked "common network access interface" in the figure.

Once again, the gateway could be thought of as software in adjacent packet switches. Each gateway is composed of two halves formed by linking the packet switches of two nets together. However, in this case, the subnetwork protocols are terminated at the gateway so that the intergateway exchange looks more like network access interaction than a node-to-node exchange. This is the approach taken by CCITT with its X.25 packet network interface recommendation and X.75 intergateway exchange recommendation.

It is important to note that the intergateway interface could be similar to the standard network access interface, but it need not necessarily be identical.

There are two basic types of network interface currently in use: 1) the datagram interface [31]; and 2) the virtual circuit interface [32]. The details of these generic interface types vary in different networks; some networks even offer both types of interface. In some, the interface to use may be chosen at subscription time; in others it may be possible for a subscriber to select the access method dynamically.

A datagram interface allows the subscriber to enter packets into the network independent of any other packets which have been or will be entered. Each packet is handled separately by the network. A virtual circuit interface requires an exchange of control information between the subscriber and the network for the purpose, for example, of setting up address translation tables, setting up routes or preallocating resources, before any data packets are carried to the destination. Some networks may implement a *fast select* virtual circuit interface in which a circuit setup request is sent together with the first (and possibly last) data packet. Other control exchanges would be used to close the resulting virtual circuits set up in this fashion.

It is essential to distinguish datagram and virtual circuit services from datagram and virtual circuit interfaces. A datagram service is one in which each packet is accepted and treated by the network independently of all others. Sequenced delivery is not guaranteed. Indeed, it may not be guaranteed that all datagrams will be delivered. Packets may be routed independently over alternate network paths. Duplicate copies of datagrams might be delivered.

Virtual circuit service tries to guarantee the sequenced delivery of the packets associated with the same virtual circuit. It typically provides to the host advice from the network on flow control per virtual circuit as opposed to the packet-by-packet acceptance or rejection typical of a datagram service. If the network operation might produce duplicate packets, these are filtered by the destination packet switch before delivery to the subscriber. Duplicate packet creation is a

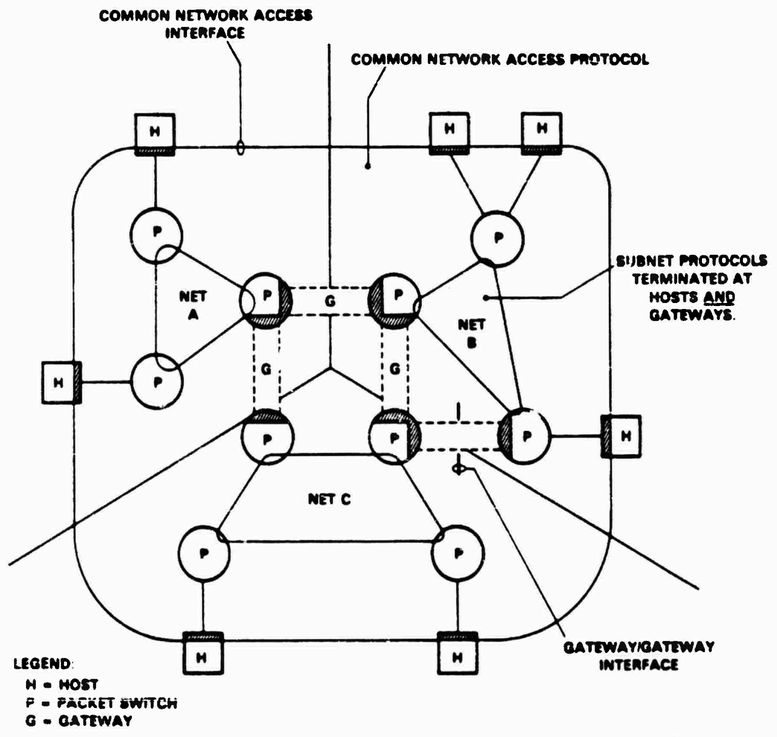

COMMON NETWORK ACCESS INTERFACE
COMMON NETWORK ACCESS PROTOCOL
SUBNET PROTOCOLS TERMINATED AT HOSTS AND GATEWAYS.
NET A
NET B
NET C
G
H
P
GATEWAY/GATEWAY INTERFACE
LEGEND:
H = HOST
P = PACKET SWITCH
G = GATEWAY

Fig. 9. Interconnection of networks with common network-access interfaces.

common phenomenon as in packet-switched store-and-forward systems. The basic mode of operation is to forward a packet to the next switch and await an acknowledgment. After a timeout, the packet is retransmitted. If an acknowledgment is lost due to line noise, for example, then two copies of the packet would have been transmitted. Even if the next switch is prepared to filter duplicates out, a network which uses adaptive routing can deliver a duplicate packet to the periphery of the network. For example, if a packet switch receives a packet successfully but the line to the sender breaks before the receiver can acknowledge, the sender may send another copy to a *different* packet switch. Both packet copies may be routed and delivered to the destination packet switch where final duplicate filtering would be needed if virtual circuit service is being provided.

Some networks offer both a datagram and a virtual circuit service; some offer a single interface, but different services. For example, the ARPANET has a basic datagram interface. However, the subnetwork will automatically provide a sequenced virtual circuit service (i.e., packets are kept in sequence when they are delivered to the destination) if the packet is marked appropriately. Otherwise, packets are not delivered in sequence nor are packet duplicates or losses, except for line by link correction, recovered within the network for nonsequenced types of traffic.

By contrast, TRANSPAC offers a virtual circuit interface and service. Subscribers transmit "call request" packets containing the full destination address to the packet switch. The request packet is forwarded to the destination, leaving behind a fixed route. The destination subscriber returns a "call accepted" packet which is delivered to the caller. As a result of this exchange, the source subscriber has associated a "logical channel number" or LCN, with the full source-destination addresses. Thus subsequent packets to be sent on the same logical channel are identified by the LCN and are kept in sequence when delivered to the destination.

Finally, it is possible to implement a datagram-like service using a virtual circuit interface. In this case, the exchange of *request* and *accept* packets might be terminated at the subscriber's local packet switch, so that even if packets were not delivered in sequence they might employ abbreviated addressing for local subscriber and packet-switch interaction.

If network interaction is to be based on a standard interface, then agreement must be reached both on the interface and an associated service or services. Furthermore, a common addressing system is needed so that a subscriber on one network can address a packet to a subscriber on any other network. A weaker assumption could be made but we are deliberately assuming a truly common service, interface, and addressing mechanism. We will return to this topic in a later section.

The choice of a standard network service through which to effect network interconnection has a primary impact on the flexibility of implementable network interconnection methods. We will consider two choices: datagram service and virtual circuit service.

a) Datagram service as a standard for network interconnection: For this case, it is assumed that every network offers a common datagram service. A uniform address space makes it possible for subscribers on any network to send packets addressed to any other subscriber on a connected network. Packets are routed between subscriber and gateway and between gateways based on the destination address. No attempt is made to keep the datagrams in any order in transit or upon delivery to the destination. Individual datagrams may be freely routed through different gateways to recover from failures or to allow load-splitting among parallel gateways joining a pair of networks.

The gateway/gateway interface may be different than the network access interface, if need be (see Fig. 9).

This strategy requires that all networks implement a common interface for subscribers. The simplicity and flexibility of the datagram interface strategy is offset somewhat by the need for all networks to implement the same interface. This is true for the pure virtual circuit interface strategy as well, as will be shown below.

One of the problems which has to be faced with any network interconnection strategy is congestion control at the gateways. If a gateway finds that it is unable to forward a datagram into the next network, it must have a way of rejecting it and quenching the flow of traffic entering the gateway en route into the next network. The quenching would typically take the form of an error or flow control signal passing from one gateway half to another on behalf of the associated network. Similar signals could be passed between subscribers and the packet network for similar reasons. Since datagram service does not undertake to guarantee end/end reliability, it is possible to relieve momentary congestion by discarding datagrams, as a last resort.

b) Virtual circuits for network interconnection: Another alternative standard network service which could be used for network interconnection is virtual circuit service (Fig. 10). Independent of the precise interface used to "set up" the virtual circuit, a number of implementation issues immediately arise if such a service is used as a basis for network interconnection.

Since it is intended that all packets on a virtual circuit be delivered to the destination subscriber in the same sequence as they were entered by the source subscriber, it is necessary that either: 1) all packets belonging to the same virtual circuit take the same path from source subscriber, through one or more gateways, to destination subscriber; or 2) all packets contain sequence numbers which are preserved end-to-end between the source DCE in the originating network and the destination DCE in the terminating network.

In the first case, virtual circuits are set up and anchored to specific gateways so that the sequencing of the virtual circuit service of each network can be used to preserve the packet sequence on delivery. This results in the concatenation of a series of virtual circuits through each gateway and, therefore, the knowledge of each virtual circuit at each gateway (since the next gateway to route the packet through must be fixed for each virtual circuit).

In the second case, there is no need to restrict the choice of gateway routing for each virtual circuit since the destination DCE will have sufficient information to resequence incoming packets prior to delivery to the destination subscriber.

In either case, the destination DCE will have to buffer and resequence packets arriving out of order due either to disordering within the last network or to alternate routing among networks, if this is permitted. Some networks may keep packets in sequence as they transit the network. This will only be advantageous at the destination DCE if the packets enter the network in the desired sequence. If such a service is relied upon in the internet environment, then each gateway must assure that on entry to such a net, the packets are in the desired order for delivery to a destination subscriber or another gateway.

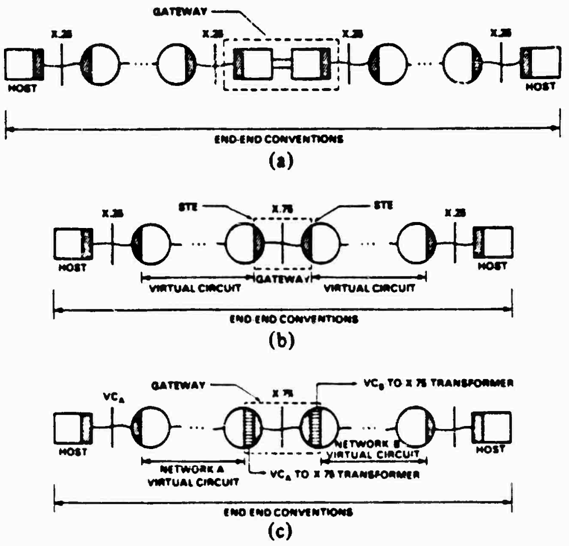


Fig. 10. Virtual circuit network interconnection strategies. (a) Subscriber-based gateway. Internet source and destination carried in user data field of X.25 call set-up packets. (b) X.75 based gateway. Note how much of the X.25 VC service is terminated at the STE. (c) X.75-based gateways with general virtual circuit networks.

The buffering and resequencing of packets within the networks or at gateways introduces substantial variation in buffer space requirements, packet transit delays, and the potential for buffer lockups to occur [50], [51], [61].

If packets for a specific virtual circuit are restricted to pass through a fixed series of gateways, and if a standard flow-control method is agreed upon as part of the virtual circuit service, then it is possible for each internet gateway to participate in end-to-end flow control by modifying the flow control information carried in packets carried end-to-end from the source DCE to the destination DCE. Consequently, a gateway may be able to adjust the amount of traffic passing through it and thereby achieve a kind of internet gateway congestion control. If this is done by allocating buffer space for "outstanding" packets, then either the gateways must guarantee the advertised buffer space or there must be a retransmission capability built into the internet virtual circuit implementation, perhaps between source DCE and destination DCE or between DCE's and gateways.

Such a mechanism does not, however, solve the problem of network congestion unless the gateway-flow control decisions take into account resources both in the gateway and in the rest of the network. Although it is tempting to assume that virtual circuit-flow control can achieve internetwork congestion control, this is by no means clear, and is still the subject of considerable research.

As a general rule, compared to the datagram method, the virtual circuit approach requires more state information in each gateway, since knowledge of each virtual circuit must be maintained along with flow control and routing information. The usual virtual circuit interface is somewhat more complex for subscribers to implement as well, because of the amount of state information which must be shared by the subscriber and the local DCE. For example, implementations of the X.25

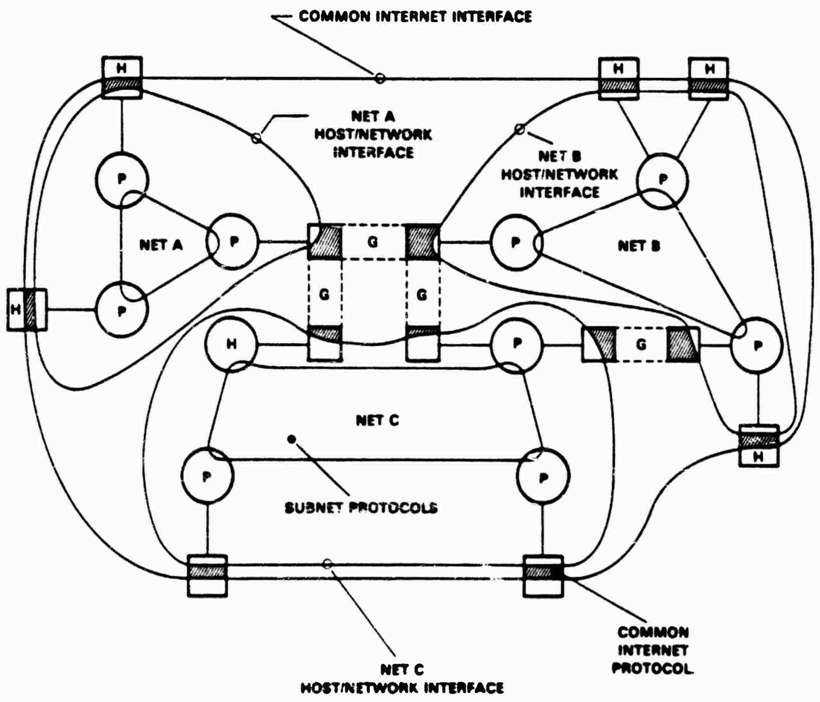

COMMON INTERNET INTERFACE
NET A HOST/NETWORK INTERFACE
NET B HOST/NETWORK INTERFACE
NET A
NET B
NET C
SUBNET PROTOCOLS
H
P
G
COMMON INTERNET PROTOCOL
NET C HOST/NETWORK INTERFACE

Fig. 11. Common internet interface.

interface protocol have been privately reported by Computer Corporation of America and University College London to require 4000–8000 words of memory on Digital Equipment Corporation PDP-11 computers. By contrast, the ARPANET and Packet Radio Network datagram interfaces require 500–1000 words of memory on the same machine. For internetwork operation, this may be even more burdensome, since any failure at a gateway may require a subscriber-level recovery through an end-to-end protocol, in addition to the virtual circuit interface software, as is shown in [52].

Nevertheless, it may be advantageous to consider internetworking standards which usefully employ both datagram and virtual circuit interfaces and services. For example, some special internet services such as multidestination delivery may be more efficient if they are first set up by control exchanges between the subscriber and the local network and perhaps gateways as well. Once set up, however, a datagram mode of operation may be far more efficient than maintaining virtual circuits for all destinations. Implicit virtual circuits which are activated by simple datagram-like interfaces are also attractive for very simple kinds of terminal equipment.

If it is not possible for all networks to implement a common network-access interface, then the next opportunity is to standardize only the objects which pass from one net to the next and to minimize any requirements for the sequencing of these objects as they move from net to net.

3) General Host Gateways: In this model, a gateway is indistinguishable from any other network host and will implement whatever host/network interface is required by the networks to which it is attached. For many networks, this may be X.25, but the strategy does not rely on this. The principle assumption is that packet networks are at least capable of carrying subscriber packets up to some maximum length, which may vary from network to network. It is specifically not assumed that these packets will be delivered in order through intermediate networks and gateways to the destination host. This minimal type of service is often termed "datagram" service to distinguish it from sequenced virtual circuit service. A detailed discussion of the tradeoff between datagram and virtual circuit types of networks is given elsewhere [52].

The basic model of network interconnection for the datagram host gateway is that internetwork datagrams will be carried to and from hosts and gateways and between gateways by encapsulation of the datagrams in local network packets. Pouzin describes this process generically as "wrapping" [37]. The basic internetwork service is therefore a datagram service rather than a virtual circuit service. The concept is illustrated in Fig. 11.

Datagram service does not offer the subscriber as many facilities as virtual circuit service. For example, not all datagrams are guaranteed to be delivered, nor do those that are delivered have to be delivered in the sequence they were sent. Virtual circuits, on the other hand, do attempt to deliver all packets entered by the source in sequence to the destination. These relaxations allow dynamic routing of datagrams among multiple, internetwork gateways without the need for subscriber intervention or alert.

The internet datagram concept gives subscribers access to a basic internet datagram service while allowing them to build more elaborate end-to-end protocols on top of it. Fig. 12 illustrates a possible protocol hierarchy which could be based on the internet datagram concept. The basic internet datagram service could be used to support transaction protocols or real-time protocols (RTP) such as packet-voice protocols (PVP) which do not require guaranteed or sequenced data

UTILITY	FTP	VTP	RTP	VP
END/END SUBSCRIBER	END/END VIRTUAL CIRCUIT		END/END DATAGRAM	
INTERNET ACCESS	INTERNET DATAGRAM			
NETWORK ACCESS	NETWORK SPECIFIC			
INTRANET, END-END	NETWORK SPECIFIC			
INTRANET, NODE-NODE	NETWORK SPECIFIC			
LINK CONTROL	NETWORK SPECIFIC			

Fig. 12. Protocol layering with internetwork datagrams.

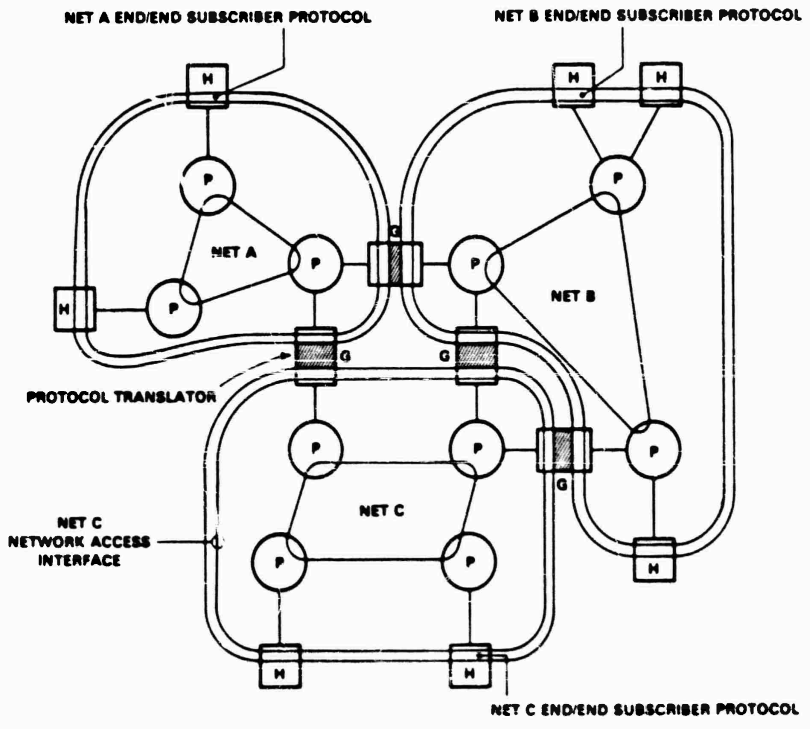

NET A END/END SUBSCRIBER PROTOCOL
NET B END/END SUBSCRIBER PROTOCOL
H
P
NET A
NET B
G
PROTOCOL TRANSLATOR
NET C
NET C NETWORK ACCESS INTERFACE
NET C END/END SUBSCRIBER PROTOCOL

Fig. 13. Host protocol translation gateway.

delivery; reliable, sequenced protocols could be constructed above the basic internet datagram service to perform end/end sequencing and error handling. Applications such as virtual terminal protocols (VTP) [40], [42], [48] or file-transfer protocols [40], [42], [49] could be built above a reliable, point-to-point, end/end service which is itself built atop internet datagrams. Under this strategy, the basic gateway functions are the encapsulation and decapsulation of datagrams, mapping of internet source/destination addresses into local network addresses and datagram routing. Gateways need not have any knowledge of higher level protocols if it is assumed that protocols above the internet datagram layer are held in common by the communicating hosts. Datagrams can be routed freely among gateways and can be delivered out of sequence to the destination host.

The basic advantage of this strategy is that almost any sort of network can participate, whether its internal operation is datagram or virtual circuit oriented. Furthermore, the strategy offers an easy way for new networks to be made "backwards compatible," with older ones while allowing the new ones to employ new internal operations which are innovative or more efficient.

Every subscriber must implement the internet datagram concept for this strategy to work, of course. The same problem arises with the standard network interface strategy since all subscribers must implement the same network interface.

4) Protocol Translation Gateways: It would be misleading to claim that the concept of protocol translation has not played a role in the discussion thus far. In a sense, the encapsulation of internet datagrams in the packet format of each intermediate network is a form of protocol translation. The basic packet carrying service of one network is being translated into the next network's packet carrying service (see Fig. 13). This concept could be extended further. For example, if two networks have a virtual circuit concept, one implemented within the subnetwork and the other through common host/host protocols, it might be possible, at the gateway between the nets, to map one network's virtual circuit into the other's. This same idea could be applied to higher level protocol mappings as well; for instance, the virtual terminal protocol for one network might be transformed into that of another "on the fly."

The success of such a translation strategy depends in large part on the commonality of concept between the protocols to be translated. Mismatches in concept may require that the service obtained in the concatenated case be a subset of the services obtainable from either of the two services being translated. Extending such translations through several gateways can be difficult, particularly if the protocols being translated do not share a common address space for internetwork sources/destinations. In the extreme, this strategy can result in subscribers "logging in" to the gateway in order to activate the protocols of the next network. Indeed, front-end computers could be considered degenerate translation gateways since they transform host/front-end protocols into network protocols.

There are circumstances when translation cannot be avoided. For instance, when the protocols of one network cannot be modified, but internet service is desired, there may be no alternative but to implement protocol translations. The model typically used to guide protocol translation gateways is that the source/destination hosts lie on either side of the translation gateway. Concatenation of protocol translations through several networks and gateways is conceivable, but may be very difficult in practice and may produce very inefficient service.

C. Names, Addresses, and Routes

In order to manage, control, and support communication among computers on one or more networks, it is essential that conventions be established for identifying the communicators. For purposes of this discussion, we will use the term *host* to refer to all computers which attach to a network at the network-access level of protocol (see Table I). Subscribers to terminal-access services can be thought of as attaching to hosts, even if the host is embedded in the hardware and software of a packet switch as a layer of protocol. Consequently, we can say that the basic task of a packet-switching network is to transport data from a source host to one or more destination hosts.

To accomplish this task, each network needs to know to which destination packets are to be delivered. Even in broadcast nets such as the ETHERNET, this information is necessary so that the destination host can discriminate packets destined for itself from all others heard on the net. At the lowest-protocol levels it is typical to associate destinations with *addresses*. An address may be simply an integer or it may have more internal structure.

At higher levels of protocol, however, it is more common to find text strings such as "MULTICS" or "BBN-TENEX" used as *names* of destinations. Application software, such as electronic mail services, might employ such names along with more refined destination identifiers. For example, one of the authors has an electronic mailbox named "KIRSTEIN at ISI" located in a computer at the University of Southern California's Information Sciences Institute.

Typically, application programs transform names into addresses which can be understood by the packet-switching network. The networks must transform these addresses into *routes* to guide the packets to their destination. Some networks bind addresses to routes in a relatively rigid way (e.g., setting up virtual circuits with fixed routing) while others determine routes as the packets move from switch to switch, choosing alternate routes to bypass failed or congested areas of the network. Broadcast networks need not create routes at all (e.g., SATNET).

In simple terms, a *name* tells what an object is; an *address* tells where it is; and a *route* tells how to get there [54]. A simple model involving these three concepts is that hosts transform names into addresses and networks transform addresses into routes (if necessary). However, this basic model does leave a large number of loose ends. The subject is so filled with issues that it is not possible in this paper to explore them all in depth. In what follows, some of the major issues are raised and some partial resolutions are offered.

One major question is "Which objects in the network should have names? addresses?" Pouzin and Zimmermann offer a number of views on this question in their paper in this issue [37]. A generic answer might be that at least all objects which can be addressed by the network should have names as well so that high-level protocols can refer to them. For example, it might be reasonable for every host connection on the network to have an name and an address. There also may be objects internal to the network which also have addresses such as the statistics-gathering *fake hosts* in the ARPANET [38].

A related issue is whether objects should or can have multiple names, multiple addresses, and multiple routes by which they can be reached. The most general resolution of this issue is to permit multiple names, addresses, and routes to exist for the same object. An example taken from the multinetwork environment may serve to illustrate this notion. Fig. 6 shows three networks which are interconnected by a number of gateways. Each gateway (or pair of gateway halves) has two interfaces, one to each network to which it is attached. Plainly there is the possibility that several alternate routes passing through different gateways and networks could be used to carry packets from a source host in one net to a destination host in another net. This is just the analog of alternate routing within a single network.

Furthermore, each gateway has two addresses, typically one for each attached network. This is just the analog of a host on one network attached to two or more packet switches for reliability. The term *multihoming* is often used to refer to multiply attached hosts.

Finally, it may be useful to permit a gateway to have more than one name, for example, one for each network to which it is attached. This might allow high-level protocols to force packets to be routed in certain ways for diagnostic or other reasons. Multiple naming also allows the use of nicknames for user convenience. Many of these same comments would apply to hosts attached to multiple networks.

An interesting addressing and routing problem arises in mobile packet radio networks. Since hosts are free to move about, the network will need to dynamically change the routes used to reach each host. For robustness, it is also desirable that hosts be able to attach dynamically to different packet radios. Thus failure of a packet radio need not prevent hosts from accessing the network. This requires that host names and perhaps host addresses be decoupled from packet radio addresses. The network must be able to search for hosts or alternatively, hosts must "report-in" to the network so that their addresses can be associated with the attached packet radio to facilitate route selection based on host address. This is just a way of supporting *logical host addressing* rather than using the more common *physical host addressing* in which a host's address is an extension of the packet-switch address.

A crucial issue in network interconnection is the extent to which it should or must impact addressing procedures which are idiosyncratic to a particular network. It is advantageous not to require the subscribers on each network to have detailed knowledge of the network address *structure* of all interconnected networks. One possibility is to standardize an internetwork address structure which can be mapped into local network addresses as needed, either by subscribers, by gateways or by both. Subscribers would know how to map internetwork service names into addresses of the form NETWORK/SERVER. Subscribers need not know the fine structure of the SERVER field. Gateways would route packets on the basis of the NETWORK part of the address until reaching a gateway attached to the network identified by NETWORK. At this point, the gateway might interpret the SERVER part of the address, as necessary, to cause the packet to be delivered to the desired host.

The addressing strategy presently under consideration by CCITT (X.121, [30]) is based on the telephone network. Up to 14 digits can be used in an address. The first 4 digits are a "destination network identification code" or DNIC. Some countries are allocated more than one DNIC (the United States has 200). The remaining ten digits may be used to implement a hierarchical addressing structure, much like the one used in the existing telephone network.

Since the CCITT agreements are for international operation, it might be fair to assume that the United States will not need more than 200 public network identifiers. However, this scheme does not take into account the need for addressing private networks. The private networks, under this addressing procedure will most likely appear to be a collection of one or more terminals or host computers on one or more public networks. It is too early to tell how much this asymmetry in addressing between public and private networks will affect private multinetwork protocols.

A related problem which is not unique to network interconnection has to do with addressing (really multiplexing and demultiplexing) at higher protocol levels. The public carriers tend to offer services for terminal as well as host access to network facilities. This typically means that addresses must be assigned to terminals. The issue is whether the terminal address should be associated with or independent of the protocols used to support terminal-to-host communication.

The present numbering scheme would not distinguish between a host address and a terminal address. A host might have many addresses, each corresponding to a process waiting to service calling terminals.

There has been discussion within CCITT concerning "subaddressing" through the use of a user data field carried in virtual call "setup" packets. This notion would support the concept of a single host address with terminal or process level demultiplexing achieved through the use of the user data field subaddressing.

It seems reasonable to predict that, as terminals increase in complexity and capability, it will eventually be attractive to support multiple concurrent associations between the terminal and several remote service facilities. Applications requiring this capability will need terminal multiplexing conventions beyond those currently provided for in the CCITT recommendations.

To simplify implementations of internet protocol software, it is essential to place bounds on the maximum size of the NETWORK/SERVER address. Otherwise, subscribers may have to construct name-to-address mapping tables with arbitrarily large and complex entries.

Even if all these issues are resolved, there is still a question of "source routing" in which a subscriber defines the route to be taken by a particular packet or virtual circuit. Depending on the range of internetwork services available, a subscriber may want to control packet routes. It is not yet clear how such a capability will interact with access control conventions, but this may be a desirable capability if gateways are not able to automatically select routes which match user service requirements.

D. Flow and Congestion Control

For purposes of discussion, we distinguish between flow and congestion control. Flow control is a procedure through which a pair of communicators regulate traffic flowing from source to destination (each direction possibly being dealt with separately). Congestion control is a procedure whereby distributed network resources, such as channel bandwidth, buffer capacity, CPU capacity, and the like are protected from oversubscription by all sources of network traffic. In general, the successful operation of flow-control procedures for every pair of network communicants does not guarantee that the network resources will remain uncongested.

In a single network, the control of flow and congestion is a complex and not well understood problem. In a multinetwork environment it is even more complex, owing to the possible variations in flow and congestion control policies found in each constituent network. For example, some networks may rigidly control the input of packets into the network and explicitly rule out dropping packets as a means of congestion control. At the other extreme, some networks may drop packets as the sole means of congestion control.

At this stage of development, very little is known about the behavior of congestion in multiply interconnected networks. It is clear that some mechanisms will be required which permit gateways and networks to assert control over traffic influx especially when a gateway connects networks of widely varying capacity. This problem is likely to be most visible at gateways joining high speed local networks to long-haul public nets. The peak rates of the local nets might exceed that of the long-haul nets by factors of 30-100 or more. Generic procedures are needed for gateway/network and gateway/gateway flow and congestion control. Such problems also show up in single networks, but are amplified in the multinetwork case.

E. Accounting

Accounting for internetwork traffic is an important problem. The public networks need mechanisms for revenue sharing and subscribers need simple procedures for verifying the accuracy of network-provided accountings.

The public packet-switching networks appear to be converging on procedures which account for subscriber use on the basis of the number of virtual circuits created during the accounting period and the number of packets sent on each virtual circuit. Indeed, it has been argued that accounting on the basis of virtual circuits at gateways requires less overhead than accounting on a pure datagram basis [32]. Scenarios can be cited which support the opposite conclusion.

Suppose there is a choice between setting up virtual circuits for each transaction and sending a datagram for each transaction, and that virtual circuit accounting includes information on each virtual circuit setup (as in the present telephone network). If datagram accounting simply accumulates the number of datagrams sent between particular sources and destinations without regard to the time at which they are sent, then the amount of accounting information which is collected for the datagram case will be substantially less than for the virtual circuit case. In the limit (i.e., one packet per transaction), the virtual circuit accounting information is proportional to $2N$, where N is the number of transactions, while for the datagram case, it is proportional to log N (base 2). This is simply because the datagram case only sums counts for traffic between source/destination pairs while the virtual circuit accounting would identify start/stop times for each virtual circuit.

Alternatively, if the bulk of the traffic involves a large number of packets per transaction, then the two accounting procedures would accumulate more nearly the same information since each would predominantly involve accounting for packet flow.

If it is chosen not to account for virtual circuit duration, but merely to account independently for the number of virtual circuits and the number of packets sent between source/destination pairs, then the virtual circuit accounting would be closer to the datagram case.

The important conclusion to be drawn is that accounting for datagrams is generally less complex than accounting for virtual circuits, but that the two can be made arbitrarily similar by suitable choice of the details of the accounting information collected.

F. Access Control

In multinetwork environments, it may be necessary for each network to establish and enforce a policy for "out-of-network" routing. For example, a public network might conclude agreements with other networks regarding the type and quantity of traffic it will forward into other networks. This might even be a function of the time-of-day. Consequently, mechanisms are needed which will permit networks to prevent traffic from entering or leaving or to meter the type and rate of traffic passing into or out of the network.

Another example of the need for control arises with the possibility of third-party routing. That is, traffic destined from network A to network B is routed through network C. It cannot be assumed that all networks have gateways to all others. However, some nets may want to limit the amount of *transit* traffic they carry. There may be explicit agreements among a subset of the nets regarding revenue sharing for transit services. If a particular network does not have a revenue-sharing agreement with the particular source/destination networks of a given virtual circuit or datagram, then it must be able to reject the offending traffic if it so chooses.

There does not seem to be any technical barrier to separating the access control policy decision mechanism from the enforcement of the policy. For example, a gateway might simply enforce policy by sending traffic for which it has no known access rules to an *access controller*. If we adhere to the model that gateways have two *halves*, then each half deals with the network to which it is connected. The access controller can either dynamically enable the flow by causing table entries at the gateways which permit the flow to be created or it can tell the gateway to reject all further traffic of that type.

Clearly, access control policies will affect routing strategies, so this adds a complicating factor into any internetwork routing strategy implemented by the gateways. At present, very little experience has been accumulated with internet access control and routing policies. For the most part, agreements among public networks have been bilateral and transit routing has been treated as a very special case. When EURONET [6] becomes operational, this problem will be particularly important to solve.

G. Internet Services

It is by no means clear what set of services should be standardized and available from, at least, all public data networks. The current CCITT recommendations provide for virtual circuit service and terminal access service on all public packet-switching networks.

Although the recommendations (X.3, X.25) provide for *fragmentation* of packets being delivered to a subscriber on a virtual circuit, the current X.75 gateway draft recommendation uses an agreed maximum packet size of 128 octets of data, not including the header. This agreement avoids for the moment the need to fragment packets crossing a network boundary, as long as all subscribers recognize that the maximum length internetwork packet allowed is 128 octets. Bilateral exceptions to this rule may develop but neither a fixed size nor a collection of special cases represent a very general solution to this problem.

It has been argued [25] that a general scheme for dealing with fragmentation is desirable so that new network technologies supporting larger packet sizes can be easily integrated into the multinetwork environment.

Apart from fragmentation, there are a set of special services such as multidestination addressing and broadcasting which could be used to good advantage to support multinetwork applications such as teleconferencing, electronic mail distribution, distributed file systems, and real-time data collection. Other services such as low delay, high reliability, high bandwidth, and high priority are also candidates for standardization at the internet level.

As in the case of access control, selection of such services might constrain the choice of packet routing to networks capable of supporting the desired services. Once again, very little experience with standard internet services has been accumulated so this subject is still a topic for research. For the most part, terminal-to-host services have been successfully offered across network boundaries using nearly all of the network interconnection methods described in this paper. It remains to be seen whether more complex applications can be equally well supported.

VII. X.25/X.75–The CCITT Strategy for Network Interconnection

The common network access interface concept is favored by CCITT for network interconnection. In the CCITT model of packet networking, all networks offer the same interface to packet-mode subscribers and this is called X.25. X.25 is a virtual circuit interface protocol. However, gateways between networks employ an interface protocol called X.75 [33], which is much like X.25 but accommodates special network/network information exchange, such as routing information, accounting information, and so on.

Fig. 10(a) illustrates the basic network interconnection strategy proposed by CCITT. To appreciate the difference between this strategy and the "common subnetwork" strategy, it is necessary to have some understanding of the X.25 packet network interface. X.25 provides a virtual circuit interface for the setup, use, and termination of virtual circuits between subscribers of the networks. X.25 provides for flow control of packets per virtual circuit flowing into or out of the network. Subscribers may set up switched virtual circuits by sending "call request" packets into the network and receiving "call confirmation" packets in return. The standard also provides for permanent virtual circuits.

The public networks plan to employ X.25 interfaces; it can therefore be assumed that source and destination hosts in different networks will essentially want to exchange "call request" and "call accepted" packets through the mediation of one or more gateways. This strategy could result in a series of virtual circuits chaining source host to gateway, gateway to gateway, and gateway to destination host; alternately an end-to-end virtual circuit could be set up from source host to destination host, with the gateways acting as relays without any special knowledge of the virtual circuits passing across the network boundary.

The principle difference between the X.25 interface and X.75 interface is that virtual circuit setup and clearing packets are passed transparently by the X.75 gateway to the next gateway or destination. For reasons which are described below, it is necessary to maintain the sequence of packets belonging to a given X.25 virtual circuit as they pass through a gateway and enter the next network. Therefore, a virtual circuit is in fact created between the source host and intermediate gateway and between gateways. The X.75 gateway does not spontaneously generate any "call acceptance" packets in response to "call request" packets, but it does participate in the sequencing and flow control of packets on each virtual circuit passing through. Other differences between the X.25 and X.75 interface have to do with the nature of the internetwork accounting or routing information which might be exchanged over X.75 which would not be appropriate for a subscriber to exchange with the network over the X.25 interface.

The design of the X.75 type of gateway depends in principle upon all networks' use of the X.25 subscriber interface. Some networks, like the ETHERNET, cannot implement it without extensive modification, because there are no packet switches in the network to support the required packet reordering at the destination. The alternative is to insist that all internet applications rely on a sequenced data protocol built into the hosts or front-ends. For some services, such as packet speech, the potential overhead of resequencing packets before delivery to the destination may prevent the service from being viable. This problem could be amplified if packets are constrained to remain in sequence as they pass the X.75 boundary.

Fig. 10(b) and (c) shows variants of the CCITT interconnection strategy. In Fig. 10(b), we see an example in which only X.25 is used both as a network access method and as a means of passing traffic across network boundaries. A single subscriber or a pair of subscribers to two nets could interface to their networks via X.25 and to each other by means of some agreed and possibly private protocol.

Virtual circuits would be explicitly set up from source host to gateway, gateway to gateway, and gateway to destination host. The "internet" addresses of the source and destination hosts could be carried in the so-called "Call User Data Field" of an X.25 Call Request packet. This leaves the packet address field free to identify intermediate destinations (e.g., gateways), but preserves an ultimate internetwork source/destination address which the gateway can use to select the destination to which the next intermediate virtual circuit is to be set up.

An alternative to this is shown in Fig. 10(c) in which the subnets *A* and *B* use nonstandard virtual circuit interfaces, but agree to build gateway software employing X.75 signaling procedures across the gateway interface. This solution is substantially the same as that shown in Fig. 10(b), except there is now additional translation software in each gateway half to make each virtual circuit network-access protocol compatible with X.75 procedures.

There are some specific problems with the X.25/X.75 gateway strategy, which do not necessarily apply to other virtual call gateways [63]. The basic X.25 interface provides for the sequence numbering of subscriber packets mod. 8 or, optionally, mod. 128. Since X.25 is an interface specification, this numbering can only be relied upon to have local significance (i.e., host-to-packet switch). Some X.25 implementations use these host-assigned sequence numbers on an end-to-end basis. Others generate internal, network-supplied numbers to allow for repacking of subscriber packets into larger or smaller units for transport to the destination. If packet sequence numbers assigned by the source host were carried transparently to the destination without change, it might be possible to allow packets to flow out-of-order across the X.75 boundary to a gateway and thence into the next network. If the packet sequence numbers were still intact, they could be carried out-of-order to the next destination which might either be a gateway or an X.25 host. In the latter case, the original packet-sequence numbers could be used to resequence the packets before delivery. If the packets were being delivered to an intermediate gateway, they would not have to be sequenced there. However, the X.25 interface specification does not undertake to carry the host-supplied sequence numbers to the destination gateway or host in a transparent fashion, primarily so that the subnetwork can deal more freely with the physical packaging of the packet stream. For example, a source may supply packets of length 128 bytes while a destination may prefer to receive packets no longer than 64 bytes. To allow for such variations, the network must be free to renumber packets for delivery. These considerations have two consequences.

1) X.25 packet sequence numbers cannot be relied on for end-to-end signaling, though they could be so used if requisite information is known about the intermediate transit networks.

2) Packets must be delivered in sequence when passing to or from gateways and hosts on X.25 networks.

The second conclusion may be modified slightly. It is at least essential that packets be delivered in relative sequence on each virtual circuit. By maintaining independent sequence numbering on each virtual circuit, it is possible for hosts and gateways to refuse traffic on one virtual circuit while accepting traffic on another. There are two penalties for this. First, a gateway must keep track of which virtual circuits are passing through it. Second, dynamic alternate routing of packets belonging to the same virtual circuit through alternate gateways is not possible without resetting or clearing the virtual circuit. This last point is simply the consequence of not defining an end-to-end sequence numbering scheme, but instead relying on sequencing of the packets of a virtual circuit on entry to and exit from each intermediate network.

Some networks implement X.25 level acknowledgments (i.e., level 3) that have an end-to-end significance, but others make this purely a host-to-packet switch matter. As a conse-

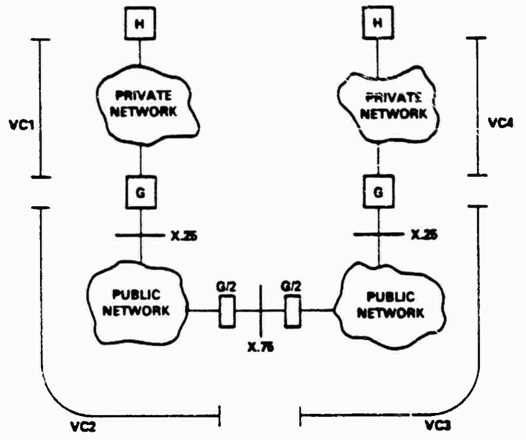


Fig. 14. Use of X.25 for public/private network interconnection.

quence, it is not possible to rely on X.25 packet acknowledgments to determine which, if any, packets were not delivered as a result of the resetting or clearing of a virtual circuit. Furthermore, even if a subnet were to offer an end-to-end acknowledgment between a source host and an X.75 gateway, this could not be assumed to guarantee that the acknowledged packet was delivered to the ultimate X.25 destination in another network.

X.75 is an interface intended for use between public networks. Thus, it is not likely to be used or even allowed as an interface between public networks and private networks. For the case illustrated in Fig. 14, X.25 interfaces could be provided between public and private networks (or other special interfaces) and X.75 interfaces between public networks. Consequently, gateways between public and private networks are likely to appear to be ordinary host computers in the view of the public networks.

The use of X.25 for private/public network interfaces and X.75 for public/public network interfaces leads to the situation shown in Fig. 14 in which an internetwork virtual circuit would have to be made up of several concatenated parts such as virtual circuits 1-2-3-4 (see also [52, Fig. 3.4]). Even if X.25 implementations uniformly permitted an end-to-end interpretation of packet sequence numbers and acknowledgments, there would still be separate virtual circuits required between the source or destination hosts and the gateways into the public networks. However, the concatenation of virtual circuits does not yield a virtual circuit. For instance, a gateway between the public and private net could acknowledge a packet but fail to get it delivered, in which case the subscriber will have been misinformed as to the delivery of the packet. This situation forces the end subscribers of private networks to implement end-to-end procedures on top of any concatenated virtual circuits provided by the public networks.

VIII. Practical Network Connections and Experiments in Progress

A number of networks have been connected successfully over the last few years. Most of these connections have been made in an *ad hoc* manner, using one of the following techniques.

1) One network is a star network with remote RJE and interactive stations. The other is a star or distributed network with clearly defined protocols. A device on the star network provides exactly the functions required by its own network on one side, and those of the other network on the other side.

2) Formal gateways are provided between the two networks, and protocol mapping occurs in the gateway.

3) A computer is a host on two networks. It is arranged that services are provided by accepting input from one network and putting it out on another, possibly after substantial processing.

4) Formal gateways are provided between the two networks. Sufficient agreement is obtained that end-to-end protocols (even high level ones) are common in the two networks. In this case, less activity is required in the gateway.

In the first method, a form of front-end computer is used. It has been adopted in the large airline and banking networks SITA [13] and SWIFT [14]. In each case the standards for the networks have been defined rigidly. SWIFT has even certified officially the devices of three manufacturers to provide interfaces to its network. The other side of the device is then programmed to meet the requirements of the star system being attached. In the two cases cited, only a simple message level of interface needed to be defined.

Other examples of the same technique are the connection of the Rutherford Laboratory (RL) star system [53] and the Livermore CTRNET to ARPANET. In these examples, more serious protocol mapping was required. ARPANET has a well-defined set of HOST-IMP, HOST-HOST, Virtual Terminal, and File Transfer protocols. All these had to be mapped into the appropriate procedures for the other network.

The second method has been applied only experimentally. The UCL interface between ARPANET and the UK Post Office Experimental Packet Switched Service (EPSS, [55]) and the National Physical Laboratory interface between EPSS and the European Informatics Network (EIN, [56]) are examples of this technique; a demonstration has even been made of EIN-EPSS-ARPANET with no extra problems encountered from the three networks being concatenated. Technically there is almost no difference between the first two methods. The second looks at first sight somewhat more general than the first, but almost the same problems have to be overcome. The difficulties come from the fundamental differences in the design choices made in the protocols of the different networks; these differences are in general difficult, and even sometimes impossible, to resolve completely. In the first method, they can sometimes be resolved using a specific facility in the star network; in the second, where two distributed networks are involved, this recourse may no longer be available.

One example of the problem occurs in the connection of EPSS and ARPANET. ARPANET can forward any number of characters at a time, and often uses full duplex remote echoing. EPSS works in a half-duplex mode, forwarding only complete records. A special "Transmit Now" has to be input by the user, and interpreted by the gateway, to ensure that partial records are forwarded. Another example, from the same application, occurs in File Transfer. ARPANET assumes an interactive process is live throughout the file transfer; all completion codes are passed over this live channel. The RL network (and EPSS) assume that file transfer is a batch process; they return network completion codes at a later time, and may delay acting on the commands. With the ARPANET-RL link [53], the file transfer job had to be given a very high priority, so that the completion code usually arrived before a timeout occurred; because of the nature of the way the computer was used for large real-time jobs, this did not always ensure that the job was run in a reasonable time.

There are several examples of the third technique. A DEC PDP 10 machine used on the Stanford University SUMEX project is a host both on ARPANET and on TYMNET; several machines at Bolt, Beranek and Newman are both on ARPANET and TELENET. Because the TENEX operating system has good facilities for linking between programs, it would be possible for interactive streams to come in one network and go out on another. File transfer problems would be simple in this configuration, because the hosts obey all the conventions, in any case, of each network. Of course, this mode of operation may require that files in transit between networks may have to be stored temporarily in their entirety in the host serving as the gateway between the networks.

The fourth technique is newer, and has many variations. As a result of agreement on the X.25, and partial agreement on the X.75, protocols, PTT networks are able to interconnect in a reasonably straightforward manner. The connections between DATAPAC and both TELENET and TYMNET have been done in this way. In each case, there has not been any agreement on higher level protocols, so the problems of host-host communication across concatenated networks is not resolved by these linkups of the subnets.

The ARPA-sponsored INTERNET project has tried to standardize to a higher level. A host-host protocol has been defined (TCP, [25]), and is being implemented on a number of different networks including Packet Radio [20], [21], ETHERNET [18], LCSNET [64] and the SATNET [22], in addition to ARPANET. This protocol is defined for use across networks; thus each packet includes an "Internet Header" which is kept invariant as the packet crosses the different networks. One aspect of the INTERNET program is to develop gateways which can interpret this header appropriately.

By late 1976, the ARPA project had connected together the Packet Radio Network, the ARPANET, and the Atlantic Packet Satellite Network using two gateways between the Packet Radio Network and the ARPANET and three gateways between the ARPANET and Packet Satellite Network. It is routinely possible to access ARPANET computing resources via either of the other nets and to artificially route traffic through multiple nets to test the impact on performance. In one such test, a user in a mobile van in the San Francisco area accessed a DEC PDP-10 TENEX system at the University of Southern California's Information Sciences Institute over the following path:

1) from van to the first gateway into ARPANET via the Packet Radio Network;
2) across the ARPANET to a second gateway in London, using a satellite link internal to the ARPANET;
3) across the Atlantic Satellite Network to a third gateway in Boston;
4) across the ARPANET again to USC-ISI.

The user and server were 400 geographical miles apart, but the communication path was 50000 miles long and passed through three gateways and four networks. Except for a slightly increased round-trip delay time, service was equivalent to a direct path through the ARPANET. Since the Packet Radio Network is potentially lossy, can duplicate packets, and can deliver packets out of order, the end/end TCP protocol was used to exercise flow and error control on an end-to-end basis. The availability of a common set of host-level protocols substantially aided the ease with which this test could be conducted.

The ARPA project also has high-level standard protocols already in existence to support file transfer and virtual terminals (the FTP and TELNET protocols [40]), and these are being retrofitted above the internet TCP protocol to provide a standard high-level internetwork protocol hierarchy.

IX. Regulatory Issues

The regulatory issues in the interconnection of packet networks takes a different form in North America than elsewhere. It is hard in a paper of this type to more than touch on some of the problems involved. The discussion here is simplistic in the extreme, and no attempt is made to put the issues in the legalistic language they really require.

In almost all countries the provision of long distance communication transmission and switching is provided by a regulated carrier. In most countries outside North America, this carrier is a single national entity—called the "PTT". In some countries (e.g., Italy) there are different carriers for different services—e.g., telegraph, telephone, intercity, international telephone, etc. In North America there are many carriers. Usually only one in each geographical area has a monopoly on public switched voice traffic. Also the so-called "Record Carriers" have some sort of monopoly on "record traffic," which is message traffic. In a "Value Added Network" (VAN), the operators rent transmission equipment from the carriers, and then add their own switching equipment. These VAN's are themselves regulated in what they may do, what traffic they may carry, and what rates they may charge. Between North America and Europe, specific "International Record Carriers" (IRC) have monopoly rights on data and message transmission —in collaboration with the appropriate European PTT's. The regulations take into account who owns the hosts and terminals, who owns the switches, who rents the transmission lines, what types of traffic is carried, what is the geographic extent of the network, and what is the technology of long distance transmission.

In Fig. 15, a single network N is sketched. It consists of switches S and transmission lines L; these together are called the data network, DN. It consists also of terminals T and hosts H; the exact difference between a terminal and a host is not very clear; we believe it is assumed that terminals mainly enter and retrieve data without processing; while a host transforms the information by processing. This definition probably does not meet the picture of modern "intelligent terminals," but it is always hard for the regulations to keep up with the technology. If the total network is all localized in one site, so that no communication lines cross public rights of way, then it can usually be considered from a regulatory viewpoint, as a single host in more complex network connections. The hosts and the terminals can be connected to the switches, and the switches to each other, either by leased lines, or by the Public Switched Telephone Network; the first type of connection is called a *leased* connection, the second *switched*. In the subsequent discussion of this section, the term "host" will include localized networks. In general we will assume the connections between the switches are via leased lines; if that is not the case, the regulations are much eased in general (though in some countries, like Brazil, no data transmission is permitted at all via switched telephone lines).

If all the hosts and switches are owned by one organization P, which also leases the lines, then P is said to own and operate the network, and it is called a "Private Network." There are

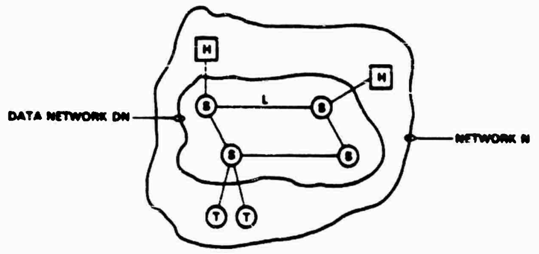


Fig. 15. Schematic of one network.

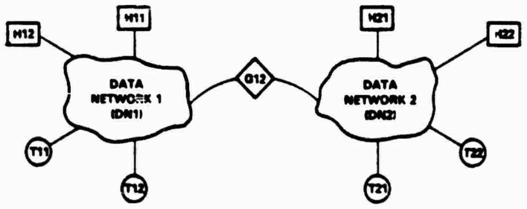


Fig. 16. Schematic of two connected networks.

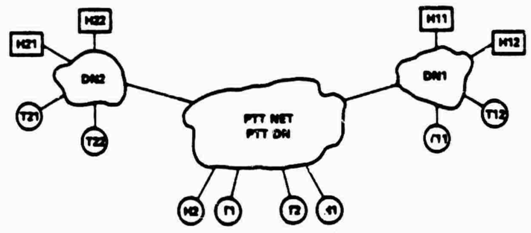


Fig. 17. Schematic of PTT model.

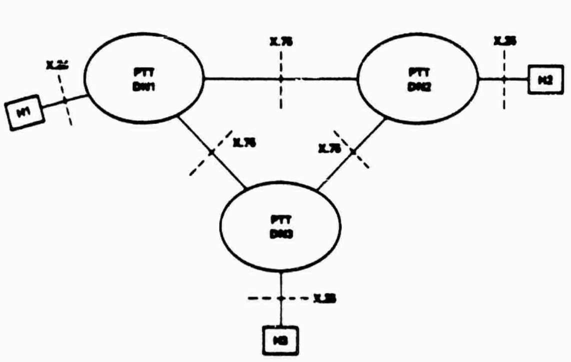


Fig. 18. Multiple PTT network interconnection.

minimal restrictions on such networks—though in West Germany, for example, higher tariffs are charged for the leased lines if any terminals or hosts are connected via the PSTN. In most countries such a network may not be used for the transfer of messages between terminals belonging to organizations other than *P*.

If the data network belongs to one organization, and the hosts to others, the data network is a VAN. Stringent regulations apply to VAN's, in most countries. With rare exceptions, in most European countries, VAN's can be operated only by the PTT's. In the U.S., they can be operated by other organizations, but only if approved as regulated Value Added Carriers (VAC's) by the Federal Communications Commission (FCC). One regulation imposed by the U.S. is that an organization operating as a VAC may not also operate a host for outside sale of services. For this reason, the companies TYMSHARE and ITT have had to spin off their VAC's into separate subsidiaries, TYMNET and ITT Data Services.

In the past, a few VAN's have been permitted to operate internationally for specific interest groups. Two such VAN's are SITA [14], for the airlines, and SWIFT [14] for the banking community. Here the regulations can be stringent. SWIFT has to pay specially high tariffs for its leased lines; its license to operate may be revoked when the PTT's can offer a comparable international service.

As soon as two networks, owned by different organizations, are interconnected, there are regulatory difficulties. This situation is illustrated schematically in Fig. 16. Even if one network is an internal one, so that it can be treated as a single host, its connection to other network immediately changes the latter's status. Thus in Fig. 16, the connection of *DN*1 to *DN*2 immediately changes *DN*2 to a VAN. In Europe it has been decreed that such private networks may not connect directly to each other, but only through a PTT network. Thus the most general configuration permitted by the European PTT's is illustrated in Fig. 17. Moreover, the PTT's have also agreed that only the X.25 interface will be provided to customers, though that interface was defined for the configuration of Fig. 15 rather than 17. The different PTT networks will themselves connect to each other by the different interface X.75 as illustrated in Fig. 18. This does not change, however, the interface seen by the private networks. Further work is needed to assess the suitability of X.25 in this role.

In the U.S., the regulations are not quite so stringent. Connections such as Fig. 15 are permitted even where one host belongs to a different organization than the network operator *P*-provided such connection is only limited and for the purposes of using the facilities of that network. This type of relaxation is really necessary, because of the difficulty of distinguishing between a "host" and a "terminal". In practice, in most countries, the line is drawn between leased line and PSTN connections. The former are usually not permitted without change of status of the network; the latter seem to downgrade the connection to that of a terminal.

The discussion above has treated the types of connections which can be made. In addition, the PTT's, and the FCC in the U.S., usually regulate the purposes for which the network can be used. In particular, there is a ban on such networks being used for message or voice transmission between organizations. How such measures are to be policed, gets us into another regulatory problem. For example the UK PO [57] has claimed a right to inspect the contents of any data message sent across lines leased from it: this right would be at variance with the privacy laws being enacted in many countries [58], [59]. This subject is a large one in its own right, and it is clearly beyond the scope of this paper.

Two other service problems will arise in international connections. First the impact and form of the privacy and transnational data flow regulations in different countries are different. Thus in the interconnection of international networks, a particular set of problems may arise, even when the appropriate regulations are obeyed in each network separately. Thus both Network 1 in country *A* and Network 2 in country *B* may obey their own national regulations. However when the networks *A* and *B* are connected, Network 1's practices may break country *B*'s regulations, and yet be accessible from country *B*. It is this class of problems which delayed seriously the permission by the Swedish Data Inspectorate Board for Swedish banks to connect their networks to SWIFT.

Secondly, some of the functions of networks or gateways legal in one country may be illegal in another. Thus U.S. carriers are not permitted to do data processing in their data networks; no such considerations apply in most European countries. Some of the protocol translation activities, some of the message processing activities, and some of the high-level services (e.g. the provision of multiaddress links) may well be classed as "Data Processing," and hence be illegal in the U.S. In interconnected networks, this raises the possibility that functions can be carried out outside the jurisdiction of the country in which the operator initiating the activity is sited, and yet which is illegal in that country. This subject is treated rather fully elsewhere [60]. A clear example of this is the use of message services operated by TYMSHARE and CCA on TELENET and TYMNET. While these services are legal in the U.S., their use by UK persons connected to TYMNET by the official International Packet Switched Service is clearly technically illegal; this use would contravene the UK Post Office Monopoly.

X. Unresolved Research Questions

There are many unresolved research questions; on some of them even the present authors do not agree with each other! Primarily these questions have a technical, policy, administrative, economic, regulatory, or operational aspect, or a combination of these.

One example of this is the question of the procedures to be used for internet routing. Here there are technical questions on what is feasible in view of the technologies used in the subnets; there are policy questions on when third country routing might be allowed; there are economic considerations on how much it would cost to do the necessary protocol translation to route through third countries, and on what charges the connecting transit network might make; there may be regulatory questions on which classes of data may flow through specific countries (related to the transnational data flow regulations); and there may be operational questions on whether in the event of failure in dynamic rerouting, reestablishment could take place with sufficient rapidity.

Among the outstanding research questions are, in alphabetic order, the following.

Access Control: What are the requirements and methods of implementation of access control? How should they affect internetwork routing?

Addressing: How should the International Numbering Plan, which goes to the level of known subscribers of public networks, be extended? Should this extension be in the numbering plan itself, or should additional user and network information be supplied? Should there be local, or only physical, addressing? Should there be internetwork source routing implied by the addressing?

Broadcast Facilities: What is the role of broadcast communication facilities in the provision of internet services? Should facilities using it be offered? Should technologies supporting it use it, particularly at gateways? What are the implications on protocols, especially with respect to duplicate and error detection?

Datagram versus Virtual Call Facilities: How should datagram and virtual call facilities be interconnected? How can one compare the relative performance and costs of the implementations? What criteria should be used in any comparison? When might datagram, or alternatively virtual calls, be desirable or essential between networks?

Data Protection: What are the effects of end-to-end data encryption on protocol translation?

Flow and Congestion Control: To what extent should one adopt congestion and flow control between gateways and their feeding networks, between gateways directly, or between gateways and the source? What are the relative effects of just discarding packets in gateways, and relying on the end-to-end protocol to detect and compensate for this? How is charging for discarded packets arranged?

High Level Protocols: There are still many questions on what should be standardized, and how rigid the standards should be. To what extent should the individual networks support common standards, and to what extent should protocol translation be feasible technically or attractive economically? What are the costs of maintaining standards or the economic advantages of standard hardware and software? How does the technology of individual networks and the proportion of internetwork traffic affect the decisions?

Internetwork Diagnosis: There are many technical problems in isolating faults in concatenated networks. There are also organizational and economic problems on who should be responsible for their repair, and how costs for service failures should be allocated.

Performance: How do choices of design parameters, and network services, affect the costs of the individual networks? How do the individual network performances and costs scale to large networks? How do the choices affect the feasibility, costs and performance of the gateways? How do the variations in technology or choice of parameters affect the performance in interconnected networks?

Routing Policies: To what extent and when should adaptive routing be used between networks? How can one recover from the partitioning of a single network, when there are still routes existing by going through other networks? How should administrative considerations affect routing policies between networks (privacy regulations, economic considerations of internet payments, desire to provide for high availability, etc.)? When is a hierarchical organization more efficient that a direct route search?

Services: What services are needed on an internetwork level? Clearly interactive and bulk transport services must be supported. What else is needed? Should the internetwork facilities be able to support voice, telemetry, and teleconferencing? What is the cost of supporting these latter services, and what is their effect on other facilities?

X.25 and X.75 and Related Recommendations: Is X.25 suitable for transaction processing? Are the present datagram proposals adequate? How should X.25 be extended for internet addressing? How should X.25/X.75 be modified to allow the connection of private to public networks, or private networks to each other? Do the X.3, X.28, X.29 pad concepts extend well to the internet environment, or should they be modified?

XI. Conclusions

In view of all the unresolved questions discussed in Section X, most of the conclusions which can be drawn in this paper must be tentative. From the early part of the paper, we have shown that it is essential that techniques be developed for connecting computer networks. Moreover, no single set of techniques will fit all applications.

The services which will normally have to be supported are terminal access, bulk transfer, remote job entry, and transaction processing. The quality and facilities of the services required will be very dependent on the applications.

The connections between networks can be made at the level of the packet switches or of hosts, and can be on a datagram or virtual call basis. Connection at the packet-switch level requires broadly similar network access procedures, or complex protocol transformation at the gateways between the networks. If the network protocols are different, interconnection can be most easily achieved if done at the host level. The higher levels of service can be mapped at service centers, which need not be colocated with the gateways—but very different philosophies of network services can be very difficult to map. Alternatively, subscribers can implement common higher level protocols if these can be agreed upon.

The principal problems in connecting networks are much the same as those in the design of the individual networks of heterogeneous systems—but the lack of a single controlling authority can make the multinet design problem more difficult to solve. It is essential to resolve the usual problems of flow control, congestion control, routing, addressing, fault recovery, flexibility, protocol standards, and economy. The public carriers have attempted to resolve many of these problems; particularly in the areas of flexibility, addressing, and economy we feel their solutions are not yet adequate. At the higher levels of protocol, much more standardization is required before we have really satisfactory long term solutions.

The advent of international computer networks, private networks which must communicate with other private networks (even if via public ones), and the new applications of computer networks, raise regulatory and legal issues which are far from resolution.

Many technical solutions to the problems of the connection of networks are discussed in this paper. Their applicability in view of the different technical, economic, and policy constraints imposed in different countries must still be assessed.

REFERENCES

[1] L. G. Roberts, "Telenet: Principles and practice," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, pp. 315-329, 1975.

[2] W. W. Clipsham, F. E. Glave, and M. L. Narraway, "Datapac network overview," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 131-136, 1976.

[3] J. Rinde, "TYMNET: An alternative to packet switching technology," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 268-273, 1976.

[4] R. E. Millstein, "The national software works: a distributed processing system," in *Proc. ACM Nat. Conf.*, Seattle, WA, 1977.

[5] A. Danet, R. Despres, A. Le Rest, G. Pichon, and S. Ritzenthaler, "The French public packet switching service, The TRANSPAC network," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 251-269, 1976.

[6] G. W. P. Davies, "EURONET project," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 229-239, 1976.

[7] R. Nakamura, F. Ishino, M. Sasaoka, and M. Nakamura, "Some design aspects of a public switched network," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 317-322, 1976.

[8] F. A. Helsel and A. J. Spadafora, "Siemens system EDS—A new stored program controlled switching system for telex and data networks," in *Proc. Third Int. Computer Communications Conf.*, Toronto, Canada, pp. 51-55, 1976.

[9] T. Larsson, "A public data network in the nordic countries," in *Proc. Third Int. Computer Communications Conf.*, Toronto, Canada, pp. 246-250, 1976.

[10] P. T. Kirstein, "Planned new public data networks," *Comput. Networks*, vol. 1, no. 2, pp. 79-94, 1976.

[11] P. T. F. Kelly, "An overview of recent developments in common user data communications networks," in *Proc. Third Int. Computer Communications Conf.*, Toronto, Canada, pp. 5-10, 1976.

[12] —, "Public packet switched data networks," this issue, pp. 1539-1549.

[13] P. Hirsch, "SITA rating a packet-switched network," *Datamation*, vol. 20, pp. 60-63, 1974.

[14] G. Lapidus, "SWIFT network," *Data Communications*, vol. 5, no. 5, pp. 20-24, 1976.

[15] L. G. Roberts and B. D. Wessler, "The ARPA network," in *Computer-Communications Networks*, N. Abramson and F. Kuo, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1973, pp. 485-500.

[16] P. M. Karp, "Origin, development, and current status of the ARPANET," in *Proc. COMPCON73*, San Francisco, CA, Feb.-Mar. 1973, pp. 49-52.

[17] L. Pouzin, "Presentation and major design aspects of the CYCLADES computer network," in *Proc. Third Data Communications Symp.*, Tampa, FL, Nov. 1973, pp. 80-85.

[18] R. M. Metcalfe and D. R. Boggs, "ETHERNET: Distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, no. 7, pp. 395-404, July 1976.

[19] A. S. Fraser, "SPYDER—A data communications experiment," *Comput. Sci. Tech. Report*, no. 23, Bell Laboratories, Dec. 1974.

[20] R. E. Kahn, "The organization of computer resources in a packet radio network," in *Proc. Nat. Computer Conf.*, AFIPS Press, pp. 177-186, May 1975.

[21] R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzelman, "Advances in packet radio technology," this issue, pp. 1468-1496.

[22] I. M. Jacobs, R. Binder, and E. V. Hoversten, "General purpose satellite networks," this issue, pp. 1448-1467.

[23] D. Lloyd and P. T. Kirstein, "Alternate approaches to the connection of computer networks," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, ONLINE, pp. 499-504, 1975.

[24] L. Pouzin, "A proposal for interconnecting packet switching networks," IFIP Working Group 6.1, General Note no. 60, Mar. 1974.

[25] V. G. Cerf and R. E. Kahn, "A protocol for packet network interconnection," *IEEE Trans. Commun. Technol.*, vol. COM-22, pp. 637-641, 1974.

[26] C. Sunshine, "Interconnection of computer networks," *Comput. Networks*, vol. 1, 1977, pp. 175-195.

[27] CCITT, "Recommendation X.3: International user facilities in public data networks," *Public Data Networks, Orange Book*, vol. viii.2, Sixth Plenary Assembly, Int. Telecommunications Union, Geneva, Switzerland, pp. 21-23, 1977.

[28] CCITT, "Recommendation X.25: Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DEC) for terminals operating in the packet mode on public data networks," *Public Data Networks, Orange Book*, vol. VIII.2, Sixth Plenary Assembly, Int. Telecommunications Union, Geneva, Switzerland, pp. 70-108, 1977.

[29] CCITT, "Provisional recommendations X.3, X.25, X.28 and X.29 on packet-switched data transmission services," Int. Telecommunications Union, Geneva, Switzerland, 1977.

[30] CCITT, "Recommendation X.121—Int. numbering plan for public data networks," Study Group VII, Temporary Document 76-E, Int. Telecommunications Union, Geneva, Switzerland, April 25, 1978.

[31] L. Pouzin, "Virtual circuits vs. datagrams—Technical and political problems," in *Proc. Nat. Computer Conf.*, AFIPS Press, pp. 483-494, 1976.

[32] L. G. Roberts, "International connection of public packet networks," in *Proc. Third Int. Conf. Computer Communications*, Toronto, Canada, pp. 239-245, 1976.

[33] CCITT, "Recommendation X.75—Terminal and transit call control procedures and data transfer system on international circuits between packet-switched data networks," Study Group VII, Temporary Document 132-E, Int. Telecommunications Union, Geneva, Switzerland, Apr. 25, 1978.

[34] D. J. Farber and L. C. Larson, "The structure of a distributed computing system—The communication system," in *Proc. Symp. Computer Communications Networks and Traffic*, Polytechnic Institute of Brooklyn, pp. 21-27, Apr. 1972.

[35] P. Baran, "Broad-band interactive communication services to the home: Part II—Impasse," *IEEE Trans. Communications*, p. 178, Jan. 1975.

[36] R. E. Schantz and R. Thomas, "Operating systems for computer networks," *Computer*, Jan. 1978.

[37] L. Pouzin and H. Zimmermann, "A tutorial on protocols," this issue, pp. 1346-1370.

[38] F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The interface message processor for the ARPA computer network," in *Proc. Spring Joint Computer Conf.*, vol. 36, AFIPS Press, pp. 551-567, 1970.

[39] S. Carr, S. D. Crocker and V. G. Cerf, "Host-Host communication protocol in the ARPA network," in *Proc. Spring Joint Computer Conf.*, vol. 36. Atlantic City, NJ: AFIPS Press, Montvale, NJ, pp. 589-598, 1970.

[40] E. Feinler and J. B. Postel (Eds.), *ARPANET Protocol Handbook*, Network Information Center, SRI International, for the Defense Communication Agency, Jan. 1978.

[41] R. F. Sproull and R. D. Cohen, "High-level protocols," this issue, pp. 1371-1386.

[42] S. D. Crocker, J. F. Heafner, R. M. Metcalfe, and J. B. Postel, "Function-oriented protocols for the ARPA computer network," in *Proc. Spring Joint Computer Conf.*, vol. 40. Atlantic City, NJ: AFIPS Press, Montvale, NJ, pp. 271-279, 1972.

[43] S. M. Ornstein, F. E. Heart, W. R. Crowther, H. K. Rising, S. B. Russel, and A. Michel, "The terminal IMP for the ARPA computer network," in *Proc. Spring Joint Computer Conf.*, vol. 40, Atlantic City, NJ: AFIPS Press, Montvale, NJ, pp. 243-254, 1972.

[44] CCITT, "Recommendation X.21: General purpose interface between data terminal equipment (DTE) and data-circuit terminating equipment (DCE) for synchronous operation on public data networks," *Public Data Networks*, Orange Book, vol. VIII.2, Sixth Plenary Assembly, Int. Telecommunications Union, Geneva, Switzerland, pp. 38-56, 1977.

[45] CCITT, "Recommendation X.21-bis: Use on public data networks of data terminal equipments (DTE's) which are designed for interfacing to V-series modems," *Public Data Networks, Orange Book*, vol. viii.2, Sixth Plenary Assembly, Int. Telecommunications Union, Geneva, Switzerland, pp. 38-56, 1977.

[46] ISO, "High level data link control (HDLC)," *DIS 3309.2 and DIS 4335*, Int. Standards Org.

[47] V. Cerf, A. McKenzie, R. Scantlebury, and H. Zimmermann, "Proposal for an international end-to-end protocol," *Computer Communication Review*, ACM Special Interest Group on Data Communication, vol. 6, no. 1, Jan. 1976, pp. 63-89.

[48] A. S. Chandler, "Network independent high level protocols," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, ONLINE, pp. 583-602, 1975.

[49] —, "A network independent file transfer protocol," EPSS High Level Protocol Group, 1977.

[50] R. E. Kahn and W. R. Crowther, "Flow control in resource sharing computer networks," *IEEE Trans. Commun.*, vol. COM-20, pp. 539-546, 1972.

[51] L. Pouzin, "Flow control in data networks—Methods and tools," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 467-474, Aug. 1976.

[52] G. V. Bochmann and P. Goyer, "Datagrams as a public packet-switched data transmission service," Universite de Montreal, *Departement D'Informatique Report*, Mar. 1977.

[53] P. L. Higginson, "The problems of linking several networks with a gateway computer," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, ONLINE, pp. 453-465, 1975.

[54] J. Shoch, *private communication*.

[55] P. L. Higginson and Z. Z. Fischer, "Experience with the initial EPSS service," in *Proc. Eur. Computing Conf. Communication Networks*, London, England, ONLINE, pp. 581-600, 1978.

[56] D. L. A. Barber, "A European informatics network: Achievements and prospects," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 44-50, 1976.

[57] B. Cross, "General license for message conveying computers," *London Gazette*, pp. 7662-7663, May 28, 1976.

[58] J. Freese, "The Swedish data act," in *Proc. Conf. Transnational Data Regulation*, Brussels, Belgium, ONLINE, pp. 197-208, 1978.

[59] R. Turn, "Implementation of privacy and security requirements in transnational data processing systems," in *Proc. Conf. Transnational Data Regulations*, Brussels, Belgium, ONLINE, pp. 113-132, 1978.

[60] A. R. D. Norman, "Project goldfish," in *Proc. Conf. Transnational Data Regulations*, Brussels, Belgium, ONLINE, pp. 67-94, 1978.

[61] E. Raubold and J. Haenle, "A method of deadlock-free resource allocation and flow control in packet networks," in *Proc. Third Int. Conf. Computer Communication*, Toronto, Canada, pp. 483-487, Aug. 1976.

[62] J. McQuillan, "The evolution of message processing techniques in the ARPA network," *Network Systems and Software*, Infotech State of the Art Report 24, Infotech Information Limited, Nicholson House, Maidenhead, Berkshire, England, 1975.

[63] P. Curran, "Design of a gateway to interconnect the DATAPAC and TRANSPAC packet switching networks," *Computer Communication Networks Group*, E-Report E-67, University of Waterloo, Canada, Sept. 1977 (ISSN 384-5702).

[64] D. D. Clark, K. T. Pogran, and D. P. Reed, "An introduction to local area networks," this issue, pp. 1497-1517.

PROTOCOLS IN A COMPUTER INTERNETWORKING ENVIRONMENT

Ray I. McFarland Jr.

United States

Department of Defense

ABSTRACT

This paper presents a model for protocol layering in a computer internetworking environment. Four distinct protocol layers are identified; the network layer, the internet layer, the transport layer, and the application layer. The functions of each are defined. Gateway functions are also addressed in the discussion of the internet layer. A set of protocols are defined for the transport layer based on communications requirements; a reliable data protocol, a datagram protocol, a speech protocol and a real time protocol. Alternatives for standardization at the network, internet and transport layers are presented. Some impacts of choosing each alternative are discussed.

INTRODUCTION

Computer networks are playing a more important role every day within the Department of Defense. More and more projects situated on different networks are finding that they have a requirement to intercommunicate. These requirements, in addition to the direction being taken by DoD to have one long haul common carrier (that is, AUTODIN II) rather than many large geographically dispersed special purpose networks, are leading to the development of computer internetworking strategies.

In order to exchange information in a meaningful way through networks of computers, there must be an agreed upon protocol, or set of protocols. This paper will present a protocol layering model for a computer internetworking environment. Four distinct protocol layers will be identified and their functions defined. The functions of network gateways will also be addressed by the model.

Alternatives for standardization of three of the four layers will be presented. Some of the impacts of the various alternatives will also be discussed.

A PROTOCOL LAYERING MODEL

One of the definitions Webster's New World Dictionary of the American Language gives for protocol is "the code of ceremonial form and courtesies, of precedence, etc. accepted as proper and correct in official dealings, as between heads of states or diplomatic officials" (1). In much the same way, a communication protocol is a defined set of control procedures and formats for the transmission of information which is agreed to by the owners of the communications gear involved. Protocols can be divided into layers in such a way that each layer implements certain control procedures, which provide a set of communication properties to the layers above it. Ideally, the higher layer protocols should be able to take advantage and build on the properties provided by the layers beneath it.

There are four major protocol layers emerging in the DoD computer internetworking environment. We call them the network layer, the internet layer, the transport layer and the application layer. These four layers are illustrated in Figure 1. For one example which will briefly show how the layers fit together, consider what a message would look like on a network with all four layers present. The first item in the message is the network layer header, which contains the control information for the network layer. Next is the internet layer header, followed by the transport layer header, an application layer header if the application control is not implicit in the data, and finally the data itself. See Figure 2. This section will define the control procedures of each layer.

The ARPAnet will be used in the following discussion to provide examples. Further information on the ARPAnet is given in (2) and (3). The term 'packets' will be used here to refer to integral units of information transmitted on a network. The term will be qualified, as in 'ARPAnet packets', when referring to specific implementations to avoid ambiguity.

Network Layer

The 'lowest' (furthest removed from the user) layer is the network layer. This layer consists of the control procedures required to actually transmit packets physically between two subscribers on one network (one or both of which could be a gateway to another network), and defines the interface to higher layer protocols. (The concept of a gateway is defined in the Internet Layer section of this paper.) For example, the ARPAnet's network layer consists of the IMP-IMP protocol, (the IMP, which stands for Interface Message Processor, is the ARPAnet packet switch), and that portion of the Network Control Program which implements the Host-IMP protocol, which is usually referred to as the Bolt, Beranek and Newman 1822 Interface Specification (3). Of necessity, this protocol layer is dependent on the specific network technology. The network protocol for an ARPAnet type packet switching network will be different than one for a ring network, a packet radio network, or a satellite network.

There are two minimal control procedures which all network layers must implement, addressing and routing. As noted in (4) and (5), these are not the same thing. An address defines where an entity is located and a routing mechanism defines how to get from one address to another. Every network must have the ability to identify the locations of machines on it (i.e., have an addressing scheme). In addition, they must have a scheme for routing packets between two points, whether it is a static or dynamic scheme, predetermined or based on a heuristic algorithm.

There are, of course, additional control procedures which a network layer may provide. One of the most important from a network health standpoint is flow control. A properly implemented flow control scheme allows the network to protect network resources from congestion. Two ways of doing this are throttling network input to a certain maximum level and redirecting traffic around a congestion point with a dynamic routing scheme. For example, the ARPAnet allows only eight ARPAnet messages at a time between any two hosts, while the dynamic routing algorithm was intended (in part) to handle traffic congestion between any two adjacent IMPs.

This layer may also provide error detection, either on a hop by hop basis or on a point of entry to point of exit basis, or some combination of the two. A strictly hop by hop scenario is the strategy typically used i. a store and forward network. When a switch receives a packet it sends an acknowledgment to the adjacent sending switch, which is then allowed to release its copy. One disadvantage of this scheme is that, if a network malfunction occurs, it is possible to lose messages. A switch crashing after having acknowledged a message but before sending it on is one example. For a strictly point of entry to point of exit scenario, the destination switch would acknowledge the packet to the source switch only after it had successfully passed it to the intended destination. (This is also referred to as end to end acknowledgement.) Thus, if the network malfunctions and drops a packet, recovery is still possible since the source switch has maintained a copy. An acknowledgment from the destination switch had not yet been received by the source switch. The ARPAnet actually uses a combination of the two, with inter-IMP acknowledgments as an ARPAnet packet traverses the network and a Ready For Next Message (RFNM) which is sent from the destination IMP to the source IMP.

A network may provide a form of fragmentation, where messages delivered to the network are broken down into smaller units for transmission. This is another mechanism commonly used by networks to maximize their resources. At the destination switch, the network is responsible for reassembly of the fragments it has created. The ARPAnet breaks messages down into packets for transmission across the IMPs and reassembles the messages at the destination IMP.

A network may also implement some form of precedence strategy for high priority packets.

The overall capability this layer provides is the capability to physically move packets of information between the network's subscribers (or gateways), without requiring the higher layers to have knowledge of the switch procedures or formats.

Internet Layer

This layer consists of the control procedures required to allow internet packets to traverse multiple networks between any two hosts. This protocol is usually implemented within hosts and gateways. The gateway attaches to two or more networks and is the bridge between the networks over which the internet packets flow. The primary function of the gateway is the passing of control information and data between two networks. In addition, the gateway must also determine what network layer control procedures are to be invoked for a particular packet. The gateway derives this information from the internet protocol header. It should not translate between the two network layers. It is preferable to derive the control information needed from the internet header and allow the destination network to implement the required control within the context of its own control constructs rather than try and match up the control constructs of two network layer implementations. In general, the translation of control constructs from one network layer implementation to another is cumbersome and a one-to-one mapping of the control constructs of two network protocols is rarely obtainable. The best chance to achieve such a mapping is if the two network protocols are exactly the same, but even then some 'fudging' of the protocol may be needed for an implementation, (e.g., the implementation and interpretation of a RFNM, which is end to end intranet, but is not end to end internet).

There are three minimal control procedures which the internet layer must implement: addressing, routing and fragmentation. The internet address must be able to uniquely specify a location on a set of networks and also identify the proper transport layer processing software to which the packets should be sent. The usual mechanism for doing this is a hierarchical addressing scheme, such as '<network address><host address><transport protocol processing module address>'. Other addressing schemes have also been devised to try and reduce the overhead of the addressing field in the header. Whatever schema is implemented, the gateway must be able to map the internet address to a specific network address, either the intended destination host or another gateway.

Gateways determine the routing at the internet layer when more than one gateway must be traversed to reach the destination host. There may even be two different gateways between the destination network and an intermediate network. The gateway between the source network and the intermediate network would then be able to choose which gateway to route the packets through. This can be even more significant when the destination network is at least three networks away from the source network. In this case, the internet routing could actually determine which network(s) the packets are to flow through. For example, network A may attach to network B through one gateway, network B attach to network C through one gateway and to network D through one gateway, network C attach to network E through one gateway, and network D attach to network E through one gateway (see Figure 3). Two alternate routes then exist from network A to network E. One route is A-B-C-E, the other is A-B-D-E. The gateway can adjust to gateway (or intermediate network) congestion by dynamically choosing which gateway individual packets should go through. This is analogous to the dynamic routing algorithm in the ARPAnet mentioned earlier. In the same way that ARPAnet packets of a given message are not constrained to a specific series of IMPs, packets of a given connection should not be constrained to a given series of gateways. However, for this to be possible, the packets of a higher layer protocol connection must not be constrained to go through one specific gateway or series of gateways to reach their destination. (The concept of a connection is defined in the Transport Layer section of this paper.) The gateway should be oblivious to the existence of connections. An additional advantage gained from this approach is the lack of a need for the gateway to store connection state information, allowing for a simple and more efficient gateway. The proper place for connection state information is at the next layer, the transport layer.

The third minimally required control procedure is fragmentation. (Fragmentation in a specific gateway is necessary when one of the attached networks has a maximum packet size which is smaller than one of the other attached networks' maximium packet size. We will assume this as the general case in this discussion.) A gateway must have the capability to interface two networks which have different maximum size packet lengths. To do this, the gateway must be able to break down a packet into fragments, each looking like an integral packet to the network with the smaller size maximum packet length. The internet protocol must, therefore, provide the means for identifying fragments and for sequencing them so that they can be reassembled.

It is important to note that if reassembly of fragments is done at the gateway, then all of the fragments which make up the larger packet are constrained to go through the same gateway when leaving a network. For error recovery by retransmission, the retransmission of the original packet must be constrained to the originally addressed gateway, which may counter any dynamic routing algorithm that may exist at the internet layer. Without dynamic routing, the gateway is a point of single failure for all connections that go through it. With dynamic routing and the requirement for reassembly of fragments at a gateway, the gateway may require some knowledge of the formats and error recovery procedures of all the transport layer protocols which can pass through it. For example, to decide whether to hold or discard a partial packet, the gateway may have to know which transport level protocols retransmit and which do not. This violates the premise that protocol layers should be kept seperate and distinct, and not rely on the formats and prcedures of protocols that are at a higher layer. A second disadvantage to dynamic routing with reassembly at a gateway is that a gateway's buffers may be tied up waiting for a 'lost' fragment of a packet while the retransmitted packet has already passed through an alternate gateway.

Where then should the fragments be reassembled? If the reader will recall, we have been discussing the functions of a gateway which are needed to process the internet protocol layer. Yet, nowhere was it mentioned that the protocol layer is either created or terminated at the gateway. The information has already existed for the gateway to process it. All well defined protocols have (at least) two distinct ends, the 'ends' for the internet layer are at the source and destination hosts. The software which implements these internet layer procedures at the hosts could be loosely referred to as 'half a gateway', since it only connects to one network. The source gateway-half is responsible for forming the internet header, deriving the necessary control information from either the host directly or from the transport layer header (e.g., precedence, sequencing information, etc.). The destination gateway-half is responsible for reassembling the fragments and demultiplexing the internet packets to the proper transport protocol processing modules. Of course, some host implementations may not have the capability to reassemble fragments. In this case, the internet protocol must allow for the source host to declare an option of 'do not fragment this packet'. Gateways which have to fragment these type of packets would either discard them or reroute them to another gateway. In fact, this information is one of the things which could go into the routing scheme which gateways implement (including the gateway-half at the source).

For retransmission efficiency, one might wish to trade off some of the flexibility in the previously described dynamic routing scheme for a simple error detection and retransmission procedure between any two gateways. In this case, there is still no need to correlate two different fragments at an intermediate gateway. Individual network packets, when retransmitted from one gateway to the other, would be constrained to go to the gateway addressed originally. However, if the gateway fragments a packet; then new internet checksums are computed for each fragment (which become individual packets for the next network). What is lost is the ability to address the retransmitted packet to an alternate gateway if the gateway addressed originally is overloaded or has crashed. (A more complex procedure could allow for both dynamic routing and gateway error detection and retransmission. The complexity is in the bookkeeping required at the sending gateway to allow it to properly process any returning acknowledgment.)

The internet layer presents the capability to move the packets over many networks and hides the necessary details of gateway functions from the higher layer protocols. For instance, the transport protocol layer need not worry about gateway addressing or network routing.

The functions of a gateway are intimately tied to the internet protocol which it implements; however, the gateway and the internet protocol are not synonymous. A gateway may have other functions beyond the strict implementation of the internet protocol. These functions must, however, meet the requirement that the gateway remain simple, efficient and easily maintainable. One such function has already been mentioned, the maintenance of network congestion information, which contributes to the routing decision at the internet layer. Other functions would be accounting and reporting to some network control center(s) for 'state of the gateway' information, such as queue lengths, traffic density, etc. A gateway could also act as an agent of a network access control center, for network accountability and self protection requirements.

Transport Layer

This layer consists of the control procedures necessary to deliver packets between two application processes on different host computers, whether the hosts are on the same or different networks. Within networks, this layer has been commonly referred to as the Host to Host protocol. The transport protocol modules interface to the application layer software modules (or some host to front end protocol module where the transport protocol is terminated in a front end). It is at the transport layer that explicit connection information makes sense, since connections are thought of as explicitly defining the transmission path between two (or more) application layer processes. Connection state information and connection maintenance is one responsibility of the transport layer protocol.

One type of transport layer protocol is not sufficient for most users. Different users have different communication requirements. These requirements are usually a function of the reliability level needed, timeliness of delivery, and the need for sequenced delivery between two application processes in different hosts. A protocol which attempts to solve all the needs of all users will probably be worthless to everyone. (It will at the least be grossly inefficient, something which cannot be tolerated in a communications environment.)

Four types of protocols seem to be needed at this level; a reliable data protocol, a datagram protocol, speech and a real time protocol. There may be more, but this paper will limit its discussion to these four. The following discussion will not attempt to define all the control procedures a particular transport protocol should have, but will give a sufficient number to allow the reader to distinguish between the four types.

The reliable data protocol is characterized by the need for a high level of reliability and the need for sequential delivery of the packets transmitted between two processes.

The need for sequenced delivery leads to the concept of a communications connection existing between two processes. The defining characteristics of a connection are: (i) each end has an explicit name and is associated with a specific process; and (ii) the packets are sequenced only with respect to the order of transmission on their connection, and independent of the sequencing of packets on other connections. The reliable data protocol is responsible for implementing the concept of a connection. This includes, but is not limited to, the opening and synchronizing of a connection, the maintenance of an open connection and the corresponding connection state information, the resynchronization of a connection if and when necessary, and the closing of a connection. In short, all the connection management functions.

The reliability requirement is satisified by a mechanism in the reliable data protocol which guarantees packet delivery at the receiver. To do this, the protocol must provide a sufficiently robust error detection scheme (which is usually some form of cyclic redundancy check). The protocol must also provide a way to positively acknowledge packets and must be persistent in the retransmission of packets until, positive acknowledgment of an error free delivery is received or an abort time out period expires. If the abort occurs, the protocol must be able to identify for the user which packets were received and which were not. Since fragmentation, dynamic routing strategies and packets received in error can result in out of order reception of error free packets and possibly duplicate reception of some packets, the protocol must compensate by being able to detect duplicate packets and reorder the original packets prior to delivery to the receiving process. The reordering of packets before delivery also aids in the identification of received packets for an aborted connection.

One last functional requirement for the reliable data protocol is the maintenance of a flow control strategy. At this layer, the flow control strategy is aimed at the management and protection of host resources, such as the amount of buffers available for received traffic.

The second type of transport protocol is the datagram protocol. This protocol is charecterized by the lack of a need for sequenced delivery and the decreased level of reliability required.

The datagram 'service' basically has a single packet orientation with no relation existing between packets, something like a telegram service. Becuase of this characteristic, there is really no need for all the overhead involved in the maintenance of explicit connections. In fact, a number of the functions that a reliable data protocol provides, such as flow control, are not even needed. Establishing a connection for a single packet can be a waste of transmission resources and be very inefficient. The datagram protocol must provide a way to identify individual packets, but it does not have to sequence them.

Some individuals within the ARPAnet community who have expressed a desire for a datagram have indicated that their application layer protocol will provide the degree of reliability wanted. An application layer protocol would simply retransmit until some form of positive acknowledgment (either explicit or implicit, such as the results of some initiated action being returned) has been received. The datagram protocol, then, must implement some form of error detection for error free delivery of packets, but it does not have to guarantee the arrival of the packets or reorder them or detect duplicates as the reliable data protocol does.

A speech protocol is a third type of transport protocol. This protocol is characterized by the need for sequenced delivery and the need for very timely delivery.

As in the case of the reliable data protocol, the speech protocol requires a connection management mechanism to preserve logical relationships among the packets through sequenced delivery. Flow control may or may not be required, depending on the particular speech application and available resources.

Individuals who are working on packetized speech have indicated that they would prefer to trade off a highly reliable protocol for one which is very timely in its delivery of packets. (Note that the retransmission of packets received in error or possibly lost in the network reduces the timeliness of their delivery.) Reordering is required for two reasons, ordered delivery to the application process and for the detection of late arriving packets, which are discarded. Though they require reordering, they do not worry about lost or undelivered packets. Gaps in the reordered packets delivered to a speech algorithm do not severly effect the quality of the speech, unless the gap is significantly large. The protocol must then provide for a way of sequencing the packets, reordering them and detecting duplicate fragments. But it does not necessarily have to implement a positive acknowledgment scheme for the purpose of retransmission of packets which were lost or received in error. An error detection scheme is required so that error packets can be discarded at the receiver.

The fourth type of transport layer protocol is for real time traffic. This is the most difficult type of traffic to deal with, since it is characterized by a need for sequenced delivery, and the need for both highly reliable and timely delivery. The distinction made between speech and real time traffic is that with speech, which is ultimately intended for the human ear, all the traffic is not required for intelligent processing of the information. The ear is an excellent filter which can integrate over missing traffic, as long as the gap is not too large. Real time traffic is more in the character of data as described under the reliable data protocol, such as the remote control of a sensitive production process. All of the information is required for processing. The requirements for both high reliability and timely delivery effects the technology choices of the networks over which the information must pass.

The types of functions that this protocol must have is a combination of those defined for the reliable data protocol and the speech protocol.

Transport layer protocols provide the 'transportation medium' to the protocols at the application layer. The transport layer hides from the application layer the implementation details of connection management and flow control, sequencing and packet errors (except for the datagram protocol). Packets generally will be delivered just as they were sent, except as noted earlier.

Application Layer

This layer defines the control procedures between two application processes necessary to accomplish a given task. For example, the control procedures for an information retrieval package might be search, extract, sort, merge, etc.

The application layer protocols provide the capability for two software processes to work together. This layer always exists, whether explicitly or implicitly, whenever two processes are required to communicate, be they on the same machine, the same network, or different networks. When this layer is explicitly defined in the design stage of a project, it increases the understandability of the software requirements and aids in the definition of clean software interfaces.

PROTOCOL STANDARDIZATION

Recalling our initial definition of the word 'protocol', it is interesting to consider what happens when two societies, which have different protocols (or shall we say 'standards' of behavior?) interact. The results can be humorous, confusing, irritating, and sometimes even violent, all at the same time. The effects can be the same when different computer networks, which have different protocols, want to intercommunicate. And this leads to the question of standardization and the degrees of standardization. How much is sufficient? How much is too much? What are the issues? These questions are questions which must be addressed, and they cannot be addressed in a vacuum. What one organization does and how they do it has an impact on other organizations, and vice versa.

Standardization of the Network Layer

The definition of the network layer protocol is primarily a function of the technology of the network because this is the protocol responsible for actually moving packets through the network's physical switches. The choice of a specific technology for networks is usually driven by the intra-network requirements, as it should be. Real time requirements also play a large role in the choice of network technology.

A network implementer may choose from a number of technologies for his network. Some network implementers might choose an R/F cable (such as the MITRE bus), or radio (such as ARPA's Packet Radio), or the use of commercial land line (such as the ARPAnet), to name just a few. It is, therefore, not really practical to argue for one, or even a few, standard protocols at the network layer. The disadvantages of forcing every network to use the same or extremely similar technologies to meet their requirements far outweigh the advantage of all networks being able to interconnect at the network layer, especially when a strategy exists which allows intercommunication between networks without imposing this type of restriction (one example being the protocol layering model given in this paper).

It does, however, make sense to argue for a standard interface to the higher layer protocol. This would allow relatively easy conversion between two network technologies when a network is upgraded and to some extent allows for transportability of higher layer protocol implementations.

Standardization of the Internet Layer

The internet layer is where the real impact of standardization or the lack thereof occurs.

There are three alternatives for implementing the internet layer: (i) define one standard internet layer protocol to be used within one communication community (such as DoD); (ii) do not standardize at all and allow all networks to implement their own internet layer protocol, requiring a protocol translation at the gateway for the internet protocol; and (iii) do not even have an internet protocol and relegate the functions to either the network layer or the transport layer. Implementation of the internet layer procedures in the network layer protocol now implies that a protocol translation must occur at the network layer. The net effect, from a standardization point of view, is the same as alternative (ii). Implementation in the transport layer protocol falls under the category of standardization for transport layer protocols. The following discussion focuses on alternative (i), a single internet protocol standard and alternative (ii), the lack of a standard.

There are a number of advantages to a standard internet protocol, most of which are reflected in the size and simplicity of the gateway. A standard protocol leads to a common approach for gateway construction, where many copies of the heart of one gateway (the internet protocol implementation) can be made and supplied to many networks. Networks would be responsible for interfacing their particular network layer to the internet layer. These modules should already exist at the network's hosts, where a gateway-half is implemented. This approach can reduce the net development costs for gateways, and software development is an expensive proposition (as we continue to experience). It would also reduce the software maintenance costs. It is possible to have the types of congestion control based on dynamic routing dicussed earlier that would probably not be possible if protocol translation were required, resulting in a form of more reliable service (reliable here in the sense that a gateway is not necessarily a single point of failure or congestion for a user's communication).

The gateway does not have to worry about connection management (which is non-trivial) as it would have to do if the procedures of this layer are relegated to the transport layer, unless a standard transport protocol is implemented. This approach maintains a transparency to all the transport layer procedures. And there may very well be more than one transport layer protocol to worry about. This results in a much simpler, more efficient, and probably more reliable gateway.

On the other hand, one standard internet layer protocol does have its disadvantages. It requires political agreement between organizations which is not always easy to obtain, especially when an organization has already invested resources to go in a different direction. Technical conformity is required, something that all skillful protocol designers have trouble living with. And it provides less flexibility to change, at least at the internet layer, to meet new requirements.

When some of the procedures which we feel should be in the internet layer have been relegated to the transport layer, the discussions of the section on transport layer protocol standardization also apply.

Standardization of the Transport Layer

Standardization of the transport layer protocol can also have a significant impact, but not as large as that of the internet layer (unless the internet layer's control procedures are implemented at the transport layer). It is possible to speak about standardization of the transport layer within a community since it is possible to define a set of closed communities which share a common network or set of networks. By a closed community, we mean that a host belonging to that community will never talk to a host outside of the community. But, when two closed communities which have their own "standard" transport layer protocols develop a requirement to intercommunicate, their protocols are no longer standard within the expanded closed community. They will face the same difficulties that other non-standard implementations will face when trying to intercommunicate.

There are three alternatives for standardizing transport layer protocols: (i) to have one standard protocol for all types of traffic; (ii) to have a set of standard protocols based on traffic type (as defined earlier); and (iii) to allow each network to develop their own transport layer protocols, i.e., not to standardize.

When one protocol is defined to answer the needs of all users, it will probably end up not serving any very well. Its generality will require a large amount of overhead, resulting in potential severe inefficiencies. It will be extremely large, possible eliminating smaller hosts from even implementing it. This approach is not a realistic alternative.

When protocols are based on the type of traffic, one protocol per type, then each protocol can be optimized to handle the communication characteristics of the traffic for which it was intended. This alternative eliminates the need for severe overhead and size. Of course protocols will continue to be enhanced, but as long as one maintains backward compatibility, this should not present a significant problem. A host will also not have to worry about implementing many different protocols for each new communication requirement which comes along.

It should be recognized that a standard set of transport level protocols still allow for divergent hardware technologies at the network layer and minimizes the impact when a network decides to change its network technology.

Development costs would be small (except for the first round), since the same protocols developed for different machines would be available off the shelf for machines of the same type that connect to a network later.

This alternative also buffers the transport layer protocols from gateway malfunctions. If a gateway were to crash (assuming the internet control procedures are in an internet layer protocol), the transport protocol does not have to worry about messy connection cleanup, since there is no transport protocol translation at the gateway.

The third alternative, the no standard approach, has the advantage of allowing the transport level protocols to be very finely tuned to specific applications. It does not have to compromise its technical approach for other requirements. Also, the very hard to get political agreements are not necessary for this approach.

Standardization, Some Concluding Remarks

We are in basic agreement with the studies which advocate a single standard internet layer within a community. We also contend that a set of transport layer protocols is what is required, not just a reliable data protocol. There, unfortunately, are no hard answers yet as to which way is best, because the implementations for internetworking are still in the study and experimental phases. The model we have presented in the first part of this paper is consistent with the ARPA approach to internetworking.

Should DoD choose to implement standard protocols within the context of a closed community, it is important to define that community judiciously. There are definitely impacts on DoD agencies and departments from the way other members of the same community design and implement their protocols.

The area of interconnection of computer networks is an exciting and interesting one, but many difficult questions remain unanswered.

ACKNOWLEDGMENT

The author wishes to recognize the contributions of the many individuals involved with the ARPA Internetworking Project and within the various agencies of the Department of Defense, whose conversations with the author made this paper possible. Unfortunately, they are too numerous to mention here by name.

REFERENCES

(1) Webster's New World Dictionary of the American Language, 2nd ed. (New York, 1972).

(2) Feinler, E. and Postel, J. "ARPAnet Protocol Handbook", SRI International, ARPA Network Information Center, (Menlo Park, CA, Jan 1978).

(3) Bolt, Beranek and Newman, "Specifications for the Interconnection of a Host and an IMP", Report No. 1822, (Cambridge, MA, May 1978).

(4) Cohen, D., "On Names, Addresses and Routings", ARPA Internet Experiment Note #23, Information Sciences Institute, (Marine Del Rey, CA, Jan 1978).

(5) Shoch, J., "Inter-Network Naming, Addressing, and Routing", ARPA Internet Experiment Note #19, XEROX Palo Alto Research Center, (Palo Alto, CA, Jan 1978). (Also published in COMPCON 78).

(6) Cerf, V. and Kahn, R., "A Protocol for Packet Network Intercommunication", IEEE Transactions

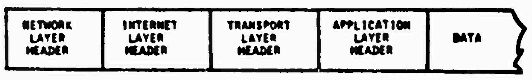
on Communications, Vol. COM-22, No. 5, (May 1974).

[7] Garlick, L., et. al. "Issues in Reliable Host-to-Host Protocols", Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, (Berkeley, CA, May 1977).

[8] Information Sciences Institute, "Internet Datagram Protocol, Version 4", ARPA Internet Experiment Note #80, (Marina Del Rey, CA, Feb 1979).

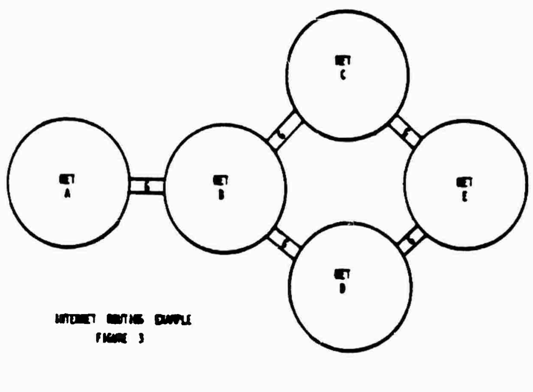
[9] Information Sciences Institute, "Transmission Control Protocol, Version 4", ARPA Internet Experiment Note #81, (Marina Del Rey, CA, Feb 1979).

[10] Sunshine, C., "Interconnection of Computer Networks", Computer Networks, Vol. 1, (1977).

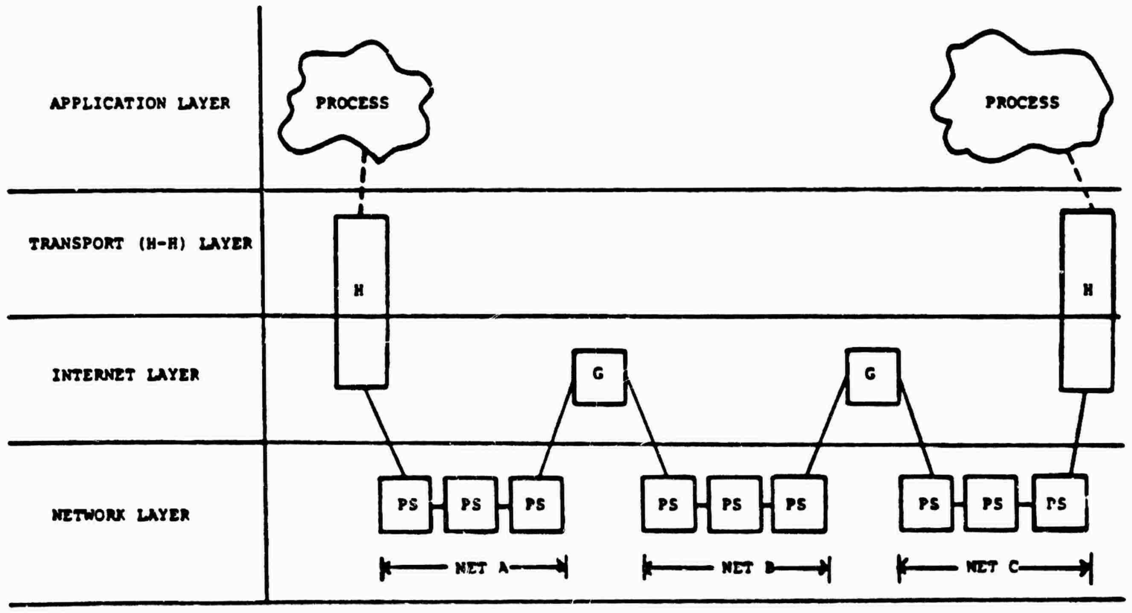


PROTOCOL LAYER HEADERS

FIGURE 2



INTERNET ROUTING EXAMPLE
FIGURE 3



H = HOST

G = GATEWAY BETWEEN 2 NETWORKS

PS = PACKET SWITCH

PROTOCOL LAYERING DIAGRAM

FIGURE 1

Internetwork Protocol Approaches

JONATHAN B. POSTEL

(Invited Paper)

***Abstract*—The motivation for interconnecting networks is to provide one or more consistent services to the set of users of the interconnected networks. To provide these services either new end-to-end service protocols must be defined or the service protocols of the individual networks must be made to interwork. In either case the issues of addressing, routing, buffering, flow control, error control, and security must be considered. Two examples of interconnection strategy are examined: the interconnection of X.25 networks, and the interconnection of ARPA research networks. The models for interconnection of networks and the role of internetwork protocols are discussed.**

INTRODUCTION

THE motivations for constructing computer communication networks—data and program exchange and sharing, remote access to resources, etc.—are also motivations for interconnecting networks. This follows from the observation that the power of a communication system is related to the number of potential participants.

This paper first discusses a few key concepts involved in computer communication networks. The view that computer networks provide an interprocess communication facility is presented. The datagram and virtual circuit services are compared. The interconnection device or gateway is discussed. The relation of the interconnection issues to the open systems architecture is described.

In this paper, two approaches to internetworking are characterized: the public data network system as implied by the CCITT X.75 Recommendation and the ARPA experimental internetwork. These two systems illustrate the virtual circuit and the datagram approaches to network interconnection, respectively. The vast majority of the work on interconnecting networks falls into one of these two approaches.

INTERPROCESS COMMUNICATION

While discussing computer communication, it is useful to recall that the communication takes place at the request and agreement of processes, i.e., computer programs in execution. Processes are the actors in the computer communication environment; processes are the senders and receivers of data. Processes operate in host computers or hosts.

The protocols used in constructing the communications capability provide an interprocess communication system. Fig. 1 shows how the combination of the network and the host network interface (hardware and software) can be viewed as providing an interprocess communication system.

When a new host computer is to be connected to an existing network, it must implement the protocol layers necessary to match the existing protocol used in the network. The new host must join the network-wide interprocess communication system so the processes in that host can communicate with processes in other hosts in the network.

The interconnection of networks requires that the processes in the hosts of the interconnected networks have a common interprocess communication system. This may be achieved by converting the networks to a new interprocess communication system, by converting one or more levels of protocol to new protocols, or by translating between pairs of interprocess communication systems at their points of contact.

DATAGRAMS AND CIRCUITS

Two types of service are commonly discussed as appropriate for the network-provided interprocess communication service: datagrams and virtual circuits.

Datagrams are one-shot simple messages. They are inherently unreliable since they travel one-way and are not acknowledged. Datagrams may also arrive in a different order than sent (at least in some networks). Datagrams are simple to implement since they do not require the networks or gateways to record and update state information. Datagrams must carry complete address information in each message. The transmission of datagrams by a process is via send and receive actions.

Virtual circuits (or connections) are designed to be reliable and to deliver data in the order sent. Implementation of virtual circuits is complicated by the need for the networks or gateways to record and update state information. Virtual circuits are created through an exchange of messages to set up the circuit; when use terminates, an exchange of messages tears down the circuit. During the data transmission phase, a short form address or circuit identifier may be used in place of the actual address. To use a virtual circuit a process must perform actions to cause the virtual circuit to be created (call setup) and terminated, as well as the actions to send and receive data.

Datagrams provide a transaction type service while virtual circuits provide a connection type service. Each of these services is needed in a general purpose communication environment. Datagrams are most efficient for transaction type information requests such as directory assistance or weather reports. Virtual circuits are useful for terminal access to interactive computer systems for file transfer between computers.

Manuscript received June 15, 1979; revised December 18, 1979. This work was supported by the Advanced Research Projects Agency under Contract DAHC15 72 C 0308, ARPA Order 2223.

The author is with the Information Sciences Institute, University of Southern California, Marina del Rey, CA 90291.

0090-6778/80/0400-0604$00.75 © 1980 IEEE

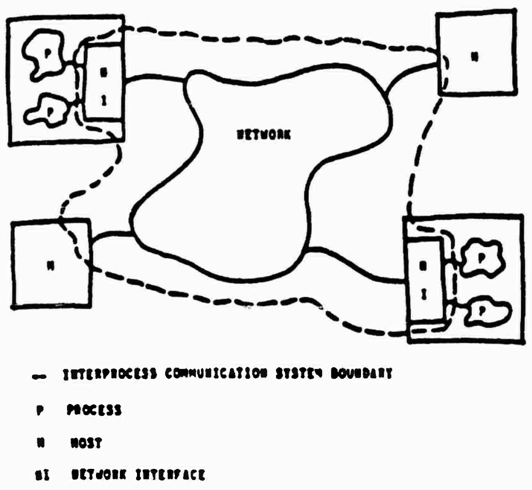


Fig. 1. Communications network.

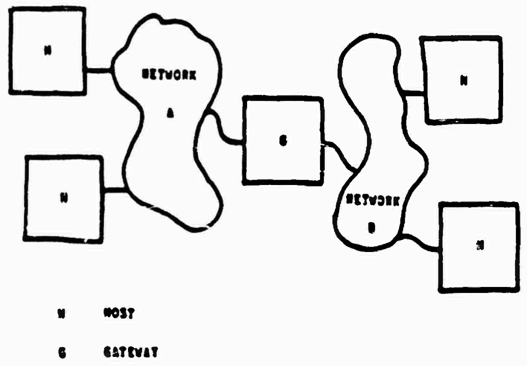


Fig. 2. Interconnected networks.

GATEWAYS

Two or more networks are connected via a device (or pair of devices) called a gateway. Such a device may appear to each network as simply a host on that network (Fig. 2).

Some gateways simply read messages from one network (unwrapping them from that network's packaging), compute a routing function, and send messages into another network (wrapping them in that network's packaging). Since the networks involved may be implemented using different media, such as leased lines or radio transmission, this type of gateway is called a media-conversion gateway.

Other gateways may translate the protocol used in one network to that used in another network by replacing messages received from one network with different messages with the same protocol semantics sent into another network. This type of gateway is called a protocol-translation gateway.

It should be clear that the distinction between media-conversion and protocol-translation is one of degree: the media-conversion gateways bridge the gap between differing link and physical level protocols, while protocol-translation gateways bridge the gap between differing network and higher level protocols.

The translation approach to network interconnection raises several issues. Success in protocol translation seems inversely correlated with the protocol level. At the lower levels, protocol translation causes no problems because the physical level and link levels are hop-by-hop in nature. It should be noted, though, that different protocols even at these low levels may have impact on the reliability, throughput, and delay characteristics of the total communication system.

At the network and transport levels, the issues of message size, addressing, and flow control become critical. Unless one requires that only messages that can be transmitted on the network with the smallest maximum message size be sent, one must provide for the fragmentation and reassembly of messages. That is, the division of a long message into parts for transmission through a small message size network, and the reconstruction of those parts into the original message at the destination. The translation of addresses is a difficult problem when one network or transport level protocol provides a larger address space than the corresponding protocol to be translated to. When end-to-end flow control mechanisms are used, as they commonly are in transport level protocols, difficulties arise when the units controlled are different. For example, when one protocol controls octets and the corresponding protocol controls letters. More difficulties arise with potential difference in the model of flow control. For example, a difference between pre- and postallocation, or between the allocation of buffer space and the allocation of transmission rate.

At higher levels, the problems are more difficult because of the increased state information kept and the lower likelihood of one-to-one translation of individual protocol messages. A further difficulty is that each level further multiplexes the communication so that each connection or stream or channel or virtual circuit must be separately translated. It should be noted that neither of the specific interconnection approaches discussed in this paper attempts higher level protocol translation.

Gateways may be thought of as having a "half" for each network they interconnect. One could model the operation of a gateway as having each gateway-half contain procedures to convert from a network specific protocol into a standard protocol and vice versa (Fig. 3).

RELATION TO OPEN SYSTEMS ARCHITECTURE

In relation to the open systems architecture, the interconnection of networks focuses on levels 3 and 4 [1].

To review, the open systems architecture defines the following levels of protocol:

Level	Function
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Link
1	Physical

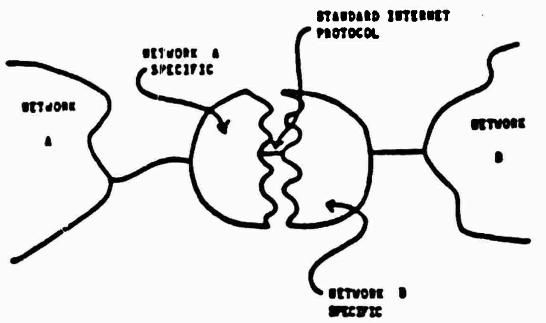


Fig. 3. Gateway halves.

The lower levels, the physical and the link levels, are hop-by-hop in nature and present no interconnection issues in terms of compatibility, although there may be some performance concerns.

The higher levels, the session level, the presentation level, and the application level, have so many compatibility requirements that it seems quite unlikely that interconnection of different protocols at those levels will be workable.

Thus, it is at the network level and the transport level that the interconnection of networks finds issues of concern.

The network level corresponds to the interface to datagram service, and the transport level corresponds to the interface to virtual circuit service.

In some networks, the network level and datagram service have been hidden from the user, forcing consideration of network interconnection at the transport level.

INTERCONNECTION OF X.25 NETWORKS

Introduction

The public data networks (PDN's) that follow the CCITT X.25 Recommendation [2] are to be interconnected via an interface specified in CCITT Recommendation X.75 [3]. Recommendation X.25 specifies the interface between the customer's equipment, called the data terminal equipment (DTE); and the network equipment, called the data circuit-terminating equipment (DCE). Recommendation X.25 implies a virtual circuit operation. Thus, the PDN's offer an interface to a virtual circuit transport level protocol. Fig. 4 shows the model of a PDN virtual circuit.

The interface between two PDN's specified in Recommendation X.75 is quite similar to that in Recommendation X.25. The equipment on either side of this interface is called a signaling terminal (STE). The STE-STE interface is much like the DTE-DCE interface. The STE-STE interconnection is a split gateway with each gateway-half in a physical device controlled by the PDN connected to that gateway-half. Fig. 5 shows the interconnection of PDN's.

The interconnection of PDN's via X.75 interfaces results in a series of virtual circuits. Each section is a distinct entity with separate flow control, error recovery, etc. Fig. 6 shows a PDN transmission path with two virtual circuits (VC's) and five separate flow control (FC) steps.

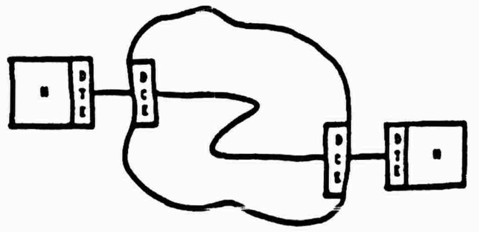


Fig. 4. PDN virtual circuit.

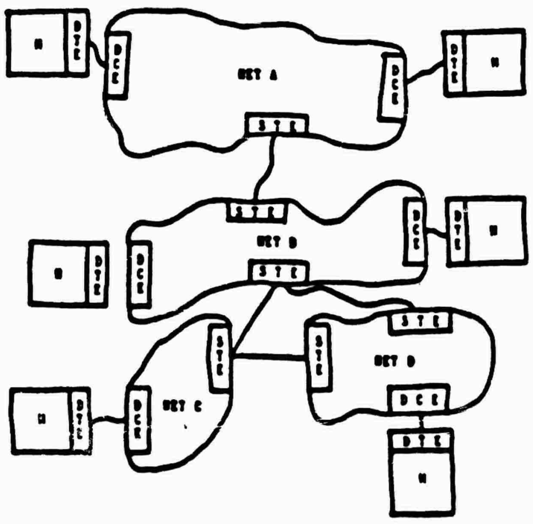


Fig. 5. Interconnection of PDN's.

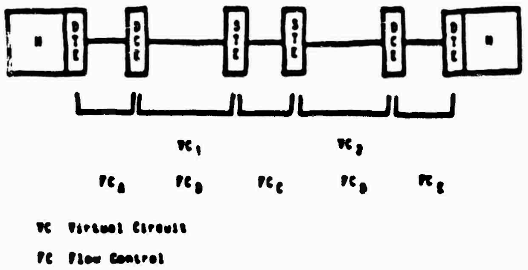


Fig. 6. PDN transmission path.

Addressing

The address field is variable in length up to 15 digits, with each digit coded in a 4 bit field. The maximum address is then 60 bits (about 8 octets).

Routing

The user has no influence over routing used. To create the series of virtual circuits, a series of call setups establishes a fixed route (between pairs of STE's at least). State information must be kept for each call in the source and destination DTE's and DCE's and in each STE in the route.

Buffering and Flow Control

Each portion of the total path is a distinct virtual circuit. Each virtual circuit has an independent flow control (and particular to that PDN). In addition, there is flow control across each STE-STE interface. All this flow control is on a per call basis. This stepwise flow control may introduce delay in the total path that could be avoided with an end-to-end scheme.

There are some concerns about the interaction of two types of flow control implemented in PDN's. One type allows one message in transit from source DCE to destination DCE at any one time. The other allows multiple messages to be in transit, the number being determined by the flow control window.

Acknowledgment

Each portion of the total path has an acknowledgment. The user to network interface also has an acknowledgment. This local acknowledgment means only that the first PDN has accepted the message for transmission, not that it has arrived at the destination.

Recovery

The X.25 and X.75 Recommendations do not specify how the PDN's deal with errors internally. If unrecoverable errors occur, the network will signal a Reset, which apparently means that the virtual circuit still exists, but the flow control is reset and messages may have been lost. More serious errors result in the call being cleared.

Because of the fixed route nature of the multinetwork path, an STE failure disrupts the communication.

Security

The X.25/X.75 Recommendations do not provide any security features.

Header Structure

Once the call is established, a header is only 3 octets. The call setup headers are substantially longer, typically 20 octets, but possibly as large as 166 octets. There is a tradeoff between header size and state information kept; in the PDN's, the tradeoff has been made toward small headers and large state. The details of the headers are shown in Appendix I.

Summary

The most important aspect of the interconnection of PDN's is that service provided to the using process is a virtual circuit with essentially the same properties a single PDN would have provided. This is done by concatenating a series of virtual circuits to provide the total path, resulting in a fixed route through a set of network interconnection points.

INTERCONNECTION OF ARPA RESEARCH NETWORKS

Introduction

The ARPA sponsored research on interconnections of networks has let to a two-level protocol to support the equivalent function of the PDN's X.25/X.75 service. The ARPA sponsored work on networks has developed an internet protocol (IP) [4], and a transmission control protocol (TCP) [5].

TCP is a logical connection transport protocol and is a level 4 protocol in the OSA model of protocol structure.

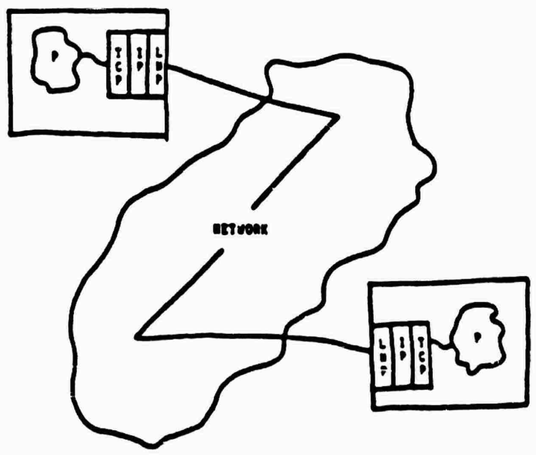


Fig. 7. End-to-end connection.

The IP is a datagram protocol. The collection of interconnected networks is called an internet. IP is the network protocol of the internet and this is a level 3 protocol in the OSA model. The actual networks used are of various kinds (e.g., the ARPANET, radio networks, satellite networks, and ring or cable networks) and are referred to as local networks even though they may span continents or oceans. The interface to a local network is a local network protocol or LNP. Fig. 7 shows the model of an end-to-end connection.

In the ARPA model, the networks interconnect via a single device called a gateway. A gateway is a host on two or more networks. Fig. 8 shows the ARPA model of the interconnection of networks.

Each network addresses a gateway on it in the same way it addresses any other host on it. The information required to deliver a message to a destination in the internet is carried in the IP header. The IP is implemented in the gateways and in hosts. A sending host prepares a datagram (which is an IP header and the original message) and then selects a gateway in its own net to forward the datagram. The sending host then sends the datagram wrapped in a local network packet to that gateway.

A gateway receives a packet from one of the local networks to which it is attached, and unwraps the IP datagram. The gateway then examines the IP header and determines the next gateway (or destination host) address in one of the local networks it is directly connected to. The gateway then sends the datagram with its IP header in a new local net packet to that gateway (or host).

The IP has no provision for flow control or error control on the data portion of the message (the IP headers are checksummed). There are no acknowledgments of IP messages. The IP is simple and the gateway may be implemented in small machines. A key point is that a gateway has no state information to record about a message. At the IP level, there are no connections or virtual circuits.

The IP does not provide a service equivalent to the PDN's X.25/X.75. To provide that type of end-to-end reliable ordered delivery of data the ARPA internet uses TCP.

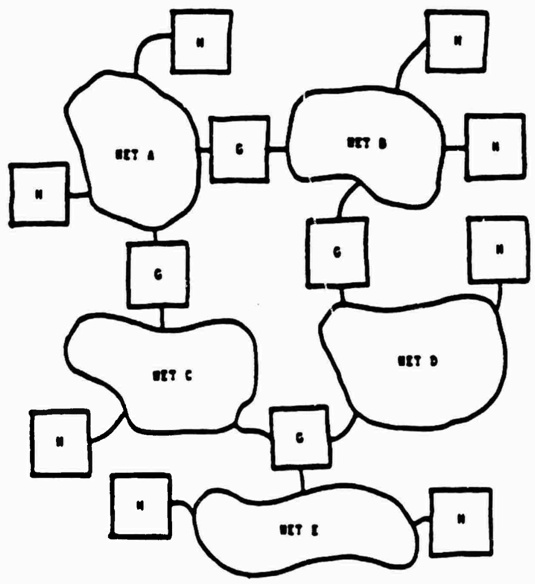


Fig. 8. ARPA model of interconnection of networks.

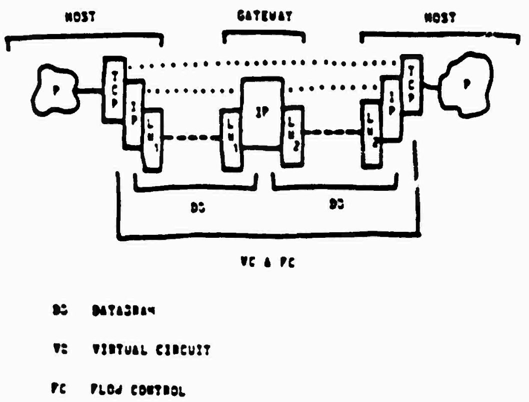


Fig. 9. ARPA model of transmission path.

TCP uses end-to-end mechanisms to ensure reliable ordered delivery of data over a logical connection. It uses flow control, positive acknowledgments with time out and retransmission, sequence numbers, etc., to achieve these goals. Fig. 9 shows the conceptual transmission path in this interprocess communication system, pointing out the datagram (DG) path between the IP modules and the virtual circuit path between the TCP modules at the source and destination and the flow control (FC) at that level.

ARPA has used these techniques to interconnect several very different networks including the ARPANET, packet radio nets, a satellite net, and several local networks.

Addressing

The size of the address in this experimental system is fixed. The IP provides a one octet network field and a three octet host field. Also a one octet protocol identifier in the IP header may be considered address information. The TCP provides a two octet port field. The total of the address length is then seven octets. Provision has been made for a host to have several addresses, so the host field is sometimes called the logical host field. The total address is the concatenation of the network, host, protocol, and port fields.

Routing

Normally, the user has no influence over the route used between the gateways. There is no call setup and the route may vary from one message to the next. No state information is kept in the gateways.

A user might insert a source routing option in the IP header to cause that particular message to be routed through specific gateways.

Buffering and Flow Control

There is no flow control mechanism in the IP. The gateways do not control the flow on connections for they are unaware of connections or any relation between one message and the next message. The gateways may protect themselves against congestion by dropping messages. When a gateway drops a message because of congestion, it may report this fact to the source of the message.

The TCP uses end-to-end flow control using windows on a per logical connection basis.

Acknowledgment

The IP has no provision for acknowledgments. The TCP uses acknowledgments for both error control and flow control. The TCP acknowledgments are not directly available to the user.

Recovery

Errors in a network or gateway result in a message being dropped, and the sender may or may not be notified. This inherent unreliability in the IP level allows it to be simple and requires the end-to-end use of a reliable protocol.

TCP provides the reliable end-to-end functions to recover from any lost messages. The TCP uses a positive acknowledgment, time out, and retransmission scheme to ensure delivery of all data. Each message is covered by an end-to-end checksum.

Because of the potential of alternate routing, the end-to-end communication may be able to continue despite the failure of a gateway.

Security

The IP provides an option to carry the security, precedence, and user group information compatible with AUTODIN II. The enforcement of these parameters is up to each network, and only AUTODIN II is prepared to do so.

The TCP end-to-end checksum covers all the address information (source and destination network, host, protocol, and port), so if the checksum test is successful the address fields have not been corrupted.

Header Structure

The IP header is 20 octets (plus options, if used), but there is no call setup and no gateway state information. Thus, at the IP level, the header size versus state information tradeoff has been made toward large header and little (no) state information.

The TCP header is 20 octets (plus option, if used). There is a connection establishment procedure called the "three-way handshake," and significant state information is kept. In this case, there are both large headers and large state tables. The details of the headers are shown in Appendix II.

Summary

The ARPA networks are interconnected by using a common datagram protocol to provide addressing (and thus routing) information and an end-to-end transport protocol to provide reliable sequenced data connections.

This model has evolved from the ARPANET experience, in particular from the internetwork protocol model suggested in a paper by Cerf and Kahn [6].

CONCLUSION

Both the PDN's and the ARPA networks are interconnected by establishing standard protocols. The PDN's provide a virtual circuit service by concatenating the virtual circuit services of the individual networks. The ARPA networks use two levels of protocol to provide both datagram and virtual circuit services.

Additional discussion of the interconnection of PDN's is provided in [7], [8]. In another paper in this issue Boggs *et al.* present in detail another example of network interconnection using the datagram approach [9].

The issues of network interconnection have been discussed for at least 5 years (for example, McKenzie [10]). The recent expositions by Sunshine [11], Cerf and Kirstein [12], and Gien and Zimmermann [13], are particularly recommended.

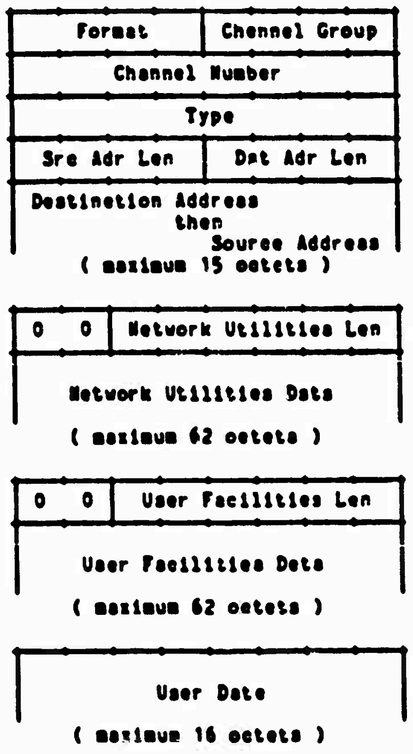
APPENDIX I

X.75 HEADER FORMATS

The call request and the data packet formats are illustrated here. These typify the X.75 packet formats. All the X.75 packets are the same in the first two octets. The format field indicated the type of packet.

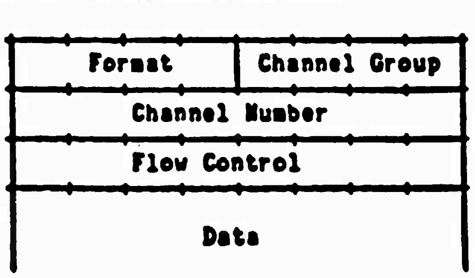
Call Request

The call request packet is variable in length from a practical minimum of 11 octets to an unlikely maximum of 160 octets.



Data

The Data packet has a three octet header.



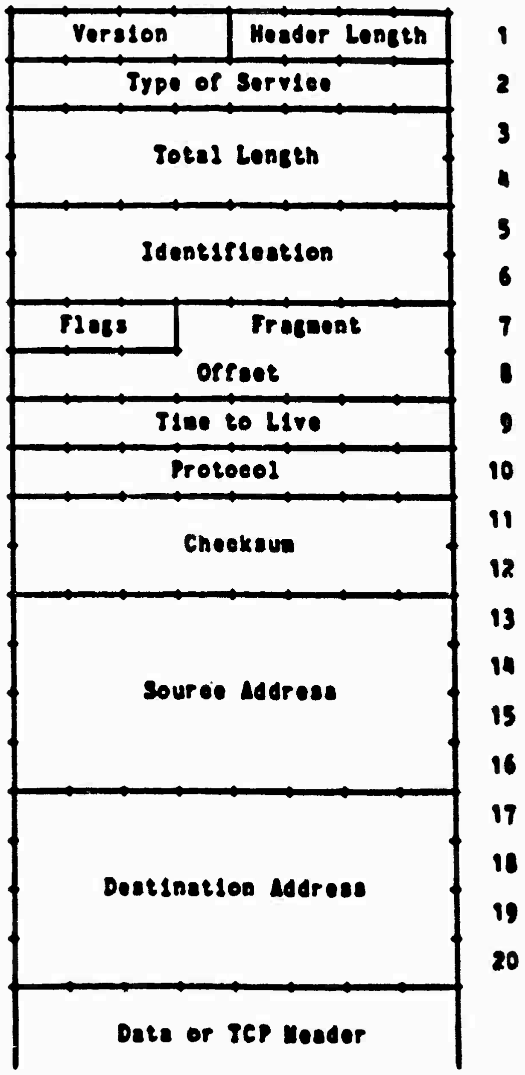
APPENDIX II

ARPA PROTOCOL HEADER FORMATS

Every datagram carries the basic IP header. Every TCP segment transmitted carries the basic TCP header.

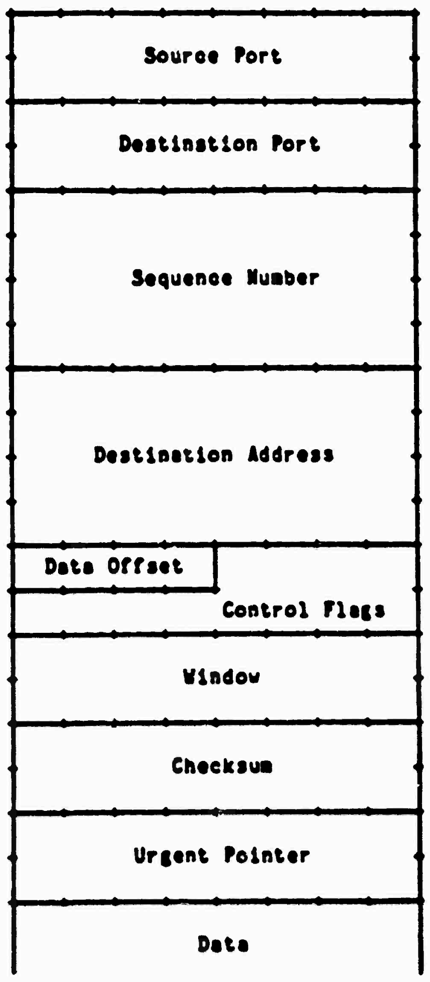
Internet Protocol

The ARPA IP has a basic header of 20 octets, and may carry a variable number of options up to a total length of 60 octets.



Transmission Control Protocol

The basic TCP header is 20 octets, and the header may be up to 60 octets long if options are used.



REFERENCES

[1] H. Zimmermann, "The ISO reference model," this issue, pp. 425–432.

[2] "Recommendation X.25/Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) for terminals operating in the packet mode on public data networks," in *CCITT Orange Book*, vol. 7, Int. Telephone and Telegraph Consultative Committee, Geneva, Switzerland.

[3] "Proposal for provisional Recommendation X.75 on international interworking between packet switched data networks," in CCITT Study Group VII Contribution no. 207, Int. Telephone and Telegraph Consultative Committee, Geneva, Switzerland, May 1978.

[4] DARPA, "DOD standard internet protocol," IEN-128, Defense Advanced Research Projects Agency, Jan. 1980.

[5] DARPA, "DOD standard transmission control protocol," IEN-129, Defense Advanced Research Projects Agency, Jan. 1980.

[6] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Trans. Commun.*, vol. COM-22, pp. 637–648, May 1974.

[7] G. Grossman, A. Hinchley, and C. Sunshine, "Issues in international public data networking," *Comput. Networks*, vol. 3, pp. 259–266, Sept. 1979.

[8] V. DiCiccio, C. Sunshine, J. Field, and E. Manning, "Alternatives for interconnection of public packet switching data networks," in *Proc. Sixth Data Commun. Symp.*, ACM/IEEE, Nov. 1979, pp. 120–125.

[9] D. Boggs, J. Shoch, E. Taft, and R. Metcalfe, "Pup: An internetwork architecture," this issue, pp. 612–624.

[10] A. McKenzie, "Some computer network interconnection issues," in *Proc. Nat. Comput. Conf.*, AFIPS, 1974, pp. 857–859.

★

Jonathan B. Postel received the B.S. and M.S. degrees in engineering and the Ph.D. degree in computer science from the University of California, Los Angeles.

He has worked for the MITRE Corporation in McLean, VA, and SRI International in Menlo Park, CA. At UCLA he was involved in the development of the ARPANET Network Measurement Center and the installation of the first host on the ARPANET. Since that time, he has participated in the development of many of the higher level protocols used in the ARPANET. He is currently a Computer Scientist at the USC/Information Sciences Institute in Marina del Rey, CA, where his research focuses on the interconnection of computer networks.

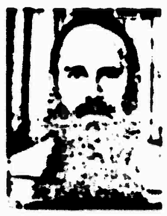
The ARPA Internet Protocol

Jonathan B. Postel, Carl A. Sunshine and Danny Cohen

Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, USA

A variety of computer networks are interconnected by gateway computers in the ARPA internetwork system. Processes on different networks may exchange messages with each other by means of an Internet Protocol which must be implemented in each subscriber (host) computer and in the gateways. The Internet Protocol is a relatively simple protocol that provides for the delivery of individual messages (datagrams) with high but not perfect reliability. This Internet Protocol does not replace the existing protocol in any network, but is used by processes to extend the range of communications. Messages in Internet Protocol are transmitted through any individual network by encapsulating them in that network's protocol. This paper presents an overview of the Internet Protocol and the operation of the gateway computers in the ARPA internet system.

Keywords: Protocol, ARPA Net, Internetwork, Datagram, Gateway.



Jonathan Postel received his B.S. and M.S. degrees in Engineering and his Ph.D. in Computer Science from the University of California, Los Angeles. Dr. Postel has worked for the MITRE Corporation in McLean, Virginia and SRI International in Menlo Park, California. He is currently a Computer Scientist at the University of Southern California Information Sciences Institute in Marina del Rey, California. At UCLA he was involved in the development of the ARPANET Network Measurement Center and the installation of the first host on the ARPANET. Since that time, he has participated in the development of many of the higher level protocols used in the ARPANET. His current research focuses on the interconnection of computer networks.

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 72C 0308, Arpa Order No. 2223. The views and conclusions in this study are the authors' and should not be interpreted as representing the official opinion or policy of ARPA, the U.S. Government or any other person or agency connected with them.

North-Holland Publishing Company
Computer Networks 5 (1981) 261–271

1. Introduction

The family of computer networks developed for the United States Defense Advanced Research Projects Agency (DARPA) represents one of the largest and most diverse internetwork systems currently in operation. The basic approach to interconnecting this variety of networks was developed over several years, and has resulted in the definition of an Internet Protocol (IP) [1]. This paper is intended primarily to document the details of the IP in the open literature, and secondarily to provide a brief discussion of the major design tradeoffs which caused the IP to take its current form.

Section 2 presents an overview of the DARPA approach to interconnection and the operation of IP. Section 3 details IP's main features, while some additional options are treated in section 4. Section 5 summarizes the IP and other functions performed in the *gateways* which interconnect networks. Section 6 discusses the major design choices in developing IP. Section 7 outlines several questions and extensions requiring further work.

Carl Sunshine is with the University of Southern California Information Sciences Institute where he is engaged in research on computer networks and their protocols. He is particularly interested in protocol modeling, and analysis, and network interconnection. Sunshine received a PhD in computer science from Stanford University in 1975, and worked at the Rand Corporation from 1975 to 1979. He is active in IFIP TC6.1 (the Internetwork Working Group) and ISO SC16, is Secretary/Treasurer of ACM SIGCOMM, and serves on the editorial board of Computer Networks journal.

Dan Cohen was born in Haifa, Israel, in 1937. He received the B.Sc. degree in mathematics from the Technion-Israel Institute of Technology in 1963, and the Ph.D. degree in computer science from Harvard University, Cambridge, MA, in 1969.

Since 1969, he has been on the faculty of Harvard University, Technion-Israel Institute of Technology, and the California Institute of Technology, Pasadena. In 1973, he joined the Information Sciences Institute of the University of Southern California, Los Angeles, here he leads several computer network-related projects. His research interests include interactive realtime systems, graphics voice communication, networking and computer architecture.

2. Overview

Since the development of the ARPANET in the early 1970's, a variety of new packet switching network technologies and operational networks have been developed under DARPA sponsorship, including satellite, packet radio, and local networks. In order to allow processes on different networks to communicate with each other, a means for interconnecting networks has been developed without requiring changes to the internal operation of any network.

The method chosen for interconnecting networks makes minimal demands on individual networks. To facilitate inclusion of a wide variety of networks, each net is required to provide only a minimal *datagram* level of service (i.e. to deliver individual packets of moderate length between its users with high but not perfect reliability). Networks are inter-connected by gateway computers that appear to be local subscribers on two or more nets. The gateways are responsible for routing traffic across multiple networks, and for forwarding messages across each net using the packet transmission protocol in each network. The gateways provide a point-to-point internet datagram service by concatenating the datagram services available on each individual net. Such a system of interconnected networks has been called a *Catenet* [2].

This approach allows the interconnection of networks that have significantly different internal protocols and performance. The networks in the ARPA-Catenet were originally designed as independent entities. In the Catenet approach no changes are required in the internal functions of any network.

Gateways provide an internet service by means of an Internet Protocol (IP) that defines the format of internet packets and the rules for performing internet protocol functions based on the control information (internet header) in these packets. IP must be implemented in host computers (subscribers) engaged in internet communication as well as in the gateways. Gateways also use a gateway-to-gateway protocol to exchange routing and control information.

IP provides for transmitting datagrams from an internet source to an internet destination, potentially in another net. IP also provides for *fragmentation* and *reassembly* of long datagrams, if necessary, for transmission through networks with small packet size limits.

IP is purposely limited in scope to provide only the function necessary to deliver datagrams over an interconnected system of networks. The functions of flow control, sequencing, additional data reliability, or other services commonly found in host-to-host protocols, and multidestination delivery capability or other services are purposely left for higher level protocols to provide as necessary. This allows the higher levels to be tailored to specific applications, and allows a simple and efficient implementation of IP.

2.1. Place in Protocol Hierarchy

As described above, IP functions on top of, or uses, the packet transmission protocol in each individual network. IP is used by higher level end-to-end protocols such as a reliable transport protocol, e.g., Transmission Control Protocol (TCP) [3] in the ARPA-Catenet or a "real time" protocol, e.g., for packet speech.

As shown in Figure 1, IP is the only level in the protocol hierarchy where a single common protocol is used. By locating this point of convergence at the internet datagram level, the Catenet approach preserves the flexibility to incorporate a variety of individual networks and protocols providing packet transmission below IP, while remaining general and efficient enough to serve as a common basis for a variety of higher level protocols. With this approach,

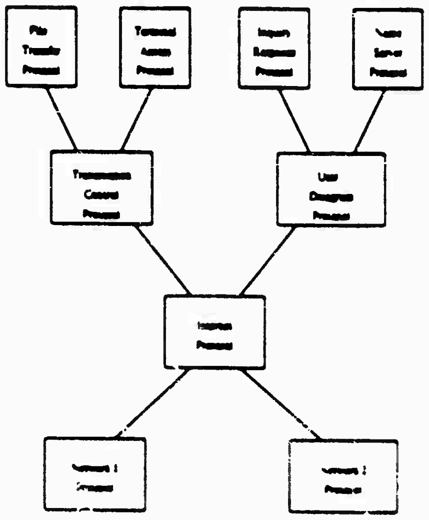


Fig. 1. Protocol Hierarchy

gateways need only provide datagram service, and remain relatively simple, inexpensive, and efficient.

2.2. Model of Operation

The Internet Protocol provides two major functions: routing a datagram across successive networks to its internet destination address, and fragmentation/reassembly of large packets when needed to cross nets with small packet size limits. To accomplish this, an IP module must reside in each host engaged in internet communication and in each gateway that interconnects networks. The following scenario describes the progress of a datagram from source to destination (assuming one intermediate gateway is involved-see Figure 2).

The basic notion is *encapsulation*. The data to be transmitted must pass through a variety of network environments. To do this the data is encapsulated in an internet datagram. to send the datagram through an individual network, it is in turn encapsulated in a local network packet, and extracted at the other side of that network where it is decapsulated from the first network protocol and is encapsulated in the second network protocol. Thus the model is a series of encapsulation/extractions, not translations. This encapsulation is an information preserving transformation, all the information is preserved even if the individual network cannot make use of it.

The sending internet user (typically a higher level protocol module such as TCP) prepares its data and calls on its local IP module to send the data as a datagram, passing the destination address and other parameters as arguments of the call.

The IP module encapsulates the data in a datagram and fills in the datagram header. The IP module examines the internet destination address. If it is on the same network as this host, it sends the datagram direc'ly to the destination. If the datagram is not on the same network then the IP module sends the datagram to a gateway for forwarding. The selection of which gateway to send the datagram to is an internet routing decision.

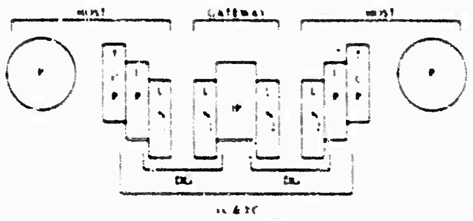


Fig. 2. ARPA Model Transmission Path.

The local network interface (note that from the IP point of view, all actual networks are "local" even if they span across the world) creates a local network packet with its own header, and encapsulates the datagram (complete with internet header) in it, then sends the result via the local network.

The datagram arrives at a gateway host encapsulated in the local network packet. The local network interface extracts the IP datagram and turns it over to the IP module.

The IP module determines from the internet destination address that the datagram should be forwarded to another host in a second network. The IP module uses the local portion of the destination address to determine the local net address for the destination host. It calls on the local network interface for the second network to send the datagram to that address.

If the datagram is too large to be sent through the second network, the IP module fragments it into several smaller datagrams and passes each one to the local net interface.

The local network interface creates a local network packet and encapsulates the datagram, sending the result to the destination host. At the destination host, the datagram is extracted from the local net packet and passed to the IP module.

The IP module determines that the datagram is for an internet user in this host. If the datagram is a fragment, the IP module collects all fragments of a particular datagram and reassembles the complete original datagram. It then passes the data to the user along with the internet source address and other information from the internet header.

2.3. Additional Mechanisms

In addition to the basic addressing and fragmentation functions described above, IP uses four key mechanisms in providing its service. *Type of Service*, *Time to Live*, *Options*, and *Header Checksum*. Each of these is summarized here and fully described in Sections 2 and 3.

The Type of Service (TOS) is used to indicate the quality of the service desired – this may be thought of as selecting among Interactive, Bulk, or Real Time, for example. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the Catenet. This type of service information is used by gateways to select the actual parameters for transmission through each individual network.

The Time to Live (TTL) is an indication of the lifetime of a datagram. Datagrams must not be allowed to persist in the ARPA-Catenet indefinitely. This is because reliable end-to-end protocols depend on there being an upper bound on datagram lifetime, especially old duplicates due to retransmissions. The time to live can be thought of as a self-destruct time limit.

The Options provide for control functions useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, error reports, and special routing.

The Header Checksum provides a verification that the information used in processing the datagram has been transmitted correctly. However, the data is not covered by the checksum, and may contain errors (see Section 2.6). If the header checksum fails, the internet datagram is discarded by the entity which detects the error.

2.4. Relation to Other Work

The current ARPA Internet Protocol evolved from ideas suggested by Cerf and Kahn [4], and from contemporaneous proposals within the International Federation for Information Processing (IFIP) Technical Committee 6.1 (also known as the International Network Working Group or INWG), in which internet functions and reliable transport functions were combined in a single protocol. Subsequent development of other high level protocols (such as packet speech) that needed internet services led to splitting internet functions and reliable transport functions into separate protocols (the current IP and TCP).

The Internet Protocol used in the ARPA-Catenet is quite similar in philosophy to the PUP protocol [5] developed by the Xerox Corporation. The PUP protocol does not include fragmentation (leaving this to each local net to perform if necessary), but does include a third level of addressing (Ports within hosts) in the internet packet header. IP and PUP share the important principle of having a single common internet datagram protocol as a point of convergence in their protocol hierarchies. Both the PUP and IP systems use the encapsulation technique, and a scheme for "mutual encapsulation" has been worked out [6]. PUP and IP both trace their roots to a joint XEROX-DARPA project at Stanford University. The network interconnection approach used by the European Informatics Network [7] is also quite similar.

Public packet switching networks, on the other hand, have chosen to use virtual circuit (VC) level of service as the level of interconnection, providing end-to-end service as a concatenation of VCs through each network. Since gateways must participate at the VC level, they are more complex and costly, and the end-to-end service may be less efficient and less robust. They are also unable to accommodate "transaction" type users without setting up a VC, although the CCITT is currently considering adding a datagram type of service. For further comparison of CCITT and Catenet approaches see [8–12].

In summary, the ARPA Internet Protocol supports delivery of datagrams from an internet source to a single internet destination. IP treats each datagram as an independent entity unrelated to any other datagram. There are not connections or logical circuits (virtual or otherwise). There are no acknowledgements either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is minimal flow control. For flexibility, it is explicitly left to higher level protocols to provide these functions.

3. Main Features

The following paragraphs describe in some detail the mechanisms of the IP. A summary of the contents of the IP header is shown in Figure 3. Further information may be found in the current specification [1].

3.1. Addressing

The IP provides a two level addressing hierarchy. The upper level of the hierarchy is the *network number* (8 bits), and the lower level is an address within that network (24 bits), and is commonly called the *host*. This second level of the hierarchical address is sometimes called the *local address*. The details of the local address are dependent on the particular network.

The local address should allow a single physical host to act as several logically distinct internet hosts. That is, there should be mapping between internet

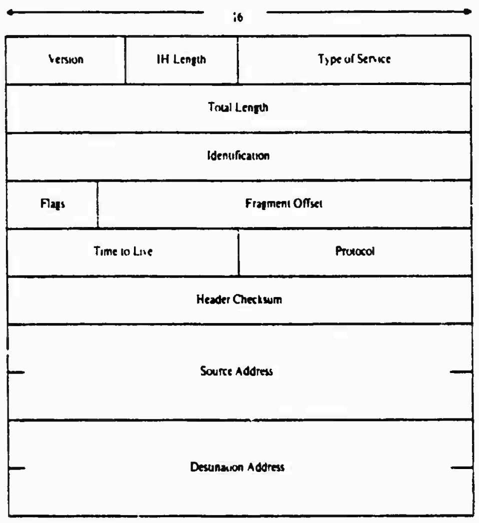


Fig. 3. INTERNET Protocol Header.

host addresses and network/host interfaces that allows several internet addresses to correspond to one physical interface. It should also be possible for several interfaces to accept or emit datagrams for the same internet address.

3.2. Protocol Number

The *Protocol Number* indicates the next level protocol used in the data portion of the datagram. This allows the internet module to demultiplex the incoming datagrams to higher level protocol modules for further processing. Hence, the protocol number indicates the format for parsing the rest of the datagram. Note that there is only one protocol number rather than a source protocol and a destination protocol because, higher level protocol modules exchange datagrams with each other using the same protocol. For example, two TCP modules exchange TCP segments via datagrams marked "TCP" in the protocol number.

One particular protocol number designates a multiplexing protocol which allows several independent data blocks from possibly different higher level protocol modules to be aggregrated together into one datagram for transmission [13].

3.3. Fragmentation and Reassembly

The IP provides information to allow datagrams to be fragmented for passage through networks with small packet size limits and to be reassembled at the destination. The necessary information includes an identification of the fragments that belong to the same datagram and the position of each fragment within the datagram.

The *Identification* (ID) field is used together with the source and destination address, and the protocol number, to identify datagram fragments to be assembled together. The *More Fragments* flag (MF) is set if the datagram is not the last fragment. The *Fragment Offset* (FO) identifies the fragment location, relative to the beginning of the original unfragmented datagram. These offsets are counted in units of 8 octets. Hence, if a datagram is fragmented, its data portion must be broken on 8 octet boundaries. This convention is designed so than an unfragmented datagram has all zero fragmentation information (MF = 0, FO = 0).

If the *Don't Fragment* flag (DF) is set, then internet fragmentation of this datagram is not permitted, although this may force it to be discarded at a gateway to a small packet network. DF can be used to prohibit fragmentation in cases where the receiving host does not wish to reassemble internet fragments. It is also possible that a small packet network could use network specific fragmentation and reassembly without the knowledge or involvement of the IP modules [14].

If a datagram is too large to be forwarded through any net, the entrance gateway breaks it into as many fragments as are necessary to fit within that net's packet size limit. Figure 4 shows a large datagram of 452 octets being fragmented into two smaller fragments (only the header fields relevant to fragmentation are given). Subsequent gateways may break the fragments into even smaller fragments if necessary using the same procedure.

Datagrams arriving at the destination IP are easily recognizable as fragments if either MF or FO is nonzero. Fragments from the same original datagram are identified by having identical ID fields (for a particular source, destination, and protocol number). Fragments are queued until the original datagram can be fully reassembled. Reassembly may be accomplished by placing the data from each fragment in a buffer at the position indicated by FO. Using the header information from the first fragment, the reas-

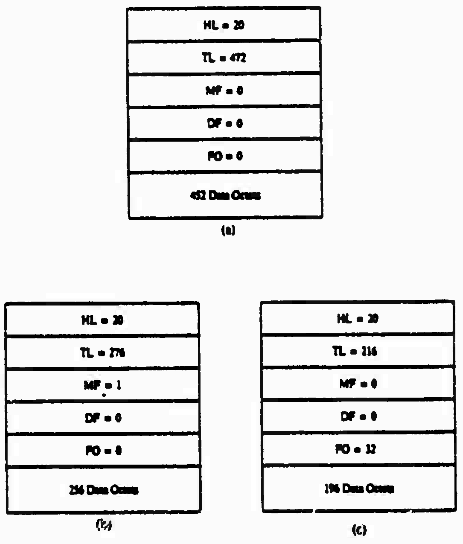


Fig. 4. Fragmentation Example.

sembled datagram is processed further just as if it had been received intact. If the time to live on any fragment expires during reassembly, the partially assembled datagram is discarded, and an error datagram is sent to the source.

A convention has been established in the current ARPA-Catenet that no datagrams larger that 576 octets will be sent, and that all receivers will be prepared to receive a reassemble datagrams up to this length (unless specifically arranged otherwise). This number is chosen to allow a data block of 512 octets and a reasonable number of header octets for several protocol levels to be transmitted in one datagram. Note that the IP header is repeated in each fragment. Hence, the minimum maximum packet size for any network in the Catenet is 20 header octets plus 8 data octets or 28 octets total.

The internet fragmentation procedure allows the fragments to be treated as independent datagrams the rest of the way to their destination (even taking different routes), with reassembly occurring only at the destination.

There is a need to uniquely identify the fragments of a particular datagram. Hence the sender must choose the identification field to be unique for each source/destination pair and protocol number for the time the datagram (or any fragment of it) could exist in the internet. Since the ID field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is beneficial for some higher level protocols to choose the identification field. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment. Note that a retransmission might be routed via a different set of networks and gateways and also may be fragmented into a different number of different sized fragments. The fragmentation information permits reassembly from fragments from either copy of the datagram.

3.4. Type of Service

The Type of Service (TOS) provides a network independent indication of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Some networks offer several precedence levels of service. Another choice involves a low-delay vs. high-reliability trade off. Typically networks invoke more complex (and delay producing) mechanisms as the need for reliability increases. A few networks offer a stream service, whereby one can achieve a "smoother" service at some cost. Typically this involves the reservation of resources within the network.

The abstract service quality parameters provided by IP are:

Precedence: Indicates the importance of this datagram.

Stream or Datagram: Indicates if there will be other datagrams from this source to this destination at regular frequent intervals justifying the maintenance of stream processing information.

Reliability: A measure of the level of effort desired to ensure delivery of this datagram.

Speed: A measure of the importance of prompt delivery of this datagram.

Speed over Reliability: Indicates the relative importance of speed and reliability when a conflict arises in achieving both.

3.5. Time to Live

The Time to Live (TTL) indicates the maximum time the datagram is allowed to exist in the Catenet.

As a datagram moves through the Catenet the TTL is decremented. If the TTL reaches zero the datagram should be discarded. The intention is to cause long delayed or undeliverable datagrams to be discarded. Guaranteeing a maximum lifetime for datagrams is important for the correct functioning of some higher level protocols such as TCP, and to protect the Catenet resources.

This field should be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no information is available on the time actually spent, the field should be decremented by 1. The time is measured in units of seconds, and the maximum TTL is 255 seconds.

3.6. *Checksum*

The IP provides a checksum on the header only. Since some header fields may change (e.g., TTL, MF, FO), this is recomputed and verified at each point that the internet header is processed. This is a hop-by-hop checksum.

This checksum at the internet level is intended to protect the internet header fields from transmission errors. If the internet header contained undetected errors, misrouting and other unanticipated behavior could result. There may be applications in which it is desirable to receive data even though there are a few bit errors. If the IP enforced a data checksum and discarded datagrams with data checksum failures such applications would be restricted unnecessarily.

The checksum is computed as the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero. This checksum is simple to compute and has been adequately reliable for usage to date, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

3.7. *Header Format*

In addition to the main features discussed above, the IP includes the following items in the datagram header:

A *Version Number* (VER) which indicates the version of the IP in use, and hence the format of the internet header.

The *Internet Header Length* (IHL) is the length of the internet header and thus points to the beginning of the data.

The *Total Length* (TL) is the length of the datagram, including internet header and data. There are several protocol options, some of which are discussed in the next section.

4. Additional Features

The following optional mechanisms are available in the IP for use when needed.

4.1. *Source Routing*

The *Source Route* option provides a means for the source of a datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination.

As described above, routing at each gateway is based on the internet address in the destination field of the datagram header. If the source routing option is used, a series of additional internet addresses will be present in the option field. When the address in the destination field has been reached and the source route is not empty, the next address from the source route becomes the new destination (and is deleted from the source route list).

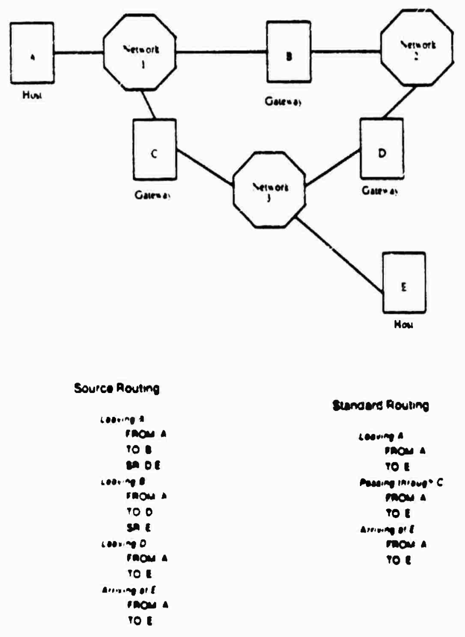


Fig. 5. Source Routing Example.

Thus, the source specifies a series of points the datagram must pass through on the way to its final destination. Normal internet routing is used to reach each of these points in turn, and the datagram may pass through a number of intermediate points between the specified addresses. Source routing may be used to specify routes to networks that are not known to the full internet system.

In Figure 5 an example of source routing is shown. Here host A is sending a datagram to host E. The normal routing would most likely be through the gateway C. We assume the user at host A would prefer in this case to have this datagram routed through gateways B and D. The Figure shows the address information at each step along the route.

4.2. Return (or Record) Route

The *Return Route* option provides a means to record the route taken by a datagram. A return route is composed of a series of internet addresses. When an IP module routes a datagram and the return route option is present, the gateway inserts its own internet address (in the environment of the next destination) into the return route option data.

4.3. Error Report

The *Error Report* option is used to report an error detected in processing a datagram to the source. A code indicates the type of error detected, and the ID is copied from the datagram in error, and additional octets of error information may be present depending on the error code. If a datagram consisting only of an error report option is found to be in error or must be discarded, no error report is sent.

Error codes are defined to report the following conditions: (0) No reason given, (1) Not Accepted – no program at the destination will accept the datagram, (2) Fragmentation Problem – the datagram cannot be delivered without fragmenting and the DF flag is set, (3) Reassembly Problem – the datagram cannot be reassembled because there are missing fragments and the time to live has expired, and (4) Gateway Congestion – the datagram was discarded to relieve congestion.

5. Gateway Functions

This section summarizes the tasks performed by a gateway: which are, interfacing to the local networks,

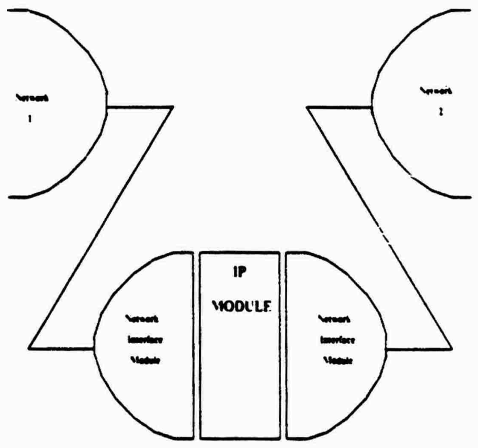


Fig. 6. Gateway.

and performing the IP functions.

The actual interconnection of networks is performed by gateways which are computers connected as hosts on several networks (see Figure 6). Messages are communicated across networks by using the protocols and conventions of the individual networks. While traversing each network the IP datagram is encapsulated within the local network protocols. At the gateway the IP datagram is decapsulated and examined by the gateway to determine how to route this datagram, and what local network options to use, if any. The gateway handles issues of routing, fragmentation (if the local network cannot handle regular size datagrams), error reporting and control, and interfacing to local networks.

The essential purpose of a gateway is to forward each datagram toward its destination. The key decision a gateway must make is the routing decision. When a gateway receives a datagram it must use the destination address in the IP header along with routing information stored in the gateway to determine where to send the datagram.

The routing information stored in the gateway may be relatively static (changed only by manual intervention) or dynamic (changed automatically). Both cases are allowed in the ARPA-Catenet system. The discussion of the techniques for dynamically updating the routing information are described by Strazisar [15].

Another important task of a gateway is to encapsulate datagrams for transmission through the next network, using that network's existing message transfer protocol. This involves adding an appropriate message header (and perhaps trailer), to the datagram. The gateway must interpret the type of service field of the IP header to select the appropriate service in the next network.

The gateway decreases the TTL to account for the time elapsed since the TTL was last adjusted. This is an estimate of the time spent in transmission and processing. If this reduces the TTL to zero the gateway discards the datagram.

If the datagram is larger than the maximum packet size of the next network, the gateway may fragment it into pieces that will be sent separately.

If the gateway must discard a datagram due to congestion or errors in processing the datagram (such as an unknown or currently unreachable address), it sends an error report datagram to the source of the discarded datagram.

Of course, the gateway verifies the IP header checksum on every datagram it receives before processing it. If the check fails the datagram is discarded with no notification to the source or adjacent gateway. Since some of the IP header information is changed during gateway processing (e.g. TTL), the gateway computes a new IP header checksum before sending it on.

Each datagram can be processed completely independently of other datagrams. The provision of error recovery, sequencing, or flow control functions are left for end-to-end protocols, and the gateway does not maintain any status information or dedicate any resources for individual virtual circuits. Indeed, the gateway is unaware of any details of the higher protocol levels.

6. Design Decisions

The key decision in the design of the ARPA Internet Protocol is the choice of a datagram basis rather than a virtual circuit basis. Using datagrams as the basis of communication in the Catenet permits the use of simpler gateways since they are not required to maintain state information about the individual virtual circuits, and allows the end-to-end communication to continue via alternate routing if a gateway fails.

Using datagrams as the basic communication service allows the construction of virtual circuit style end-to-end services (e.g., TCP), and other services. In the DARPA research program there are needs for other styles of communication service. For example, the packet speech requires a service which provides minimal delay even at the cost of a few dropped messages. Such a service can be built on a datagram base, but not on a virtual circuit base. For more detail on the tradeoff between a datagram base and a virtual circuit base for communications see references [8–12].

This choice of a datagram base for the operation of the Catenet results in the separation of the internet protocol from the end-to-end protocols in general and TCP in particular. The early proposals for TCP did not focus clearly on the responsibilities of the gateways and did not allow for alternate styles of communication service. Once these needs were apparent the protocol functions were separated into distinct layers.

The decision to use the encapsulation/decapsulation technique to send the IP datagrams through local nets was made to maximize individual networks' autonomy, and to avoid the need for modifications of individual networks (particularly in the area of routing) to support internet traffic [10].

The decision to fragment datagrams in gateways as they pass from a large packet network into a small packet network, but not reassemble the fragments until they reach the destination host, allows simpler gateways and minimizes the delay in the Catenet. The alternate approach of reassembly in the next gateway is explored in reference [14].

Perhaps the most difficult design decision was the choice of the address size and structure. The size of the address field is a compromise that allows enough addresses for the anticipated growth of the Catenet yet is not an excessive overhead burden. The structuring of the address into network and host fields allows the gateways to process datagrams destined for distant networks on the basis of just the network field. This field separation also reflects an administrative delegation of the address assignment function.

In addition to the address, IP carries additional address or multiplexing information in the protocol field. This indicates which next level protocol should be used to interpret this datagram. Most of the higher level protocols have further multiplexing information called *ports* in their headers. The IP approach to addressing may be characterized as hierarchical [10].

An option in IP supports the concept of source routing. This means a source may specify a series of addresses which are used in turn until the ultimate destination is reached [10]. The decision to include this feature was motivated by the realization that many small networks may be interconnected to the Catenet via ad hoc arrangements, and destinations in such networks (or such networks themselves) may be unknown to gateways in the general Catenet.

IP uses a Time To Live which is decremented by each gateway by at least one unit (more if the datagram is delayed in the gateway for a substantial time). Other protocols use a hop count which is incremented by each gateway [5]. The practical difference is small, though the time to live approach remains effective as the size of the network changes, and allows the source to specify a maximum life for the datagram.

7. Research Issues

7.1. Multiple Addresses

There are several issues related to more flexible addressing that the current IP does not deal with. One case is a host with two (or more) internet addresses, either on one network or even on different networks. Sometimes this serves to distinguish between logically separate hosts, but in other cases it is desirable to consider both addresses as the "same place" as far as higher level protocols are concerned. It is not clear how a gateway could know when or how to route messages sent to one address to another address (e.g. if the first address was unreachable). A particularly difficult example of this problem is a mobile packet radio which moves from one network to another while trying to maintain unbroken communication.

7.2. Local Networks

A second issue is the addressing of local networks. There will soon be a large number of local networks (e.g., networks within one building or on a campus) wishing to use the ARPA-Catenet for long distance interconnection. It seems unreasonable that every one of these should have the same status as a nationwide network, with all gateways responsible for maintaining routing information about them. It may be preferable to introduce another level in the addressing hierarchy, or to combine a gateway plus internal address for such nets in the local address field of IP addresses [16].

7.3. Multiple Destinations

Another addressing issue is provision of a capability to send datagrams to a number of destinations at once. Broadcast to all is, of course, the ultimate multi-destination, but "to all" is easier to handle then "to some." This capability is inherent in the technology of some networks (e.g. satellite, ring, and Ethernets) but there is no provision in the current IP for such multidestination addressing. These is work underway in the ARPA community on an internetwork digital packet speech conferencing experiment. A protocol called ST developed for that experiment does contain a multidestination capability [17].

7.4. Naming/Addressing/Routing

The mapping of character string names that are convenient for people into internet addresses is often a problem. This can be eased by the provision of a "directory assistance" service or name server [18]. A name server is a service with a table of name/address correspondences. When the name server is sent a query about a name it responds with the name and corresponding address(es). Directory services can be provided in a centralized and/or distributed fashion. For a further discussion of the roles of names, addresses, and routes see [19].

7.5. Congestion Control

Congestion control is a problem for any network. The gateways may be viewed as nodes of the Catenet, much as IMPs are the nodes of the ARPANET. As internet traffic increases, gateways may become overloaded, even while the individual networks connecting them are enforcing their own congestion controls. Thus there may be a need for an internet congestion control mechanism which is effective with the datagram mode of operation in the Catenet. Several methods such as isarithmic control, buffer categories, and "choke" packets [20] have been proposed for such environments. The ARPA gateways implement a simple strategy of notifying the source when a packet must be discarded due to congestion.

7.6. Monitoring and Administrative Control

Accounting is another basic internetworking requirement. Traffic statistics are useful for monitoring and control purposes, and are easily collected by the gateways either on a net-to-net basis, or with more detail by internet source/destination pairs. Volume of packets and/or bits can be collected by a set of counters, and periodically dumped to a Catenet monitoring and accounting center. A gateway monitoring and control center is now operating to coordinate the collection of these statistics [21].

8. Conclusions

The ARPA Internet Protocol provides a common base for supporting higher level protocols in a network independent multi-network environment. The datagram basis of the internet protocol has allowed the flexible evolution of a variety of application specific higher level protocols while allowing simple gateways to interconnect networks. The principle of encapsulation for transmission through individual networks is essential for the provision of internet service over a variety of networks without requiring changes to each networks' internal operation.

As of August 1980, IP is implemented in 12 gateways interconnecting 10 networks, including packet radio, satellite, local nets, and the original ARPANET. Gateways are typically PDP 11/40 or 11/03 processors with limited memory. High level protocols including TCP, terminal access (Telnet), and file transfer (FTP) are in use above IP. Transaction oriented services such as directory assistance (Name Server) are also in use. Other applications are under development.

Acknowledgments

IP has developed in the context of a multicontractor research effort. It is not possible to list all the contributors that have participated in the DARPA Internet Program, however special mention should be made of the coordinating effort and technical contributions of Dr. Vinton G. Cerf of DARPA.

References

[1] Postel, J., "DOD Standard Internet Protocol," IEN 128, RFC 760, USC/Information Sciences Institute, NTIS document number ADA079730, January 1980.

[2] Cerf, V., "The Catenet Model for Internetworking," IEN 48, Defense Advanced Research Projects Agency, July 1978.

[3] Postel, J., "DOD Standard Transmission Control Protocol," IEN 129, RFC 761, USC/Information Sciences Institute, NTIS document number ADA082609, January 1980.

[4] Cerf, V. and R. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Transactions on Communications, vol. COM-22, no. 5, May 1974.

[5] Boggs, D.R., et al., 'Pup: An Internetwork Architecture," IEEE Transactions on Communications, vol. COM-28, no. 4, April 1980.

[6] Shoch, J., D. Cohen, and E. Taft, "Mutual Encapsulation of Internet Protocols," Trends and Applications 1980: Computer Network Protocols, National Bureau of Standards, Gaithersbug, Maryland, May 1980.

[7] Deparis, M., et. al., "The Implementation of an End-to-End Protocol by EIN Centers: A Survey and Comparison," Proceedings International Conference on Computer Communication, Toronto, Canada, August 1976.

[8] Grossman, G.R., A. Hinchley, and C.A. Sunshine, "Issues in International Public Data Networking," Computer Networks, vol. 3, no. 4, August 1979.

[9] DiCiccio, V., et. al., "Alternatives for Interconnection of Public Packet Switching Networks," Proceedings. Sixth Data Communication Symposium, ACM/IEEE, November 1979.

[10] Sunshine, C., "Interconnection of Computer Networks," Computer Networks vol. 1, no. 3, pp. 175–195, January 1977.

[11] Cerf, V. and P. Kirstein, "Issues in Packet-Network Interconnection," Proceedings of the IEEE, vol. 66, no. 11, November 1978.

[12] Postel, J., "Internetwork Protocol Approaches," IEEE Transactions on Communications, vol. COM-28, no. 4, April 1980.

[13] Cohen, D. and J. Postel, "On Protocol Multiplexing," Sixth Data Communication Symposium, ACM/IEEE, November 1979.

[14] Shoch, J., "Packet Fragmentation in Internetwork Protocols," Computer Networks, vol. 3, no. 1, February 1979.

[15] Strazisar, V., "How to Build a Gateway," IEN 109, Bolt, Beranek, and Newman, August 1979.

[16] Cohen, D., "On Addressing and Related Issues (Or Fuel for a Discussion)," IEN 122, USC/Information Sciences Institute, October 1979.

[17] Forgie, J., "ST – A Proposed Internet Stream Protocol," IEN 119, M.I.T. Lincoln Laboratory, September 1979.

[18] Pickens, J., E. Feinler, and J. Mathis, "The NIC Name Server – A Datagram Based Information Utility," Proceedings of the Fourth Berkeley Conference on Distributed Data Management and Computer Networks, August 1979.

[19] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON Fall 78 Proceedings, (IEEE Catalog Number 78CH-1388-8C), Washington, D.C., September 1978.

[20] Grange, J., and M. Gien, eds., Flow Control in Computer Networks, North Holland, February 1979.

[21] Flood Page, D., "ARPA Catenet Monitoring and Control," IEN 105, Bolt, Beranek, and Newman, May 1979.

"INTERNETWORKING IN THE MILITARY ENVIRONMENT"

by B.H. Davies and A.S. Bates

Royal Signals and Radar Establishment
Gt. Malvern, Worcs, U.K.

Abstract

The increasing requirement for data communications in the military environment and the heterogeneous nature of the network technologies and protocols involved are highlighted. The main section of the paper discusses how the design of a military internet architecture is influenced by the military requirements especially that of survivability. Comparison with the civilian PTT approach to internetworking shows that while there are economic advantages to using civilian international standards where possible, these standards do not satisfy the military requirements. In particular the strategies for routing in a heavily damaged network environment and addressing hosts that migrate from one network to another must form an integral part of the overall architectural design. This results in gateways whose routing tables have a finer degree of detail of the internet topology than is usually required but which do not contain connection oriented information.

Finally, practical experience gained on the ARPA catenet system is described.

I. Introduction

The increasing complexity and tempo of modern warfare has rapidly created the need for flexible data communications, parallel to those associated with the "information technology" growth in the civilian environment. The aim of this paper is to highlight the differences in emphasis between data communications in the civilian and military environments, and to examine the consequence of these differences. In particular, the importance of an overall communications architecture, in order to provide survivable and interoperable communications involving both present and future systems, cannot be overstated.

Experience gained in connecting a prototype military network to the ARPA catenet system and measurements made using internetworking data transport protocols are described. Enhancements to the system to improve survivability and performance are suggested.

II. The Requirement

To a large extent, the increase in the demand for data communications stems from the increasing use of computers, microprocessors and digital circuitry in weapons, sensor, and command and control systems. These devices are used for similar reasons to those pertaining in the civilian environment, in that they can perform well specified tasks faster, more reliably and more cheaply than human personnel. However, in order to accomplish the overall goal of efficient deployment of military resources, these geographically separated devices must communicate with each other and exchange information in a hostile environment. A distinctive property of the communications between these devices, is the very "bursty" or non-continuous nature of the information transfers, which makes packet switching an attractive means of providing the communications. In packet switching, bandwidth is only allocated on demand, and therefore this technique allows considerably more efficient sharing of communication resources than the use of dedicated communication links. A further advantage of a well designed network, is the inherent survivability of communications that it provides. This does not mean that networks in a damaged condition provide the same quality of service as in their pristine condition, hence the necessity for priority markings to indicate which data is the most important. However, we can say that packet switching is an economical means of distributing the communications resources in such a manner that it is difficult for the enemy to completely destroy communications between users of the network.

So far we have described a single set of users connected to one network. However, there are many different types of networks based on different technologies and providing different types of service. This diversity of network types is due to the different user requirements and environments. For example, naval data communications may well be provided by a packet satellite network because of the large geographical area of coverage required and the great mobility of the hosts or users of the network. In the forward area tactical environment, the data communications may well be provided by a frequency hopping packet radio network, because of the extreme hostility of the electromagnetic environment. Finally, in an underground control centre, or on board a single ship, the communications may be provided by a "local area network".

Besides these different hardware technologies the grade of service provided to the user may differ. For example, a network which is primarily designed for transporting sensor information, may well be optimized for providing minimum delay in the delivery of the data, rather than providing reliability of delivery, because of the perishable nature of the data. Thus, users who are primarily interested in reliable delivery would have to initiate transport control features on an end-to-end basis, to provide for loss and misordering of the data by the network.

There is a requirement for users on the different networks to communicate with each other [1]. In particular, the long haul communications may be provided by a common bearer network, which may interconnect forward area networks with local command centre networks. Also, with additional tasks and new capabilities, there will continue to be new and unknown data communications requirements, which will have to be integrated with existing systems.

The main requirements of data communications are that they should be secure, survivable and interoperable [2]. This paper concentrates on the survivability and interoperability issues, and the reader is referred to the references which concern computer and network security [3,4]. However, it is necessary to point out that the more interoperable the systems are, the greater the security risks, because there are more avenues of attack on the confidentiality and integrity of the data, by a greater number of personnel. In particular, "access controllers" or security sentinels in critical gateways, which interconnect networks, may restrict access to certain types of traffic, thus sacrificing survivability and flexibility in the interests of security. Survivability of communications has many different meanings, but in its strictest sense it implies fully automatic routing around damaged switching components or links, and the ability to use alternate routes, even through other networks, in such a way that data integrity is maintained on an end-to-end basis.

III. Reasons for an Overall Architecture

To date, most communications systems have not been designed with an overall communications architecture in mind. This has resulted in great difficulty in providing interoperability with other systems. Because the modulation and coding, addressing and message representation, have often been combined, interconnection with another system has involved a very expensive box between the two systems. The disadvantages of this approach are:-

1) Each interface box is a special 'one off' design, which is custom built and therefore very expensive in design time and procurement cost.

2) Inevitably, in translating between one system and another, there will be certain features and services that will not have an equivalent in both systems.

3) Because of the processing power required to translate at all protocol levels, the interface unit will be a large and expensive piece of hardware. This has an effect on survivability, in that because the interface units are expensive, the minimum will be procured and the survivability of the overall communications will be determined by these vulnerable interface units.

The problem of deciding on the best architecture for computer to computer communications, has been the subject of sustained discussion over the past decade. In particular, the International Standards Organization's subcommittee 16 has produced a major document in this field, "Reference Model of Open Systems Interconnection" [5]. The central thesis of this document is that the most flexible architecture is a layered one, in which each layer has a well specified function and provides a well specified service to the layer above it. In particular, any given layer views the layers below it as a single entity. This is analogous to structured programming, where the user of a procedure call is only interested in how parameters are passed to and from the procedure and not in the internal structure of the procedure. The seven layer model is illustrated in figure 1. Two points about the model are relevant to the discussion below. Firstly, the functional specification of each layer is more difficult to agree on, the higher the layer, because in these layers in the architecture there are more choices. Secondly, there has as yet been no ISO agreed protocols for implementing any of the layers. The model itself does not preclude more than one protocol implementing a given layer of the architecture.

IV. Current State of Civilian Standards

In Europe, with its highly regulated public communications authorities, there has been a very active co-operation among various countries to establish data communications standards from the outset. The CCITT (The International Telegraph and Telephone Consulative Committee), which is the corporate body representing the telecommunications authorities of these countries, has developed standard protocols, X25 [6], for levels 1,2 & 3 of the ISO reference model. It is important to note that in arriving at these standards, the PTTs (Public Telegraph and Telephone authorities) have identified that most customers want a connection orientated type of service, ensuring ordered and reliable delivery of packets. The network reserves the right, in event of a network error or congestion, to send a reset to both ends, indicating loss of data integrity. At present, no figures are available to indicate the frequency of such events. Because the main public networks in Europe are X25 networks, there has been considerable pressure on computer manufacturers to provide X25 hardware and software products off the shelf. This has led manufacturers of private networks, in particular local area networks, to consider providing X25 accesses, in order to facilitate connections to existing machines and operating systems. Thus, X25 is rapidly becoming a de facto international standard in Europe.

What about the interconnection of X25 networks? Obviously, connecting networks which use the same access protocols and provide the same grade of service, is not so difficult a problem as interconnecting very dissimilar networks. Thus, there are X series protocols, X75, X121 [6], which enable PTT's to provide connections between users on different X25 networks, and although not all X25 facilities are available on internetwork connections, the service offered is analagous to STD dialling of international telephone calls. However, these protocols do rely on the X25 networks themselves, to route the internet packets to the gateways. It appears that private networks will not be allowed to connect to public networks via X75 gateways, and so gateways between private and public networks will have to provide a service between two X25 calls back-to-back, and will thus act as a staging post for the user's data.

Protocols for the transport layer (layer 4 of the OSI Reference model), are not so well developed as for the lower layers. However, in the United Kingdom a transport protocol [7] has been defined, and implementations above X25 have been realized. The most notable feature of this protocol is the flexible addressing structure, which allows connections to be established across different naming/addressing domains.

Before considering the applicability of these developments in the military environment, it is useful to consider some of the differences in emphasis, between civilian and military networks, and their usage.

V. Comparison Between Civilian and Military Networks and their Usage

1) The usage of military networks in time of war is very difficult to predict. Although major exercises give some idea of the user demand, past experience has shown that these are slightly artificial and may not give a true picture. In civilian networks, usage can generally be accurately predicted by extrapolating present usage patterns, with economic and equipment sales factors being taken into account.

2) The availability of the full capacity of a military network may well be degraded when it is most needed, because links may be jammed and nodes and gateways physically destroyed. In the civilian environment, there is usually a very high availability of hardware and data links, with the use of standby power supplies and 'hot' spares for critical nodes such as gateways.

3) In general, there is a considerably higher degree of mobility of both users and networks in the military environment. In particular, airborne networks such as JTIDS (Joint Tactical Information Distribution System), with users such as fighter aircraft, will place stringent requirements on internetwork connections and survivability. A consequence of this will be that the users may well be completely unaware of the internet topology. While mobile access to networks will obviously develop in the civilian environment, in general it constitutes a fairly static community of networks and users.

4) One of the major advantages of geographically distributed databases, which are flexibly interconnected with communications links, is the decrease in vulnerability of the overall system to the total failure of a site (eg by physical destruction). Thus, when designing military networks it is important not to introduce an Achilles heel by, for example, employing a centralized network control centre. However, centralized control may well be the most convenient and cost-effective solution in civilian environment.

5) Both civilian and military network authorities wish to provide secure, survivable, interoperable, and guaranteed grades of service to their users. The questions arise as to how much the user is willing to pay for these properties, and how important the properties are? The question of the importance of the property, depends on the threats to the network, and these are obviously substantially greater in the military case. This means that the solutions for military networks may well be more expensive, in terms of implementation and running costs, than those for the civilian environment.

VI. Techniques For Network Interconnection

At present there are two main architectural methods [8] for providing process to process communication across dissimilar networks. They are referred to as the "end-to-end" and "hop-by-hop" methods, because in the former, all the control information relevent to a particular data connection is held only in the source and destination hosts, while in the latter, connection oriented information is also held in various intermediate switching nodes, called gateways.

The end-to-end approach is based on the assumption that all networks will offer at least an unreliable datagram service, ie if a sequence of packets is injected into the network then the destination will receive some of them, possibly misordered, and with possible duplication. Any improvement on this grade of service will be achieved by implementing end-to-end procedures to perform reordering, retransmission of losses and detection of duplicates. A legitimate criticism of this approach is that these upgrading procedures are acting across all the networks in the chain, which in the case of good networks means that there are extra overheads which involve needless expenditure. Thus, in the hop-by-hop approach, the required level of internet service is provided by procedures implemented across each network. This is obviously more expensive initially, in that the procedures are different for the different networks, but its running costs are cheaper because unnecessary control and retransmissions do not occur across the networks providing the higher grade of service.

There are also two schools of thought on addressing strategy, which are difficult to completely separate out from the ideas set out above. The first school, which has to date been associated with the end-to-end approach, is that all networks worldwide should have a unique network number allocated to it by a global authority. Thus, any host address can be uniquely defined worldwide by concatenating its network number with its host number. The addressing of internet packets is then simple. The other school believes that such international agreement on address formats is not achievable in the near future, and that there will exist multiple naming/addressing authorities. Thus, the address field will have to consist of a list of addresses in different formats, which will be parsed by the gateways of the different naming authorities as the packet wends its way through the internet system. This second system is considerably more flexible than the first, but as we shall see has other consequences as well.

To date, operational systems of the end-to-end variety have used a flat addressing space and the hop-by-hop systems have used the multiple domain system. A schematic representation of the protocol layering involved, in an internetwork connection across three networks, is shown for both the hop-by-hop and end-to-end approaches in figure 2. The hop-by-hop diagram clearly illustrates that the total service is provided by three concatenated services, involving different transport protocols on different types of networks. The end-to-end representation illustrates the singular nature of the transport service, which is independent of attributes of the underlying networks. We will now compare the advantages and disadvantages of the two systems, in the light of operation in the military environment.

1) Running Costs The hop-by-hop approach has the advantage over the end-to-end approach as far as the civilian user is concerned, in that it is very 'tariff' conscious (ie it only uses the minimum amount of transport protocol necessary to provide the required grade of service). Now as many of the European networks provide the high reliability of a virtual call service, this means that hop-by-hop implementations of the transport service for these networks will involve minimum overheads in terms of extra bits to be transmitted, and therefore their running costs will be minimal.

In the end-to-end approach, every packet carries a full internet source and destination address in its header, so that it can make its own way to its destination. In the hop-by-hop approach once the call has been set up, only the destination address for that particular network has to be carried, because the gateways on route contain addressing information for further hops.

2) Development Costs The philosophy of the hop-by-hop approach implies a different protocol for each different type of network. This is not so serious in the civilian environment, because of the considerable influence of the CCITT standards which means that most European public and private networks are of the X25 variety. Even local networks with very high speed interfaces are planning to implement an X25 access. However, in the military environment, where there is a considerably greater range of networks, this could require the development of a number of transport protocols.

3) Trusting Transit Networks When a user makes a multi-net connection, using the hop-by-hop approach, it implies that he trusts the level of transport service being offered by the intermediate gateways in the internet route. Furthermore, it implies that he is happy with the reliability of intermediate gateways which, albeit temporarily, take responsibility for his data at the termination of each hop. We believe that this is a state of affairs that is considerably more acceptable in the benign civilian environment than in the hostile military one.

In the end-to-end approach, only an unreliable datagram delivery service is expected from the set of concatenated networks, and loss of data in any intermediate switching node or gateway will be recovered by a retransmission from the source. Therefore, maintaining the bit integrity of the data transmission does not rely on the continuing correct operation of an intermediate node.

4) Addressing Strategy In the multi-domain address strategy, if a user in one domain wishes to communicate with users in another domain, the user must know the topology of the interconnection of these domains, so that he can supply the information necessary for his data to reach the destination domain. This information could be obtained automatically for him, but it implies separate and possibly different bilateral agreements between the various domain authorities.

In the end-to-end approach with a flat addressing space, each packet contains complete addressing information, and is free to find the best current route across all intermediate networks (figure 3). This dynamic internet routing has similar resource allocation advantages to dynamic routing on single networks. This flexibility of routing in the internet environment is more important in the context of the more rapidly changing scenario of the military environment.

5) Transport Control The end-to-end control is certainly less flexible than the hop-by-hop control. Timeouts in particular, may vary by an order of magnitude, even on the networks in service today. End-to-end flow control, also requires more sophisticated strategies than are needed in the hop-by-hop method.

6) Gateway Complexity One of the chief attractions of the end-to-end approach with flat addressing is the conceptual simplicity and relative smallness of the gateways with respect to the hop-by-hop approach. This is because the only modules that vary from gateway to gateway are the network access modules that pertain to each network (and these are just the modules needed on all hosts attached to that network). The fact that no connection oriented information is held in the gateway, greatly simplifies the action that the gateway has to take on receiving a packet and the amount of buffer storage it needs. This property ties in well with the gateway policy for military networks, namely that networks should be multiply connected by

gateways in order to provide survivable internetwork communications. Thus, the "simplicity" of the gateways will result in cheapness and the ability to provide more than one gateway between every pair of networks.

Thus, although the end-to-end approach involves higher overheads in terms of packet headers we believe that it offers considerably increased survivability in a hostile environment. Furthermore, in a situation in which users and networks are mobile, it is necessary for all networks to come under a single naming/addressing authority (eg NATO) if these changes in topology are to be distributed rapidly and efficiently throughout the internet system.

VII. The ARPA Catenet System

An example of the end-to-end approach with a flat address space, which has been running operationally for about 5 years, is the ARPA catenet system. This system connects about thirty different networks including land-line, satellite and radio based networks, as well as a variety of local area networks. The thinking and concepts involved in the architecture of this system have been fully described in a number of papers [9, 10].

The protocols responsible for data transport in this system and their hierarchical relationship are shown in figure 4.

1) Internet Protocol (IP) [11] This provides for transmitting blocks of data, called datagrams, from sources to destinations. Its main parameters are source and destination addresses which are globally unique. Implementations of this protocol exist in the gateways and internet hosts. The datagrams are routed from one internet module to another through individual networks. In this approach, datagrams may be routed across networks whose maximum packet size is smaller than they are. In this case, a fragmentation module breaks up the packet into smaller packets, replicating enough information in the headers to allow reassembly at the destination. Reassembly does not take place in the gateways, because packets may take different routes to their destinations. There are a number of options available in the internet protocol and these are specified in the control information of the header. Thus, the internet header is of variable length.

2) Transmission Control Protocol (TCP) [12]. TCP is a data transport protocol appropriate to level 4 of the ISO reference model, and is especially designed for use on interconnected systems of networks. TCP is a connection oriented, end-to-end reliable protocol, designed to fit into a layered hierarchy of protocols which support multi-network applications. It provides for reliable interprocess communications, between pairs of processes in host computers, attached to distinct but interconnected computer communication networks. The TCP assumes it can obtain a simple, potentially unreliable, datagram service from the lower level protocols. It fits into a layered protocol architecture just above a basic Internet Protocol, which provides a way for the TCP to send and receive variable length segments of information enclosed in internet datagram envelopes. In order for the TCP to provide a reliable logical circuit between pairs of processes, on top of the less reliable internet communication system, it performs the functions of basic data transfer, data acknowledgement, flow control and multiplexing.

3) Gateway to Gateway Protocol (GGP) [13]. The gateway to gateway protocol is responsible both for distributing routing information through the gateways of the catenet and for advising communicating hosts of routing changes, congestion control and unreachable destinations. The basic routing algorithm, in use today, is the original ARPAnet routing algorithm. This involves gateways telling their nearest neighbours which networks they can reach and how many gateway to gateway hops are involved in the route. If a gateway is directly connected to a network, then it is said to be zero hops to that net. Gateways continuously monitor the state of the network access switch to which they are connected and their nearest neighbour gateways to ensure that routes through them are still available.

VIII. Practical Experience of the ARPA Catenet System

In the autumn of 1978, RSRE set up a collaborative program of research and development in communications with the Advanced Research Projects Agency of the US Department of Defense. This collaborative program involved the connection of the PPSN (Pilot Packet Switched Network), our own in-house research network, to the ARPA catenet system, and providing terminal and file access from an internet host on the PPSN to some of the major ARPAnet hosts. The first two years of the program were allocated to the development and implementation of a reliable connection between PPSN and the ARPA catenet system. We have implemented the DoD standard Transmission Control Protocol, the Internet Protocol and the Gateway-to-Gateway Protocol in Coral 66. In addition, we have made many measurements on the performance of the catenet system, particularly in terms of round-trip delays as the connectivity and the development of the catenet has evolved.

The current configuration is shown in figure 5. The RSRE internet host (PDP-11/23) contains the standard internet protocols of Telnet, TCP and IP, and which run under our own virtual memory operating system EMMOS [19]. The link level protocol, X25 level 2, is used to interface to the PPSN. This protocol is implemented on a microprocessor communication interface (X25 line unit) which is connected to PDP-11 hosts via a standard interface [18].

The PPSN is connected to the rest of the catenet via the RSRE gateway. The gateway (PDP-11/23) has three network interfaces on it, each using a X25 line unit. They are used to provide, 1) access to PPSN, 2) a test port which can be directly connected to a measurement host, and 3) an interface which connects the RSRE gateway to a gateway at University College, London (UCL) via a 9.6k bits/s Post Office line.

The UCL gateway is connected to two other networks, 1) UCL net and 2) Satnet (ARPA packet satellite network). The connection to Satnet is via the Goonhilly SIMP (Satellite Interface Message Processor). Packets destined for Arpanet are forwarded by the Goonhilly SIMP, over the shared 64k bits/s half duplex satellite channel to the Etam SIMP, and from there they are forwarded on to the BBN gateway, and hence into Arpanet.

Catenet Measurements. Some of these measurements were made by echoing packets off the various catenet gateways (figure 5), and a small but representative sample are listed below. By time stamping the packets as they leave the measurement host at RSRE, and then comparing the time stamps with the local time when the packets return, having been echoed off the gateways, the single round trip delay is measured. These delays not only include network transition times, but also any internal delays in the gateways.

The round trip delays from the RSRE measurement host to various gateways, for internet packets containing 6 data bytes are:-

Gateway	Mean Delay (secs)	Min/Max Delay (secs)
RSRE	0.2	0.2
UCL	0.35	0.35 - 0.4
BBN	2.0	1.5 - 3.8
SRI-PRI	2.5	1.9 - 3.8

The results for the RSRE and UCL gateways correspond to the theoretical delays expected due to line speeds. The results for the BBN and SRI-PRI gateways are due to the longer satellite delays and control algorithm of Satnet.

Retransmissions in TCP. TCP can be used for communications over a variety of different networks, therefore the wide variation of round trip delays, as shown above, means that a fixed retransmission period is not suitable, since in some cases there will be significant delays when a TCP segment is lost, while in others there will be unnecessary retransmissions.

To overcome this problem we have implemented a dynamic timeout algorithm for use in TCP. This algorithm measures the time elapsed between sending a data octet with a particular sequence number, and receiving an acknowledgement that covers that sequence number. Using that measured elapsed time as the round trip time (RTT), we compute a smoothed round trip time (SRTT) as:

SRTT = (ALPHA * SRTT) + ((1 - ALPHA) * RTT)

and based on this, compute the retransmission timeout (RTO) as:

RTO = min (BOUND, BETA * SRTT)

where BOUND is an upper bound on the timeout (eg 0.9), and BETA is a delay variance factor (eg 1.5).

The performance of this algorithm has shown to be very good and has significantly reduced the number of unnecessary retransmissions.

IX. Enhancements To The Catenet System

There are a number of situations, peculiar to the military context, which are not catered for by the algorithms presently used in the catenet. Before discussing these and possible enhancements to the catenet which would improve its survivability in the military environment, we must introduce the concepts of "partitioned networks" and "source routing".

A "partitioned network" is one that is so badly damaged that there exists no paths between certain of its switching nodes. Typically, this results in two or more subsets or partitions of nodes, within which communications are possible, but which cannot communicate with each other. Hosts connected to different partitions cannot communicate in the usual way. However, if this network is connected by more than one gateway to the catenet system and there is at least one gateway on each partition, hosts could still communicate by an internetwork path as illustrated in figure 6. The concepts of routing to partitioned networks are concerned with automatic and efficient routing of packets under the conditions mentioned above.

The principle of "source routing" is one of providing some of the routing intelligence in the packet header, by providing not just the destination address, but also some or all of the intermediate node addresses through which the packet has to pass. This facility is provided as an option in the present DoD Internet Protocol.

1) Changes to the Catenet Routing Algorithm. The catenet system as presently configured, permits routing around damaged networks and gateways. It assumes that hosts know the addresses of their local gateways, and are prepared to poll these gateways to determine their status, and have procedures for using alternate gateways, if the primary one is congested or inoperative. Presently, routing to a partitioned network would involve knowing the topology of the catenet and inserting the routing information in the packet header in the form of a source route. This is perfectly feasible, but in a fast changing military environment it would be preferable if the gateways contained enough information to perform automatic routing to hosts on partitioned networks.

If the internet system of gateways is regarded as a super-datagram network, whose node to node protocol is the Internet Protocol, then it would seem reasonable that the internode routing be based on gateway or node identifiers. The routing information distributed to gateways should permit routing to a specific gateway, rather than to a network. As there may be more gateways than networks, this will involve the storage of more information in the gateways than at present. However, if there are additional gatewey nodes for providing survivability it is a waste of resources if the information is not disseminated and used when most needed.

There are two reasons for wishing to change the present catenet routing algorithm:-

(i) The present algorithm suffers from oscillations when certain link failures occur, because it uses repeeted minimization to compute the shortest path. Presently, this problem is overcome by having a narrow range of link costs.

(ii) The granularity, or fineness, of the information distributed by the present algorithm which performs routing to networks, is insufficient for automatic routing to partitioned networks. This is because the route into a destination net via two different gateways may be wildly separated, as illustrated in figure 7. If the network is partitioned, we need to specify the entry into the net rather than just the net.

A recognized candidate for the improved routing algorithm is a modification of the New Arpanet Routing Algorithm [14], which is currently used on Arpanet. Using this algorithm, all the gateways broadcast information to all other gateways using a flooding technique. In particular, two types of information are disseminated:-

(i) Each gateway broadcasts the names of the nets to which it is directly connected.

(ii) Each gateway broadcasts the names of its neighbours with which it can communicate.

From this information, all gateways can determine which networks are partitioned, because a partitioned net will have two or more gateways attached to it which are unable to communicate. Having implemented this algorithm, there are one or two additional techniques that are necessary for dealing with routing to partitioned networks. The main remaining problems are, determining the partition in which the destination host is located, and specifying this in packets to be sent to that host. Now specifying the partition could be accomplished by specifying the identifier of the gateway through which the partition communicates with the rest of the catenet. However, at present there is no format for specifying gateway identifiers in the internet header. The determination of which pertition the destination host is in, is best done by the geteway connected to the source host's network. This gatewey will know how many partitions the destination network is divided into, and the entry gateways to these partitions. When the connection is being set up, the opening peckets will be sent to all partitions, and the resultant reply will contain the relevant pertition identifier. A minor expension of the internet header will be required for specifying gateway identifiers in the internet packet headers.

2) <u>Mobile Hosts in the Military Environment</u>. There are already a number of requirements for aircraft flying from one tactical net to enother, to be able to maintein communication with a ground based command and control centre [15]. There has been considerable discussion on possible solutions to this problem [16,17]. The solution should, if possible avoid using a centralized database, not only because of its vulnerability but also because a separate communication must be successfully performed with the database, as a pre-requisite for a successful connection to the mobile host. Furthermore, as the host moves from one net to another, updates to the database must be made in a timely manner. Obviously, a third party has to be involved if two mobile hosts wish to communicate. However, the ground control centre is a natural anchor for mobile communications, and if the TCP connection identifiere were divorced from physical addresses, the scheme below would provide total data integrity as the mobile host changed networks.

An interesting point, that is immediately highlighted when considering this problem, is that the unique identification of a TCP connection is at present tied down to physical addressee. We believe that this is undesirable, and has led to the present restricted attempts at solving this problem. We believe that unique TCP identifiers should be exchanged at the start of the connection and that these be used throughout, so that any changes in the physical addresses can be exchanged without closing the connection (ie when the aircraft changes nets it inserts its new address in the source address field, this is then used by the ground to continue the connection). It is possible that there will be a little hiccup as the change over from one net to another occurs, because packets may arrive out of order, however retransmission would take care of this. It would obviously be the responsibility of the mobile host to 'login' to the ground centre on entering a network, so that a connection could be opened up from the ground. An alternative approach would be to include another protocol layer directly above the TCP layer. This new protocol would be responsible for opening and closing TCP connections and maintaining data integrity as the mobile host moved onto another network. The disadvantege of this approach, is the necessity to transfer the mobile host's new address on a three way handshake basis, before the host moved onto the new network.

3) <u>Congestion Control in the Catenet</u>. The catenet is essentially a super datagram network, and congestion control consists of using all possible routes to the best advantege and being able to offer a graceful degradation of service when the users demand exceed the network resources. It is important that fairness is exercised in providing a service to users, assuming that they are of the same priority. The above implies thet the cost of a route should change if substantial queues build up on it, so that alternate routes become preferable in an SPF (Shortest Path First) routing algorithm. The change in cost will be reflected in the routing updates, and alternate less congested routes will be preferred. This requires a more realistic measure of internet routing costs, than the number of gateway hops used at present. This needs to be implemented on the catenet for realistic trials,

even though the numbers of alternate routes is very small. Having thus made the best use of the internet resources, the only remaining action is to throttle off users when, by their weight of numbers, they overload the system. This throttling must be fair, bearing in mind priorities. One aspect of the fairness problem is that gateways handle packets on an independent datagram basis and are not therefore conscious of "greedy" users disobeying advisory flow control messages. A full solution of this problem would require a complex control theory model to be solved. This would involve the knowledge of the queuing sizes and delays on all intergateway links. The despatching of packets from the initial gateway would only occur when its journey through the system could be undertaken without it exceeding a specified delay band.

X. Summary

Many of the concepts presented in this paper have been widely discussed in the ARPA internet community. The authors wish to thank their colleagues in the ARPA internet community for many discussions on the concepts presented in this paper.

XII. REFERENCES

[1]. G.A. LaVean, "Interoperability in Defense Communications", IEEE Trans.Comm, vol com-28, no 9, pp 1445-1455, Sept 1980.

[2]. F.F. Kuo, "Defense Packet Switching Networks in the US", Interlinking of Computer Networks, pp 307-313, NATO ASI, Bonas Sept 1978.

[3]. R.B. Stillman and C.R. Defiore, "Computer Security and Networking Protocols", IEEE Trans.Comm, vol com-28, no 9, pp 1472-1477, Sept 1980.

[4]. D.H. Barnes, "Provision of End-to-end Security for User Data on an Experimental Packet Switch Network", IEE 4th Intnl. Conf. on Software Engineering for Telecommunications Switching Systems, Warwick July 1981.

[5]. Reference Model of Open Systems Interconnection, ISO/TC97/SC16/N227, International Standards Organization, 1979.

[6]. CCITT Recommendations X Series "Public Data Networks", Orange Book, ITU, Nov 1980.

[7]. British Post Office User Forum, "A Network Independent Transport Service", Feb 1980.

[8]. J.B. Postel, "Internet Protocol Approaches", IEEE Trans.Comm, vol com-28, no 4, pp 604-611, April 1980.

[9]. V. Cerf and R. Kahn, "A Protocol for Packet Network Interconnection", Comput. Networks, vol 3, pp 259-266, Sept 1974.

[10]. V. Cerf "DARPA Activities in Packet Network Interconnection", Interlinking of Computer Networks, pp 287-313, NATO ISO, Bonas, Sept 1978.

[11]. DARPA, "DOD Standard Internet Protocol", IEN-128, Defense Advanced Research Projects Agency, Jan 1980.

[12]. DARPA, "DOD Standard Transmission Control Protocol", IEN-129, Defense Advanced Research Projects Agency, Jan 1980.

[13]. V. Strazisar, "How to Build a Gateway", Internet Experiment Note 109 Aug 1979.

[14]. J.M. McQuillan, I. Richer and E.C. Rosen, "The New Routing Algorithm for the ARPAnet IEEE Trans.Comm, vol comm-28, no 5, pp 711-719, May 1980.

[15]. V.G. Cerf, "Internet Addressing and Naming in the Tactical Environment", Internet Experiment Note 110, Aug 1979.

[16]. C.A. Sunshine and J.B. Postel, "Addressing Mobile Hosts in the ARPA Internet Environment" Internet Experiment Note 135, March 1980.

[17]. R. Perlman, "Flying Packet Radios and Network Partitions", Internet Experiment Note 146, June 1980.

[18]. A.F. Martin and J.K. Parks, "Intelligent X25 Level 2 Line Units for Switching", Data Networks: Development and Uses, Online Publications Ltd., pp 371-384, 1980.

[19]. S.R. Wiseman and B.H. Davies, "Memory Management Extensions to the SRI Micro Operating Systems for PDP-11/23/34/35/40", Internet Experiment Note 136, May 1980.

[20]. B.H. Davies and A.S. Bates, "Internetworking in Packet Switched Communications: First Report on the RSRE-ARPA Collaborative Program" RSRE Memorandum No 3281, July 1980.

Application Layer	User Programs or Processes that wish to exchange information
Presentation Layer	Concerned with data formats of information exchanged
Session Layer	Concerned with synchronization and delimiting of information exchanges
Transport Layer	Provides a universal data transport layer independent of the underlying network
Network Layer	Provides network access and routing
Link Layer	Provides data transmission over a potentially unreliable link
Physical Layer	Specifies electrical signalling for data and control of the physical medium

FIGURE 1 ISO MODEL FOR OPEN SYSTEM INTERCONNECTION

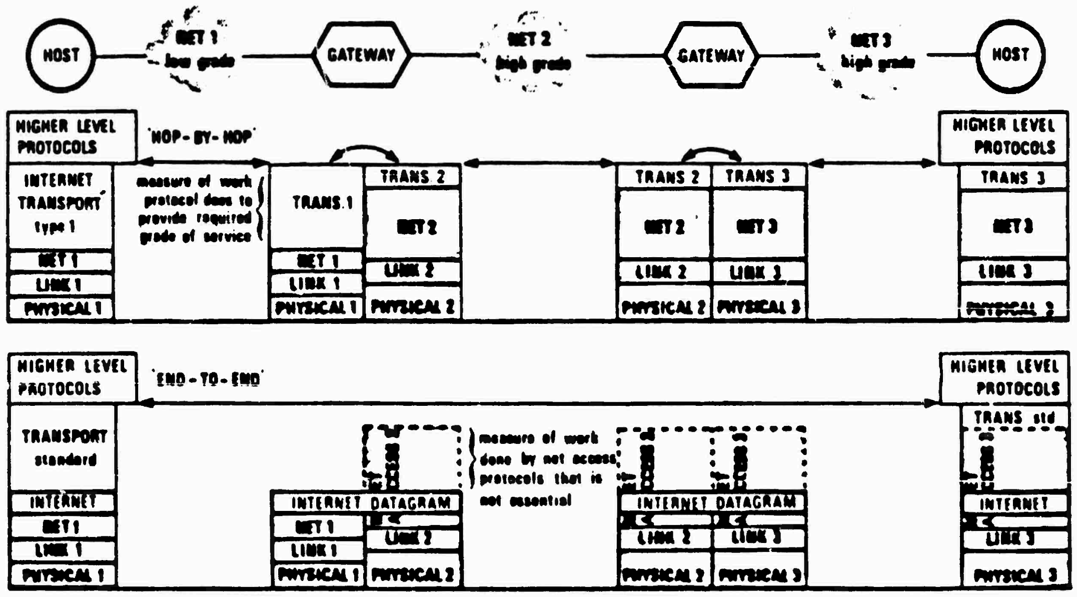
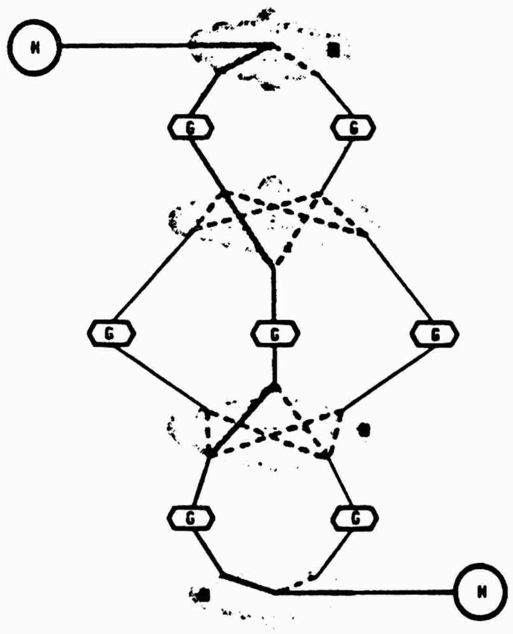


FIGURE 2 REPRESENTATION OF HOP-BY-HOP & END-TO-END PHILOSOPHIES OF INTERNETWORKING

27



Internet datagrams in the "end-to-end" approach may take any of the dashed or the solid routes, but data in the "hop-by-hop" approach takes the solid route set up when the connection was established

FIGURE 3 ROUTING FLEXIBILITY OF END-TO-END APPROACH

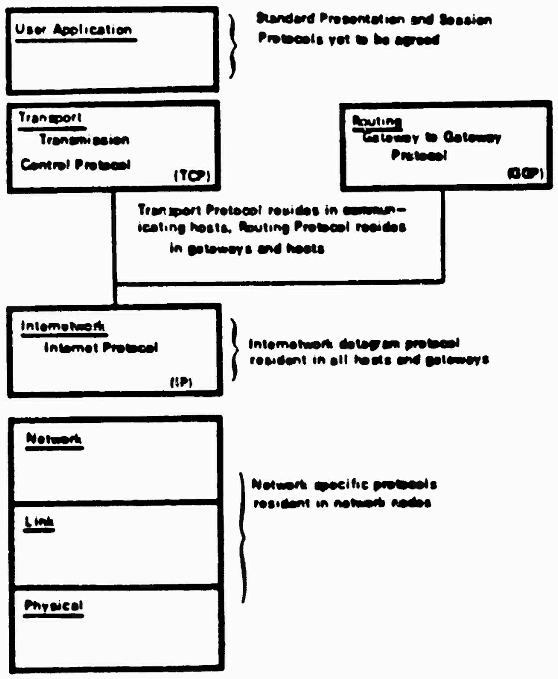

User Application
Standard Presentation and Session Protocols yet to be agreed
Transport
Transmission Control Protocol
(TCP)
Routing
Gateway to Gateway Protocol
(GGP)
Transport Protocol resides in communicating hosts. Routing Protocol resides in gateways and hosts
Internetwork
Internet Protocol
(IP)
Internetwork datagram protocol resident in all hosts and gateways
Network
Link
Physical
Network specific protocols resident in network nodes

FIGURE 4 PROTOCOL LAYERS IN INTERNETWORKING

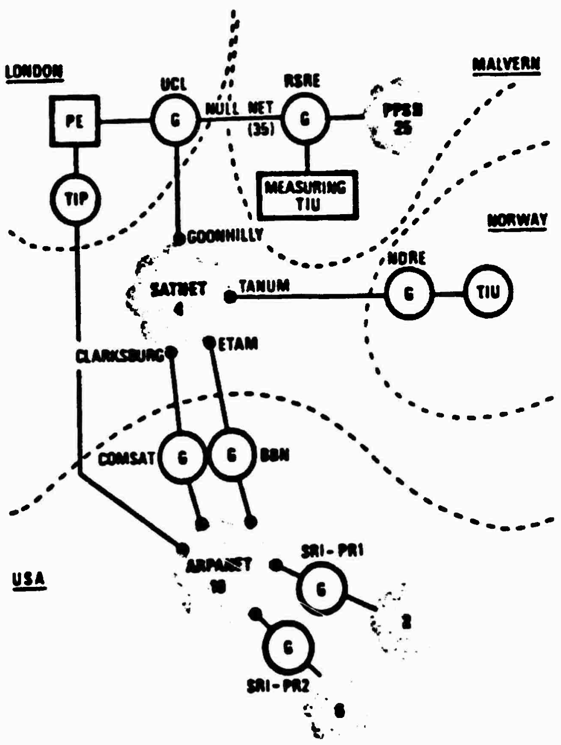
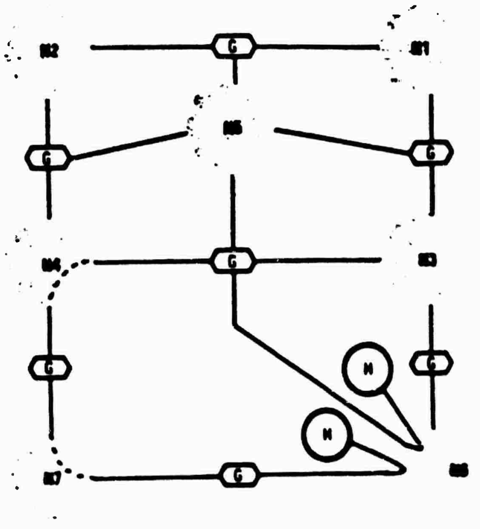


FIGURE 5 MEASUREMENTS CONFIGURATION
JANUARY 1982



Hosts on net 6 are communicating via internet because of heavy damage

FIGURE 6 INTERNET SYSTEM CAN PROVIDE INCREASED COMMUNICATIONS SURVIVABILITY

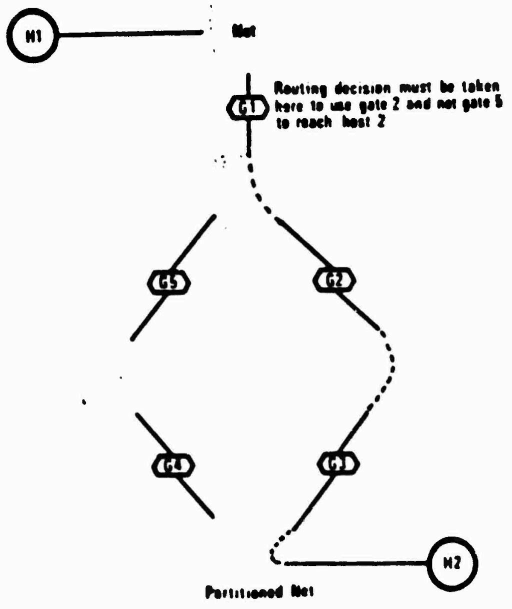

H1
Net
G1
Routing decision must be taken here to use gate 2 and not gate 5 to reach host 2
G5
G2
G4
G3
H2
Partitioned Net

FIGURE 7 ROUTING TO PARTITIONED NETWORKS

Alan Sheltzer, Robert Hinden, and Mike Brescia,
Bolt Beranek and Newman Inc., Cambridge, Mass.

Connecting different types of networks with gateways

Darpa's Internet connects more than 20 networks by gateways, which transmit datagrams and allow adaptive routing.

Just as packet-switching technology matured and spread to commercial applications, internetworking technology is now moving from the research environment into the commercial world. Gateways are being built to interconnect X.25 public packet-switching networks, and many more are planned to link various local networks such as Ethernet.

One of the original interconnected group of networks is the Department of Defense Advanced Research Project Agency's (Darpa) Internet System. It uses communications processors as gateways to link more than 20 networks that use diverse technologies.

The Internet System has been a focal point for internetworking development, with much of the technology supplied by Bolt Beranek and Newman (BBN) of Cambridge, Mass. For example, the Internet gateway transmits information in the form of datagrams and allows different routing schemes to be determined dynamically depending on the best available path. The alternative approach to the datagram model for gateways is the virtual-circuit approach, which determines and establishes a route before information is transmitted. Each scheme has advantages and disadvantages related to congestion, reliability, and overhead

In general, gateways extend network users' abilities to access remote machines, transfer files between different vendors' computers, and send electronic mail. They also provide a solution to the problem of deciding which of the many networking methods is best by allowing all of them to be used, depending on the application. The different types of networks can then be interconnected by gateways, thus giving the user a view of only one large network configuration.

The fundamental technology of gateways is straightforward. For example, two networks "A" and "B," composed of hosts, nodes, and lines, are linked by connecting a communications processor that runs internetworking software to each of the networks. The combination processor and software is a gateway. Host A can send a message to host B on network B by first sending the message to the gateway. The gateway then forwards the message through network B to destination host B.

Several gateways can be used to interconnect a number of different networks. These multiple gateways provide redundancy and additional load capacity.

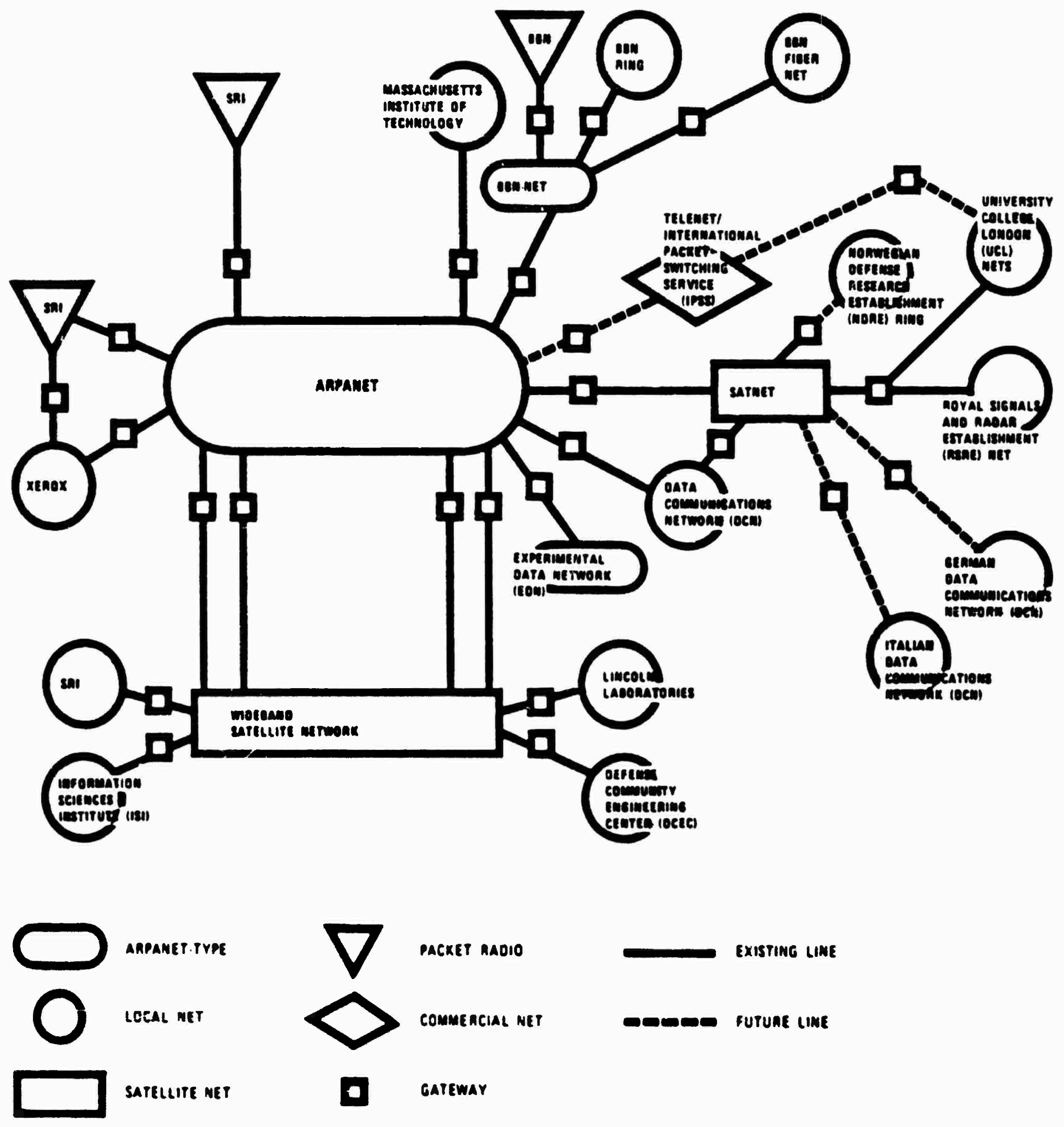
The user view of the interconnected networks is simplified if the gateways are regarded as switching nodes and the networks as lines. Then the entire configuration can be viewed as a single network, built from a collection of separate networks.

Gateways forward messages across networks to other gateways within an internetwork system just as switching nodes forward messages across lines to other switching nodes within a single computer network. However, to provide an efficient, reliable communications service, the gateways should also provide switching node functions such as adaptive routing, flow control, and network monitoring.

The transatlantic connection

There are two approaches to internetworking: the virtual-circuit approach and the datagram. In the architecture that the International Consultative Committee for Telegraphy and Telephony (CCITT) recommends, the internetwork switching nodes provide virtual-circuit service between networks. To do this, each switching node, called an X.75 gateway, is directly connected to X.75 gateways on other networks. When a call is established between two networks, virtual circuits are set up between the source host and an X.75 gateway on the source network, between neighboring X.75

1. The Darpa Internet core. *There are more than 20 networks and gateways, several hundred host computers, and several thousand terminals that make up the Darpa Internet System. The networks are connected in a distributed fashion with multiple paths between networks and alternate paths that span other networks.*


SRI
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
BBN
BBN RING
BBN FIBER NET
BBN NET
TELENET/ INTERNATIONAL PACKET-SWITCHING SERVICE (IPSS)
UNIVERSITY COLLEGE LONDON (UCL) NETS
NORWEGIAN DEFENSE RESEARCH ESTABLISHMENT (NDRE) RING
SRI
ARPANET
SATNET
ROYAL SIGNALS AND RADAR ESTABLISHMENT (RSRE) NET
XEROX
DATA COMMUNICATIONS NETWORK (DCN)
EXPERIMENTAL DATA NETWORK (EDN)
GERMAN DATA COMMUNICATIONS NETWORK (DCN)
ITALIAN DATA COMMUNICATIONS NETWORK (DCN)
SRI
LINCOLN LABORATORIES
WIDEBAND SATELLITE NETWORK
INFORMATION SCIENCES INSTITUTE (ISI)
DEFENSE COMMUNITY ENGINEERING CENTER (DCEC)
ARPANET-TYPE
PACKET RADIO
EXISTING LINE
LOCAL NET
COMMERCIAL NET
FUTURE LINE
SATELLITE NET
GATEWAY

gateways, and between the remote neighbor gateway and the destination host.

Since the X.75 gateway provides virtual-circuit service, it must send messages reliably and in sequence to neighboring X.75 gateways. Flow control between gateways also prevents one gateway from sending more traffic than its neighbors can handle—which is an advantage.

Opponents of the CCITT's virtual-circuit approach to gateways reason that the X.75 architecture is designed to interconnect X.25 networks only and cannot easily link together networks that use different access protocols. These networks include Ethernets, ring networks, and satellite networks. In addition, the X.75 architecture does not provide adaptive routing between networks—when an X.75 call is made, the selection of gateways is fixed. Therefore a failure in one of the X.75 gateways disconnects the call and an alternate route can only be established by making a new call.

The Darpa community has developed an internetwork architecture that allows networks with different access protocols to be interconnected. The Internet System differs in several important ways from the CCITT architecture. For example, gateways are connected directly to networks instead of being connected to other gateways. In addition, traffic is sent across the networks in the form of datagrams instead of via virtual circuits between the networks. And, most importantly, the Darpa Internet uses an adaptive-routing scheme that guarantees that packets exchanged between hosts on different networks travel on the shortest path through gateways. This means that if one gateway fails and there is an alternative gateway available, the alternative gateway will be used automatically without disrupting host-to-host connections.

The current Darpa Internet System consists of more than 20 networks and gateways, several hundred host computers, and several thousand terminals. The Internet networks are connected in a general distributed fashion, with multiple paths between networks and alternate paths that span other networks (Fig. 1). The gateways dynamically decide the best path for a message to be routed to its destination, taking into account topology changes as they occur.

Diverse networks make up the Internet System: terrestrial packet-switching networks such as Arpanet and BBN-Net; satellite networks such as the Atlantic Packet Satellite Network (Satnet) and the Darpa-sponsored Wideband Packet Satellite Network (Wideband); local networks such as Ethernet and the Norwegian Defense Research Establishment (NDRE) Ringnet; and mobile radio networks such as SRI International's packet radio network. These networks vary in characteristics such as message size, speed, delay, reliability, and local address format (Table 1).

It's all in the family

The Darpa research community has developed a family of protocols that provides the mechanisms for host computers to communicate over Internet. These protocols offer services that may be lacking in the underlying networks that make up Internet. As a result of the small number of network requirements, new networks are easily added.

To be part of Internet, a network needs only to be able to deliver messages to a destination and have a minimum message size. The family of Darpa Internet protocols then provides the following services:

- Datagrams
- Addressing
- Message fragmentation and reassembly
- Data reliability
- Message sequencing
- Flow control
- Connections

The Internet System's protocols are a layered family of protocols, as shown in Figure 2. The two main protocols that provide user data transfer are the Internet protocol (IP) [Ref. 1] and the Transmission Control protocol (TCP) [Ref. 2]. In addition, there are protocols for specific applications such as terminal traffic (Telnet), file transfer (FTP), and electronic mail transfer (MTP). Internet also has specialized protocols for functions such as gateway routing, gateway monitoring and control, and error reporting.

Individual network protocols are not specified in the Internet System. Instead, each network has its own access protocols. For instance, Arpanet uses the 1822 Host and IMP protocol (a protocol for interconnection of a host and IMP) [Ref. 3], and Satnet uses the Host Satnet protocol [Ref. 4].

Individual network protocols are used to encapsulate the Internet protocols for transmission across that network. When a message traverses Internet, each gateway creates a new network header appropriate to the next network (Fig. 3).

Datagram delivery

The IP in the second layer of the Internet protocol family transports datagrams across an interconnection of networks. Datagrams are messages that consist of source and destination addresses, plus data. They are not required to be delivered reliably or in sequence. No type of connection needs to be set up to send or receive them. In contrast, virtual-circuit services are provided by high-level end-to-end protocols.

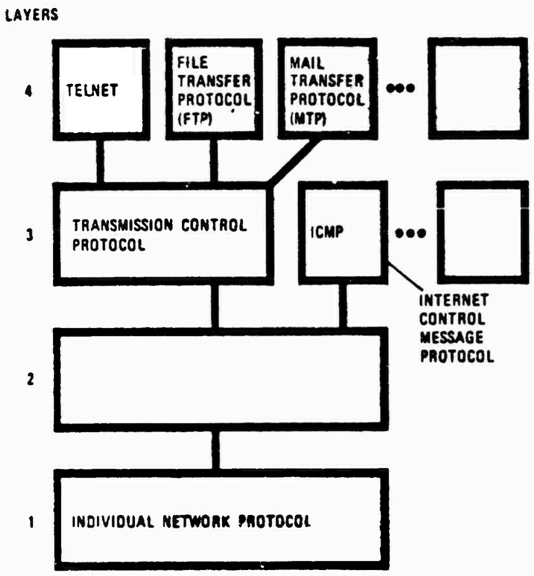
A major advantage of the datagram approach to gateways is that networks are not required to provide many services in order to send a datagram. Therefore, it is comparatively easy to interconnect networks of

Table 1 Network Characteristics

NAME	MESSAGE SIZE IN BYTES	SPEED	DELAY	GUARANTEED DELIVERY	NOTES
ARPANET	1,000	MEDIUM	MEDIUM	YES	
SATNET	256	LOW	HIGH	NO	SATELLITE NETWORK
WIDEBAND	2,000	HIGH	HIGH	NO	SATELLITE NETWORK
PACKET RADIO	254	MEDIUM	MEDIUM	NO	VARYING TOPOLOGY
NDRE RING	2,048	HIGH	LOW	YES	LOCAL NETWORK

WHERE SPEED IS LOW = < 100 KBIT/S, MEDIUM = 100 KBIT/S TO 1 MBIT/S, HIGH = > 1 MBIT/S; AND DELAY IS LOW = < 50 ms, MEDIUM = 50 ms TO 500 ms, OR HIGH = > 500 ms.

2. Internet protocol relationships. *The layered family of Internet protocols permits hosts to communicate over Internet and provides for specific applications.*


LAYERS
4
TELNET
FILE TRANSFER PROTOCOL (FTP)
MAIL TRANSFER PROTOCOL (MTP)
3
TRANSMISSION CONTROL PROTOCOL
ICMP
INTERNET CONTROL MESSAGE PROTOCOL
2
1
INDIVIDUAL NETWORK PROTOCOL

diverse characteristics.

The IP provides two basic services in the second layer: addressing and fragmentation/reassembly. A common address format is maintained across Internet. Addresses are fixed-length (32 bits) and consist of the network number and a local address. The network-number field contains the address of a particular network, and the local-address field contains the address of a host within that network.

The networks that make up Internet have different message sizes. The IP provides a fragmentation/reassembly service to overcome these variations. When a datagram originates in a network that allows large messages, and the datagram must traverse a network with a smaller message limit, the datagram must be broken into smaller "pieces," or fragments. The IP provides a mechanism to permit datagrams to be fragmented and to be later reassembled into one piece at the destination host.

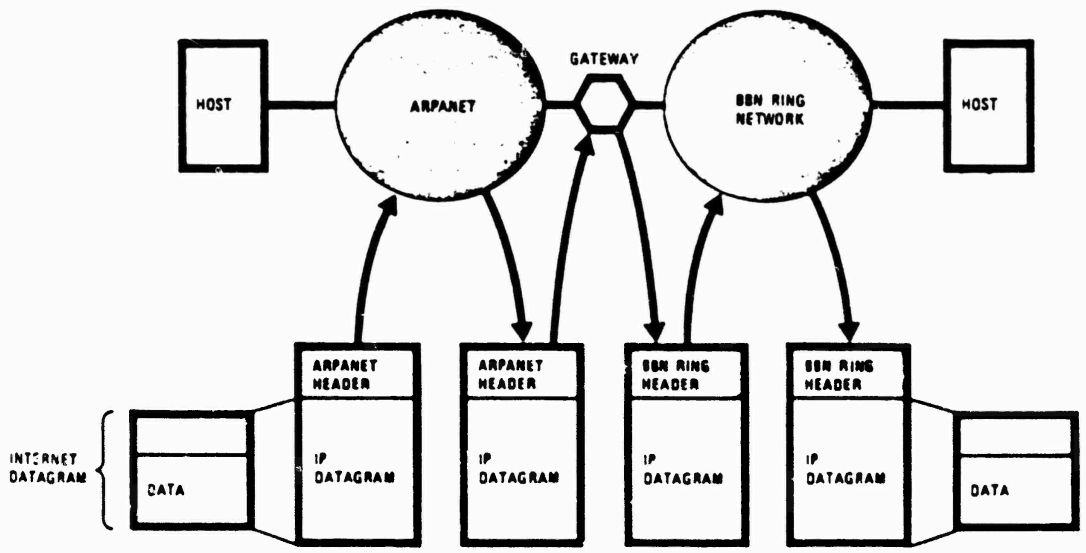
The TCP, in the third layer, is a connection-oriented, reliable end-to-end protocol. It provides the services necessary for reliable message transmission over the Internet System.

The networks that make up Internet are not required to guarantee that all datagrams are delivered. Also, the originator of a datagram does not necessarily know through which networks a datagram will be routed to arrive at its destination. Therefore it is necessary to provide message reliability end-to-end—that is, at the source and the final destination. To address these requirements, the TCP provides reliability, flow control, multiplexing, and connection functions.

Reliability is achieved through checksums (error-detecting codes) and positive acknowledgments of all data. Data that is not acknowledged is retransmitted.

End-to-end flow control lets the receiver of the data regulate the rate at which it is sent. To allow many processes (applications) within a single computer (for example, many terminals talking to one host) to use

3. Message encapsulation. *When a message traverses networks on Internet, individual network protocols are used to encapsulate the Internet protocols for transmission across each network. When the message reaches a gateway, that gateway creates a new network header appropriate to the next network.*



Data Communications/August 1982

Table 2 Network table for BBN gateway

NETWORK NAME	NET ADDRESS	*ROUTE
SATNET	4	DIRECTLY CONNECTED
ARPANET	10	DIRECTLY CONNECTED
BBN-NET	3	1 HOP VIA RCC 10.3.0.72 (ARPANET 3/72)
PURDUE-COMPUTER SCIENCE	192.5.1	2 HOPS VIA PURDUE 10.2.0.37 (ARPANET 2/37)
INTELPOST	43	2 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
DECNET-TEST	38	3 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
WIDEBAND	28	3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
BBN-PACKET RADIO	1	2 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
DCN-COMSAT	29	1 HOP VIA MILLS 10.3.0.17 (ARPANET 3/17)
FIBERNET	24	3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
BRAGG-PACKET RADIO	9	1 HOP VIA BRAGG 10.0.0.38 (ARPANET 0/38)
CLARK NET	8	2 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
LCSNET	18	1 HOP VIA MIT-LCS 10.0.0.77 (ARPANET 0/77)
BBN-TERMINAL CONCENTRATOR	192.1.2	3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
BBN-JERICHO	192.1.3	3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
UCLNET	11	1 HOP VIA UCL 4.0.0.60 (SATNET 60)
RSRE-NULL	35	1 HOP VIA UCL 4.0.0.60 (SATNET 60)
RSRE-PPSN	25	2 HOPS VIA UCL 4.0.0.60 (SATNET 60)
SAN FRANCISCO-PACKET RADIO-2	6	1 HOP VIA C3PO 10.1.0.51 (ARPANET 1/51)

*NAMES AND ACRONYMS IDENTIFY GATEWAYS IN THE INTERNET SYSTEM

the protocol simultaneously, the protocol provides for ports to allow individual processes to be identified. The protocol also provides a mechanism for interprocess communications between computers.

Open the gates

A host computer that wants an IP datagram to reach a host on another network must send the datagram to a gateway. A local-network header containing the address of the gateway is attached to the datagram before it is sent into the network. When the packet is received by the gateway, its local-network header is checked for possible errors and the gateway performs any necessary host-to-network protocol functions.

The Internet control message protocol (ICMP) [Ref. 5] is the control protocol associated with the IP that is used to convey error and status information to Internet users. For example, if the header indicates that the packet contains an Internet datagram, then the packet is passed to the Internet header check routine, which performs a number of validity tests on the IP header. Packets that fail these tests are discarded, and an error packet is sent from the gateway to the Internet source of the packet.

After a datagram passes these checks, its Internet destination address is examined to determine if the datagram is addressed to the gateway. Each of the gateway's Internet addresses—one for each network interface—is checked against the destination address in the datagram. If a match is not found, the datagram is passed to the forwarding routine. If the datagram is destined for the gateway, then the datagram is processed according to the protocol in the IP header. Some types of datagrams that might be addressed to the gateway include monitoring packets, gateway routing packets, or remote debugging packets.

Multiple hops

Among other functions, the gateway must make a routing decision for all datagrams that are to be forwarded. The routing procedure provides two pieces of information: which network interface should be used to send the packet, and which destination address should be in the packet's local-network header.

The gateway maintains a network table that contains an entry for each reachable network (Table 2). The entry consists of a network number and either the address of the neighbor gateway on the shortest route to the network or an indication that the gateway is directly connected to the network. A neighbor gateway is one that shares a common network with this gateway. The distance measurement that is used to determine which neighbor is closest is "number of hops." In other words, a gateway is considered to be zero hops from its directly connected networks, one hop from a network that is reachable via one other gateway, and so on. The Gateway-to-Gateway protocol (GGP) [Ref. 6] is used to build the network table.

The gateway tries to match the destination network address in the IP header of the datagram to be forwarded with a network in its network table. If no match is found, the gateway drops the datagram and sends an ICMP packet to the IP source. If the gateway does find an entry for the network in its table, it uses the network address of the neighbor gateway entry as the local network destination address of the datagram. However, if the final destination network is one to which the gateway is directly connected, the destination address in the local-network header is simply built from the destination address in the datagram's IP header.

If the routing procedure decides that an IP datagram is to be sent back out of the same network interface from which it was read, then the source host has chosen a gateway that is not on the shortest path to the IP final destination. The datagram will still be forwarded to the next address chosen by the routing procedure, but a redirect-ICMP packet will also be sent to the IP source host indicating that another gateway should be used to send traffic to the final IP destination.

Break it up

After the routing decision is complete, the datagram is passed to the fragmentation procedure. If the next network through which the datagram must pass has a smaller maximum packet size than the size of the datagram, the gateway will break the datagram into fragments. These fragments are then transported as independent datagrams themselves and are ultimately collected and assembled at the destination host to recreate the original datagram.

The gateway now builds a new network header for the datagram. The gateway uses the information obtained from its routing procedure to choose the proper network interface for the datagram and to build the destination address in the new network header.

The gateway then queues the packet for delivery to its destination. It also enforces a limit on the size of the output queue for each network interface so that a slow network does not unfairly use up all of the gateway's buffers. A packet that cannot be queued because of the limit on the output-queue length is dropped. Whether or not the packet is retransmitted depends on the type of packet.

When the packet finally reaches its destination, the network header is stripped off and the information inside the IP datagram is processed. In addition, if the original datagram was fragmented, the destination host collects all of the fragments and reassembles them into the original datagram.

To provide Internet service, the IP gateway must support a variety of protocols. For example, the gateway has to send and receive packets on its connected network interfaces. Therefore, it must implement all of these networks' access protocols, such as the Arpanet 1822 protocol or the Satnet Host Access protocol.

Since all Internet traffic is sent in the form of Internet datagrams, the gateway must also implement the IP protocol. In addition, the gateway sends control information, such as "This destination network is unreachable," to hosts using the ICMP protocol.

Monitoring and support of gateways is aided by the Cross Network Debugger protocol [Ref. 7], which allows remote debugging of the gateway, and the Host Monitoring protocol [Ref. 8], which allows the gateway to report the status of its interfaces. The gateway also has an internal message generator that is used as a testing facility.

The right way

The IP gateway uses the GGP for four functions concerned with routing:

- Determining if its network interfaces are operational
- Determining if its neighbor gateways are operational
- Building a table of networks that can be reached via neighbor gateways
- Adding new neighbor gateways and new networks to its network table

Gateways use the information obtained from GGP packets to ensure that a datagram uses the best route through Internet to reach its destination.

GGP packets are sent reliably using sequence numbers and an acknowledgment scheme. The gateway determines if its network interfaces are up by sending GGP packets, called "interface probes," addressed to itself every 15 seconds. When a number of these probes have been successfully received, the interface is declared operational. If a number of probes are missed, the interface is declared down.

In order to determine whether other gateways are operating properly, each gateway has a built-in table of neighbor gateways. Every 15 seconds, a gateway will send a GGP echo packet ("neighbor probe") to each of its neighbors to determine which are operational. When a neighbor gateway has echoed a number of probes, it is declared operational. However, if several probes are sent to a neighbor but are not echoed, the neighbor is declared down.

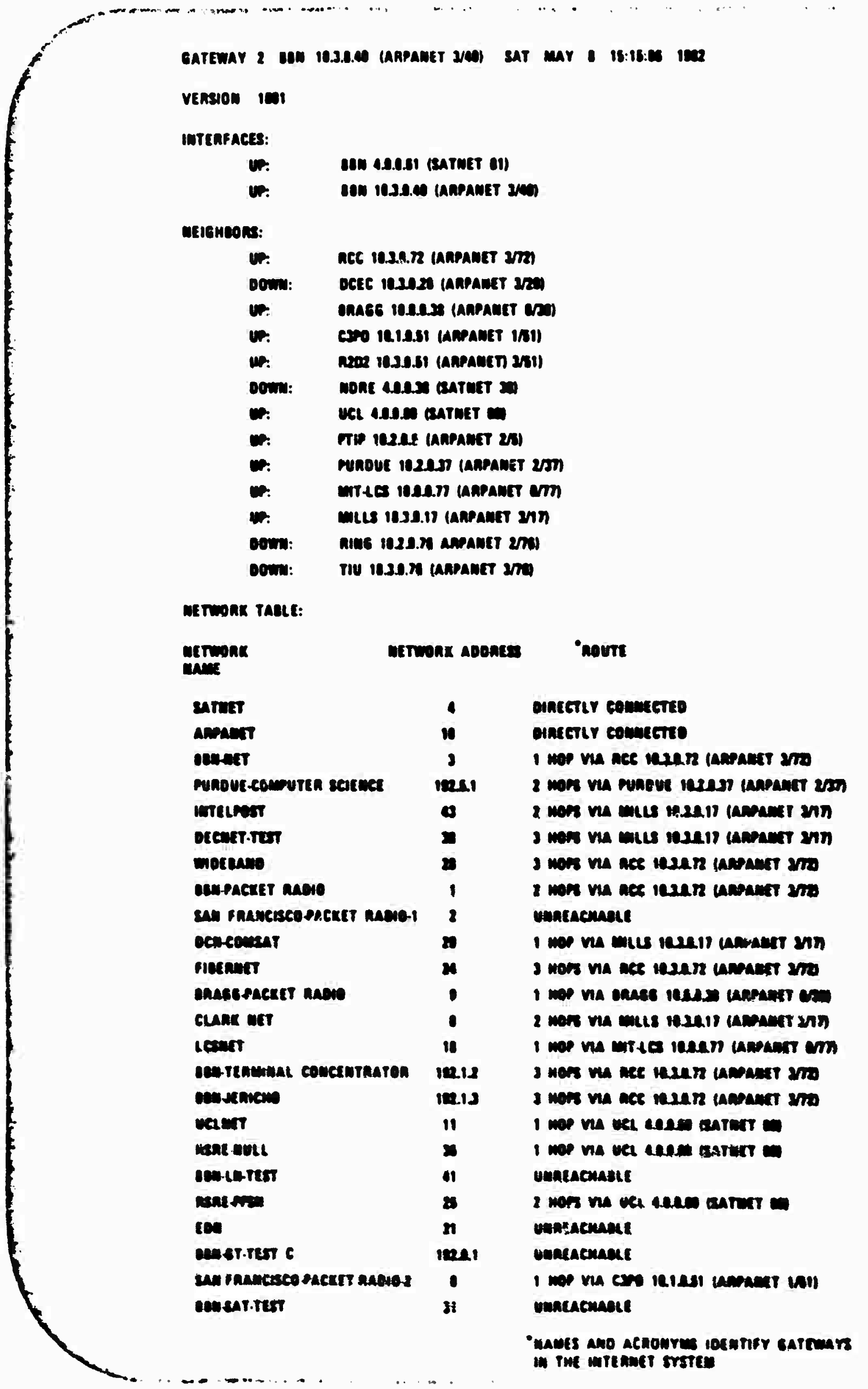
Whenever a gateway determines that there has been a change in Internet routing, such as when it declares one of its network interfaces to be down, it sends a GGP-routing-update packet to each of its neighbors. This packet indicates for each network the distance and address of the gateway on the shortest path to the network.

On receiving a routing update, a gateway will recalculate its network table to ensure that it uses the neighbor on the shortest route to each network. If the routing update packet is from a new neighbor or contains information about a new network, the gateway updates its neighbor or network tables. It thereby learns about new neighbors and networks without having to undergo reconfiguration.

Finding the alternate path

The gateway uses the information in its routing tables to minimize congestion and delay by adapting its routing to the situation. For example, suppose there are two gateways, X and Y, that can be used to reach net-

Table 3 Gateway status report



```
GATEWAY 2  BBN  10.3.0.40  (ARPANET 3/40)   SAT   MAY  8  15:15:06  1982

VERSION   1001

INTERFACES:
          UP:          BBN 4.0.0.61 (SATNET 61)
          UP:          BBN 10.3.0.40 (ARPANET 3/40)

NEIGHBORS:
          UP:          RCC 10.3.0.72 (ARPANET 3/72)
          DOWN:        DCEC 10.3.0.20 (ARPANET 3/20)
          UP:          BRAGG 10.0.0.38 (ARPANET 0/38)
          UP:          C3PO 10.1.0.51 (ARPANET 1/51)
          UP:          R2D2 10.3.0.51 (ARPANET) 3/51)
          DOWN:        NDRE 4.0.0.38 (SATNET 38)
          UP:          UCL 4.0.0.60 (SATNET 60)
          UP:          PTIP 10.2.0.5 (ARPANET 2/5)
          UP:          PURDUE 10.2.0.37 (ARPANET 2/37)
          UP:          MIT-LCS 10.0.0.77 (ARPANET 0/77)
          UP:          MILLS 10.3.0.17 (ARPANET 3/17)
          DOWN:        RING 10.2.0.78 ARPANET 2/78)
          DOWN:        TIU 10.3.0.78 (ARPANET 3/78)

NETWORK TABLE:

NETWORK                      NETWORK ADDRESS       *ROUTE
NAME

 SATNET                            4           DIRECTLY CONNECTED
 ARPANET                          10           DIRECTLY CONNECTED
 BBN-NET                           3           1 HOP VIA RCC 10.3.0.72 (ARPANET 3/72)
 PURDUE-COMPUTER SCIENCE       192.5.1         2 HOPS VIA PURDUE 10.2.0.37 (ARPANET 2/37)
 INTELPOST                        43           2 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
 DECNET-TEST                      38           3 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
 WIDEBAND                         28           3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 BBN-PACKET RADIO                  1           2 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 SAN FRANCISCO-PACKET RADIO-1      2           UNREACHABLE
 DCN-COMSAT                       29           1 HOP VIA MILLS 10.3.0.17 (ARPANET 3/17)
 FIBERNET                         24           3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 BRAGG-PACKET RADIO                9           1 HOP VIA BRAGG 10.0.0.38 (ARPANET 0/38)
 CLARK NET                         8           2 HOPS VIA MILLS 10.3.0.17 (ARPANET 3/17)
 LCSNET                           18           1 HOP VIA MIT-LCS 10.0.0.77 (ARPANET 0/77)
 BBN-TERMINAL CONCENTRATOR     192.1.2         3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 BBN-JERICHO                   192.1.3         3 HOPS VIA RCC 10.3.0.72 (ARPANET 3/72)
 UCLNET                           11           1 HOP VIA UCL 4.0.0.60 (SATNET 60)
 NSRE-NULL                        35           1 HOP VIA UCL 4.0.0.60 (SATNET 60)
 BBN-LN-TEST                      41           UNREACHABLE
 RSRE-PPSN                        25           2 HOPS VIA UCL 4.0.0.60 (SATNET 60)
 EDN                              21           UNREACHABLE
 BBN-GT-TEST C                 192.0.1         UNREACHABLE
 SAN FRANCISCO-PACKET RADIO-2      6           1 HOP VIA C3PO 10.1.0.51 (ARPANET 1/51)
 BBN-SAT-TEST                     31           UNREACHABLE

                                              *NAMES AND ACRONYMS IDENTIFY GATEWAYS
                                               IN THE INTERNET SYSTEM
```

work A. When gateway X goes down, all of its neighbors will send out routing updates reporting that network A is no longer reachable via gateway X. When a gateway receives this routing update it will recalculate its network table and find that gateway Y can be used to reach network A. Gateways will now forward datagrams through gateway Y to reach network A without disrupting any host-to-host connections.

Putting it together

The IP gateway operates on Digital Equipment Corporation PDP-11 or LSI-11 16-bit processors under a small real-time operating system called the Micro Operating System (MOS), developed by SRI International. MOS provides facilities for multiple processes, interprocess communications, buffer management, asynchronous input/output, and a shareable real-time clock.

There is one MOS network process and accompanying data structure called a netblock, which contains information about, for example, network interface status and queueing for each network that is directly connected to the gateway. Each network process waits for input from one of the gateway's interfaces. When an IP datagram is received, the appropriate network process "wakes up" and calls procedures to forward the datagram toward its destination.

The IP gateway is written in Macro-11 assembly language instead of a higher-level language because memory is limited by the 16-bit address space. The gateway code occupies about 10K words of memory. The MOS operating system occupies an additional 3K words of code space, leaving 15K words for buffers. These buffers are shared by various network processes for reading and writing packets.

Adding support to connect a new network to the IP gateway is a relatively easy task. A programmer must write a device driver that handles the hardware interface of the new network as well as a routine to implement the new host-to-network access protocol. The programmer also creates a gateway-configuration file that contains gateway-specific information, such as interface-device addresses. The macro assembler then assembles a new gateway program. This programming task is simplified because more than 75 percent of the code in all IP gateways is identical because of the modularity of the gateway software.

Keeping order

Fault isolation can be a major problem in the daily operation of a computer network. Some issues that must be resolved are: When communications fails, what is to blame? Is the problem with the host machine, the network, the lines, or the user program?

Internet fault isolation is even more difficult because of the number and diversity of users, networks, paths, and requirements involved. For example, the communications path may traverse many networks and gateways so that the potential sources of communications disruption are multiplied.

The ability to identify areas of congestion is also a more complex task. For example, poor performance can be the result of individual networks failing to provide users with adequate throughput or of a bottleneck in connections between networks.

For Bolt Beranek and Newman, the solution to Internet monitoring and control is to apply techniques much like those used to operate Arpanet. In fact, tools developed by the company to monitor the gateways that are the switching nodes of Internet are similar to those used to monitor the switching nodes in Arpanet.

These tools include a central monitoring facility called the network operational center (NOC) [Ref. 9] that runs on BBN's C/70 computer under the Unix operating system. The NOC regularly receives traffic statistics and reports of important events from each of the Internet gateways. Data communications users can interrogate the NOC to find the current status of any Internet gateway. The monitoring facility then prints a gateway status report (Table 3).

The NOC's status- and event-monitoring capabilities pinpoint hardware and software problems during the operation of Internet. For example, when communications is disrupted between Internet hosts, the NOC monitoring tools help determine whether the problem lies with a gateway, network, communications line, or with one of the Internet hosts. Whenever a gateway receives an erroneous packet, a report that identifies the source of the packet is sent to the NOC. These reports help to diagnose malfunctioning hardware and aid in debugging Internet host software.

References

1. "DOD Standard Internet Protocol," RFC: 791, Information Sciences Institute, University of Southern California, Marina del Rey, Calif., September 1981.
2. "DOD Standard Transmission Control Protocol," RFC: 791, Information Sciences Institute, University of Southern California, Marina del Rey, Calif., September 1981.
3. "Specification for the Interconnection of a Host and IMP," BBN Technical Report 1822, Bolt Beranek and Newman, May 1978.
4. D. McNeill, "Host/Satnet Protocol," IEN: 192, Bolt Beranek and Newman, June 1981.
5. "Internet Control Message Protocol," RFC: 792, Information Sciences Institute, University of Southern California, Marina del Rey, Calif., September 1981.
6. V. Strazisar, "How to Build a Gateway," IEN: 109, Bolt Beranek and Newman, August 1979.
7. J. Haverty, "XNET Formats for Internet Protocol Version 4," IEN: 158, Bolt Beranek and Newman, October 1980.
8. B. Littauer, A. Huang, and R. Hinden, "A Host Monitoring Protocol," IEN: 197, Bolt Beranek and Newman, September 1981.
9. P. Santos, B. Chalstrom, J. Linn, and J. Herman, "Architecture of a Network Monitoring, Control, and Management System," Proceedings of the 5th International Conference on Computer Communications, October, 1980.

Note: References are available from Jake Feinler at the Network Information Center, SRI International, Menlo Park, Calif. ■

SECTION 3. APPENDICES

This section contains various auxiliary documents related to protocol development and implementation.

Network Working Group
Request for Comments: 960

Obsoletes RFCs: 943, 923, 900, 870,
820, 790, 776, 770, 762, 758,
755, 750, 739, 604, 503, 433, 349
Obsoletes IENs: 127, 117, 93

J. Reynolds
J. Postel
ISI
December 1985

ASSIGNED NUMBERS

Status of this Memo

This memo is an official status report on the numbers used in protocols in the ARPA-Internet community. Distribution of this memo is unlimited.

Introduction

This Network Working Group Request for Comments documents the currently assigned values from several series of numbers used in network protocol implementations. This RFC will be updated periodically, and in any case current information can be obtained from Joyce Reynolds. The assignment of numbers is also handled by Joyce. If you are developing a protocol or application that will require the use of a link, socket, port, protocol, network number, etc., please contact Joyce to receive a number assignment.

Joyce Reynolds
USC - Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292-6695

Phone: (213) 822-1511

ARPA mail: JKREYNOLDS@USC-ISIB.ARPA

Most of the protocols mentioned here are documented in the RFC series of notes. The more prominent and more generally used are documented in the "Internet Protocol Transition Workbook" [39] or in the old "ARPANET Protocol Handbook" [40] prepared by the NIC. Some of the items listed are undocumented. Further information on protocols can be found in the memo "Official ARPA-Internet Protocols" [104].

In all cases the name and mailbox of the responsible individual is indicated. In the lists that follow, a bracketed entry, e.g., [nn,iii], at the right hand margin of the page indicates a reference for the listed protocol, where the number ("nn") cites the document and the letters ("iii") cites the person. Whenever possible, the letters are a NIC Ident as used in the WHOIS service.

ASSIGNED NETWORK NUMBERS

The network numbers listed here are used as internet addresses by the Internet Protocol (IP) [39,92]. The IP uses a 32-bit address field and divides that address into a network part and a "rest" or local address part. The division takes 3 forms or classes.

The first type of address, or class A, has a 7-bit network number and a 24-bit local address. The highest-order bit is set to 0. This allows 128 class A networks.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|   NETWORK   |                Local Address                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class A Address

The second type of address, class B, has a 14-bit network number and a 16-bit local address. The two highest-order bits are set to 1-0. This allows 16,384 class B networks.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0|           NETWORK         |          Local Address        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class B Address

The third type of address, class C, has a 21-bit network number and a 8-bit local address. The three highest-order bits are set to 1-1-0. This allows 2,097,152 class C networks.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 0|                    NETWORK              | Local Address |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class C Address

Note: No addresses are allowed with the three highest-order bits set to 1-1-1. These addresses (sometimes called "class D") are reserved.

One commonly used notation for internet host addresses divides the 32-bit address into four 8-bit fields and specifies the value of each field as a decimal number with the fields separated by periods. This is called the "dotted decimal" notation. For example, the internet address of USC-ISIB.ARPA in dotted decimal is 010.003.000.052, or 10.3.0.52.

The dotted decimal notation will be used in the listing of assigned network numbers. The class A networks will have nnn.rrr.rrr.rrr, the class B networks will have nnn.nnn.rrr.rrr, and the class C networks will have nnn.nnn.nnn.rrr, where nnn represents part or all of a network number and rrr represents part or all of a local address.

There are four catagories of users of Internet Addresses: Research, Defense, Government (Non-Defense), and Commercial. To reflect the allocation of network identifiers among the categories, a one-character code is placed to the left of the network number: R for Research, D for Defense, G for Government, and C for Commercial (see Appendix A for further details on this division of the network identification).

Network numbers are assigned for networks that are connected to the ARPA-Internet and DDN-Internet, and for independent networks that use the IP family protocols (these are usually commercial). These independent networks are marked with an asterisk preceding the number.

The administrators of independent networks must apply separately for permission to interconnect their network with either the ARPA-Internet of the DDN-Internet. Independent networks should not be listed in the working tables of either the ARPA-Internet or DDN-Internet hosts or gateways.

For various reasons, the assigned numbers of networks are sometimes changed. To ease the transition the old number will be listed for a transition period as well. These "old number" entries will be marked with a "T" following the number and preceding the name, and the network name will be suffixed "-TEMP".

Special Addresses:

In certain contexts, it is useful to have fixed addresses with functional significance rather than as identifiers of specific hosts. When such usage is called for, the address zero is to be interpreted as meaning "this", as in "this network". The address of all ones are to be interpreted as meaning "all", as in "all hosts". For example, the address 128.9.255.255 could be interpreted as meaning all hosts on the network 128.9. Or, the address 0.0.0.37 could be interpreted as meaning host 37 on this network.

Assigned Network Numbers

Class A Networks

*	Internet Address		Name	Network	References
	000.rrr.rrr.rrr			Reserved	[JBP]
R	004.rrr.rrr.rrr		SATNET	Atlantic Satellite Network	[SHB]
D	006.rrr.rrr.rrr	T	YPG-NET-TEMP	Yuma Proving Grounds	[10,BXA]
D	007.rrr.rrr.rrr	T	EDN-TEMP	DCEC EDN	[EC5]
R	008.rrr.rrr.rrr	T	BBN-NET-TEMP	BBN Network	[JSG5]
R	010.rrr.rrr.rrr		ARPANET	ARPANET	[10,40,SA2]
D	011.rrr.rrr.rrr		DODIIS	DoD INTEL INFO SYS	[AY7]
C	012.rrr.rrr.rrr		ATT	ATT, Bell Labs	[MH12]
C	014.rrr.rrr.rrr		PDN	Public Data Network	[REK4]
R	018.rrr.rrr.rrr	T	MIT-TEMP	MIT Network	[20,103,DDC1]
D	021.rrr.rrr.rrr		DDN-RVN	DDN-RVN	[MLC]
D	022.rrr.rrr.rrr		DISNET	DISNET	[FLM2]
D	023.rrr.rrr.rrr		DDN-TC-NET	DDN-TestCell-Network	[DH17]
D	024.rrr.rrr.rrr		MINET	MINET	[10,DHH]
R	025.rrr.rrr.rrr		RSRE-EXP	RSRE	[RNM1]
D	026.rrr.rrr.rrr		MILNET	MILNET	[FLM2]
R	027.rrr.rrr.rrr	T	NOSC-LCCN-TEMP	NOSC / LCCN	[RH6]
R	028.rrr.rrr.rrr		WIDEBAND	Wide Band Satellite Net	[CJW2]
D	029.rrr.rrr.rrr	T	MILX25-TEMP	MILNET X.25 Temp	[MLC]
D	030.rrr.rrr.rrr	T	ARPAX25-TEMP	ARPA X.25 Temp	[MLC]
G*	031.rrr.rrr.rrr		UCDLA-NET	UCDLA-CATALOG-NET	[CXL]
R	032.rrr.rrr.rrr		UCL-TAC	UCL TAC	[PK]
R	036.rrr.rrr.rrr	T	SU-NET-TEMP	Stanford University Network	[PA5]
R	039.rrr.rrr.rrr	T	SRINET-TEMP	SRI Local Network	[GEOF]
R	041.rrr.rrr.rrr		BBN-TEST-A	BBN-GATE-TEST-A	[RH6]
R	044.rrr.rrr.rrr		AMPRNET	Amateur Radio Experiment Net	[HM]
	001.rrr.rrr.rrr-003.rrr.rrr.rrr			Unassigned	[JBP]
	005.rrr.rrr.rrr			Unassigned	[JBP]
	009.rrr.rrr.rrr			Unassigned	[JBP]
	013.rrr.rrr.rrr			Unassigned	[JBP]
	015.rrr.rrr.rrr-017.rrr.rrr.rrr			Unassigned	[JBP]
	019.rrr.rrr.rrr-020.rrr.rrr.rrr			Unassigned	[JBP]
	033.rrr.rrr.rrr-035.rrr.rrr.rrr			Unassigned	[JBP]
	037.rrr.rrr.rrr-038.rrr.rrr.rrr			Unassigned	[JBP]
	040.rrr.rrr.rrr			Unassigned	[JBP]
	042.rrr.rrr.rrr-043.rrr.rrr.rrr			Unassigned	[JBP]
	045.rrr.rrr.rrr-126.rrr.rrr.rrr			Unassigned	[JBP]
	127.rrr.rrr.rrr			Reserved	[JBP]

Class B Networks

*	Internet Address	Name	Network	References
	128.000.rrr.rrr		Reserved	[JBP]
R	128.001.rrr.rrr	BBN-TEST-B	BBN-GATE-TEST-B	[RH6]
R	128.002.rrr.rrr	CMU-NET	CMU-Ethernet	[HDW2]
R	128.003.rrr.rrr	LBL-CSAM	LBL-CSAM-RESEARCH	[JS38]
R	128.004.rrr.rrr	DCNET	LINKABIT DCNET	[69,DLM1]
R	128.005.rrr.rrr	FORDNET	FORD DCNET	[69,DLM1]
R	128.006.rrr.rrr	RUTGERS	RUTGERS	[CLH3]
R	128.007.rrr.rrr	DFVLR	DFVLR DCNET Network	[HDC1]
R	128.008.rrr.rrr	UMDNET	Univ of Maryland DCNET	[69,DLM1]
R	128.009.rrr.rrr	ISI-NET	USC-ISI Local Network	[CMR]
R	128.010.rrr.rrr	PURDUE-CS-NET	Purdue Computer Science	[CAK]
R	128.011.rrr.rrr	BBN-CRONUS	BBN DOS Project	[64,WIM]
R	128.012.rrr.rrr	SU-NET	Stanford University Net	[LB3]
D	128.013.rrr.rrr	MATNET	Mobile Access Terminal Net	[SHB]
R	128.014.rrr.rrr	BBN-SAT-TEST	BBN SATNET Test Net	[SHB]
R	128.015.rrr.rrr	S1NET	LLL-S1-NET	[EAK1]
R	128.016.rrr.rrr	UCLNET	University College London	[PK]
D	128.017.rrr.rrr	MATNET-ALT	Mobile Access Terminal Alt	[SHB]
R	128.018.rrr.rrr	SRINET	SRI Local Network	[GEOF]
D	128.019.rrr.rrr	EDN	DCEC EDN	[EC5]
D	128.020.rrr.rrr	BRLNET	BRLNET	[10,MJM2]
R	128.021.rrr.rrr	SF-PR-1	SF-1 Packet Radio Network	[JEM]
R	128.022.rrr.rrr	SF-PR-2	SF-2 Packet Radio Network	[JEM]
R	128.023.rrr.rrr	BBN-PR	BBN Packet Radio Network	[JAW3]
R	128.024.rrr.rrr	ROCKWELL-PR	Rockwell Packet Radio Net	[EHP]
D	128.025.rrr.rrr	BRAGG-PR	Ft. Bragg Packet Radio Net	[JEM]
D	128.026.rrr.rrr	SAC-PR	SAC Packet Radio Network	[BG5]
D	128.027.rrr.rrr	DEMO-PR-1	Demo-1 Packet Radio Network	[LCS]
D	128.028.rrr.rrr	C3-PR-TEMP	Testbed Development PR NET	[BG5]
R	128.029.rrr.rrr	MITRE	MITRE Cablenet	[111,TML]
R	128.030.rrr.rrr	MIT-NET	MIT Local Network	[DDC1]
R	128.031.rrr.rrr	MIT-RES	MIT Research Network	[DDC1]
R	128.032.rrr.rrr	UCB-ETHER	UC Berkeley Ethernet	[DAM1]
R	128.033.rrr.rrr	BBN-NET	BBN Network	[JSG5]
R	128.034.rrr.rrr	NOSC-LCCN	NOSC / LCCN	[RH6]
R	128.035.rrr.rrr	CISLTESTNET1	Honeywell	[52,53,JLM23]
R	128.036.rrr.rrr	YALE-NET	YALE NET	[128,JQ5]
D	128.037.rrr.rrr	YPG-NET	Yuma Proving Grounds	[10,BXA]
D	128.038.rrr.rrr	NSWC-NET	NSWC Local Host Net	[RLH2]
R	128.039.rrr.rrr	NTANET	NDRE-TIU	[PS3]
R	128.040.rrr.rrr	UCL-NET-A	UCL	[RC7]
R	128.041.rrr.rrr	UCL-NET-B	UCL	[RC7]
R	128.042.rrr.rrr	RICE-NET	Rice University	[69,128,PGM]
R	128.043.rrr.rrr	DRENET	Canada REF ARPANET	[10,JR17]

D 128.044.rrr.rrr	WSMR-NET	White Sands Network	[TBS]
C 128.045.rrr.rrr	DEC-WRL-NET	DEC WRL Network	[128,RKJ2]
R 128.046.rrr.rrr	PURDUE-NET	Purdue Campus Network	[CAK]
D 128.047.rrr.rrr	TACTNET	Tactical Packet Net	[9,KTP]
G*128.048.rrr.rrr	UCDLA-NET-B	UCDLA-Network-B	[10,CXL]
R 128.049.rrr.rrr	NOSC-ETHER	NOSC Ethernet	[128,RLB3]
G 128.050.rrr.rrr	COINS	COINS On-Line Intel Net	[RLS6]
G 128.051.rrr.rrr	COINSTNET	COINS TEST NETWORK	[RLS6]
R 128.052.rrr.rrr	MIT-AI-NET	MIT AI NET	[128,MDC]
R 128.053.rrr.rrr	SAC-PR-2	SAC PRNET Number 2	[BG5]
R 128.054.rrr.rrr	UCSD	UC San Diego Network	[128,GH29]
R*128.055.rrr.rrr	MFENET	LLNL MFE Network	[109,DRP]
D 128.056.rrr.rrr	USNA-NET	US Naval Academy Network	[TXS]
D 128.057.rrr.rrr	DEMO-PR-2	Demo-2 Packet Radio Net	[LCS]
C*128.058.rrr.rrr	SPAR	Schlumberger PA Net	[128,RXB]
R 128.059.rrr.rrr	CU-NET	Columbia University	[128,LH2]
D 128.060.rrr.rrr	NRL-LAN	NRL Lab Area Net	[WF3]
R*128.061.rrr.rrr	GATECH	Georgia Tech	[128,SXA]
R 128.062.rrr.rrr	MCC-NET	MCC Corporate Net	[128,CBD]
R 128.063.rrr.rrr	BRL-SUBNET	BRL-SUBNET-EXP	[RBN1]
R 128.064.rrr.rrr-128.079.rrr.rrr		Net Dynamics Exp	[ZSU]
D 128.080.rrr.rrr	CECOMNET	CECOM EPR NET	[PFS2]
R 128.081.rrr.rrr	SCRC-ETHERNET	SCRC ETHERNET	[128,CH2]
R 128.082.rrr.rrr	UMICH	UOFMICHIGAN	[8,HWB]
R 128.083.rrr.rrr	UTAUSTIN	U. Texas Austin	[128,JSQ1]
R 128.084.rrr.rrr	CORNELL-NET	Cornell Backbone Net	[128,BN9]
C*128.085.rrr.rrr	DRILL-NET	Teleco Drilltech Net	[DBJ]
R 128.086.rrr.rrr	MRC	UK.CO.GEC.RL.MRC	[RHC3]
R 128.087.rrr.rrr	HIRST	UK.CO.GEC.RL.HRC	[RHC3]
R*128.088.rrr.rrr	HP-NET	HEWLETT-PACKARD-NET	[AXG]
R 128.089.rrr.rrr	BBN-ENET-TEMP	BBN ETHER NETWORK	[128,SGC]
C*128.090.rrr.rrr	PQS	PERQ SYSTEMS CORP	[128,DXS]
R 128.091.rrr.rrr	UPENN	UPenn Campus Network	[128,IXW]
R 128.092.rrr.rrr	INTELLINET	INTELLICORP NET	[128,DAVE]
R*128.093.rrr.rrr	INRIA-ROCQU	INRIA Rocquencourt	[MXA1]
R*128.094.rrr.rrr	SYSNET	AT&T SYSNETWORK	[EXY]
R*128.095.rrr.rrr	WASHINGTON	Comp Sci Ether Net	[128,RA17]
C*128.096.rrr.rrr	BELLCORE-NET	BELLCORE-NET	[PK28]
R 128.097.rrr.rrr	UCLANET	UCLA Network	[BJL5]
128.098.rrr.rrr-191.254.rrr.rrr		Unassigned	[JBP]
191.255.rrr.rrr		Reserved	[JBP]

Class C Networks

*	Internet Address	Name	Network	References
	192.000.000.rrr		Reserved	[JBP]
R	192.000.001.rrr	BBN-TEST-C	BBN-GATE-TEST-C	[RH6]
	192.000.002.rrr-192.000.255.rrr		Unassigned	[JBP]
R	192.001.000.rrr-192.001.004.rrr		BBN local networks	[SGC]
R	192.001.005.rrr	BBN-ENET2	BBN-ENET2	[SGC]
R	192.001.006.rrr		BBN local network	[SGC]
R	192.001.007.rrr	BBN-ENET	BBN-ENET	[SGC]
R	192.001.008.rrr		BBN local network	[SGC]
R	192.001.009.rrr	BBN-ENET3	BBN-ENET3	[SGC]
R	192.001.010.rrr	BBN-NETR	BBN-NETR	[SGC]
R	192.001.011.rrr	BBN-SPC-ENET	BBN-SPC-ENET	[SGC]
R	192.001.012.rrr-192.003.255.rrr		BBN local networks	[SGC]
R*	192.004.000.rrr-192.004.255.rrr		BELLCORE-NET	[128,PK28]
R	192.005.001.rrr	CISLHYPERNET	Honeywell	[JLM23]
R	192.005.002.rrr	WISC	Univ of Wisconsin Madison	[RS23]
C	192.005.003.rrr	HP-DESIGN-AIDS	HP Design Aids	[NXK]
C	192.005.004.rrr	HP-TCG-UNIX	Hewlett Packard TCG Unix	[NXK]
R	192.005.005.rrr	DEC-MRNET	DEC Marlboro Ethernet	[119,KWP]
R	192.005.006.rrr	DEC-MRRAD	DEC Marlboro Developmt	[119,KWP]
R	192.005.007.rrr	CIT-CS-NET	Caltech-CS-Net	[126,DSW]
R	192.005.008.rrr	WASHINGTON	University of Washington	[JAR4]
R	192.005.009.rrr	AERONET	Aerospace Labnet	[2,LCN]
R	192.005.010.rrr	ECLNET	USC-ECL-CAMPUS-NET	[MAB4]
R	192.005.011.rrr	CSS-RING	SEISMIC-RESEARCH-NET	[RR2]
R	192.005.012.rrr	UTAH-NET	UTAH-COMPUTER-SCIENCE-NET	[GW22]
R	192.005.013.rrr	GSWDNET	Compion Network	[128,FAS]
R	192.005.014.rrr	RAND-NET	RAND Network	[128,JDG]
R	192.005.015.rrr	NYU-NET	NYU Network	[EF5]
R	192.005.016.rrr	LANLLAND	Los Alamos Dev LAN	[128,JC11]
R	192.005.017.rrr	NRL-NET	Naval Research Lab	[AP]
R	192.005.018.rrr	IPTO-NET	ARPA-IPTO Office Net	[SA2]
R	192.005.019.rrr	UCIICS	UCI-ICS Res Net	[MTR]
R	192.005.020.rrr	CISLTTYNET	Honeywell	[JLM23]
D	192.005.021.rrr	BRLNET1	BRLNET1	[10,MJM2]
D	192.005.022.rrr	BRLNET2	BRLNET2	[10,MJM2]
D	192.005.023.rrr	BRLNET3	BRLNET3	[10,MJM2]
D	192.005.024.rrr	BRLNET4	BRLNET4	[10,MJM2]
D	192.005.025.rrr	BRLNET5	BRLNET5	[10,MJM2]
D	192.005.026.rrr	NSRDCOA-NET	NSRDC Office Auto Net	[TC4]
D	192.005.027.rrr	DTNSRDC-NET	DTNSRDC-NET	[TC4]
R	192.005.028.rrr	RSRE-NULL	RSRE-NULL	[RNM1]
R	192.005.029.rrr	RSRE-ACC	RSRE-ACC	[RNM1]
R	192.005.030.rrr	RSRE-PR	RSRE-PR	[RNM1]
R*	192.005.031.rrr	SIEMENS-NET	Siemens Research Network	[PXN]

R 192.005.032.rrr	CISLTESTNET2	Honeywell	[52,53,JLM23]
R 192.005.033.rrr	CISLTESTNET3	Honeywell	[32,33,JLM23]
R 192.005.034.rrr	CISLTESTNET4	Honeywell	[32,33,JLM23]
R 192.005.035.rrr	RIACS	USRA	[113,RLB1]
R 192.005.036.rrr	CORNELL-CS	CORNELL CS Research	[128,DK2]
R 192.005.037.rrr	UR-CS-NET	U of R CS 3Mb Net	[67,LB1]
R 192.005.038.rrr	SRI-C3ETHER	SRI-AITAD C3ETHERNET	[128,BG5]
R 192.005.039.rrr	UDEL-EECIS	Udel EECIS LAN	[120,CC2]
R 192.005.040.rrr	PUCC-NET-A	PURDUE Comp Cntr Net	[JRS8]
D 192.005.041.rrr	WISLAN	WIS Research LAN	[111,JRM1]
D 192.005.042.rrr	AFDSC-HYPER	AFDSC Hypernet	[MCA1]
R 192.005.043.rrr	CUCSNET	Columbia CS Net	[128,LH2]
R 192.005.044.rrr	Farber-PC-Net	Farber PC Network	[DJF]
R 192.005.045.rrr	AIDS-NET	AI&DS Network	[128,KFD]
R 192.005.046.rrr	NTA-RING	NDRE-RING	[PS3]
R 192.005.047.rrr	NSRDC	NSRDC	[PXM]
R 192.005.048.rrr	PURDUE-CS-EN	Purdue CS Ethernet	[128,CAK]
R 192.005.049.rrr	UCSF	Univ of Calif, San Fran	[120,TE6]
R 192.005.050.rrr	CTH-CS-NET	Chalmers CSN Net	[120,UXB]
R 192.005.051.rrr	Theorynet	Cornell Theory Center	[128,AB13]
R 192.005.052.rrr	NLM-ETHER	NLM-LHNCBC-ETHERNET	[92,JA1]
R 192.005.053.rrr	UR-CS-ETHER	U of R CS 10Mb Net	[67,LB1]
R 192.005.054.rrr	AERO-A6	Aerospace	[2,LCN]
R 192.005.055.rrr	UCLA-CECS	UCLA-CECS Network	[128,RBW]
C 192.005.056.rrr	TARTAN-NET	Tartan Labs	[SXB]
R 192.005.057.rrr	UDEL-CC	UDEL Comp Center	[120,RR18]
R 192.005.058.rrr	CSNET-PDN	CSNET X.25 Network	[60,RDR4]
R*192.005.059.rrr	INRIA SM90	Inria GIP SM-90	[MXS]
R*192.005.060.rrr	SM90 X1	Inria SM-90 exp. 1	[MXS]
R*192.005.061.rrr	SM90 X2	Inria SM-90 exp. 2	[MXS]
R*192.005.062.rrr	LITP SM90	LITP SM-90	[MXS]
R 192.005.064.rrr	AMES-NAS-NET	NASA ARC NAS LAN	[119,MF31]
R 192.005.065.rrr	NPRDC-Ether	NPRDC TRCF Ethernet	[LRB]
R 192.005.066.rrr	HARV-NET	Harvard Comp Sci Net	[SB28]
R 192.005.067.rrr	CECOM-ETHER	CECOM ADDCOMPE ETHER	[120,GIH]
R 192.005.068.rrr	AERO-130	AEROSPACE-130	[LCN]
R 192.005.069.rrr	UIUC-NET	Univ of IL at Urbana	[128,AKC]
G 192.005.070.rrr	CELAN	COINS Exper. LAN	[MXM]
R 192.005.071.rrr	SAC-ETHER	SAC C3 Ethernet	[128,BG5]
R*192.005.072.rrr-192.005.087.rrr		U Chicago	[TXN]
R 192.005.088.rrr	YALE-EE-NET	YALE-EE-NET	[128,AG22]
R 192.005.089.rrr	HARV-APPOLLO	Harvard University	[4,SB28]
R 192.005.090.rrr	HARV-ETHER	Harvard CS Ethernet	[SB28]
R 192.005.091.rrr	PURDUE-ECN1	Purdue ECN	[36,55,GG11]
R 192.005.092.rrr	BRAGG-ETHER	SRI Bragg Ether	[121,GIH]
R 192.005.093.rrr	SRI-DEMO	SRI Ether Demo	[121,GIH]
R*192.005.094.rrr	SDCRDCF-10MB	SDC R&D primary net	[128,DJV1]
R*192.005.095.rrr	SDCRDCF-3MB	SDC R&D old net	[67,DJV1]

R*192.005.096.rrr	UBC-CS-NET	UBC Comp Sci Net	[128,PXB]
R*192.005.097.rrr	UCLA-CS-LNI	UCLA CS LNI Network	[RBW]
R*192.005.098.rrr	UCLA-PIC	UCLA PIC Network	[128,RBW]
R 192.005.099.rrr	SPACENET	S-1 Workstation Net.	[128,TW11]
R*192.005.100.rrr	HCSC-NET	Honeywell CSC Net	[128,RL2]
R 192.005.101.rrr	PUCC-NET-B	Purdue Gateway Network	[JRS8]
R 192.005.102.rrr	PUCC-RHF-NET	PUCC RHF Based Net	[JRS8]
C*192.005.103.rrr	TYM-NTD-NET	Tymnet NTD Ethernet	[SMF]
R 192.005.104.rrr	THINK-INET	Thinking Machines	[128,BJN1]
R 192.005.105.rrr	CCA-POND	CCA Ethernet1 (POND)	[128,AL6]
C*192.005.106.rrr	BITSTREAM	Bitstream Type Foundry	[128,PXA]
R*192.005.107.rrr	PASC-ETHER	IBM PASC Ethernet	[128,GXL]
R*192.005.108.rrr	PASC-BB	IBM PASC Broadband	[56,GXL]
R*192.005.109.rrr	CWR-JCC-T	ARJCC TOPS-20 NET	[128,JAG3]
R*192.005.110.rrr	CWR-JCC-L	ARJCC LOCAL NET	[128,JAG3]
R*192.005.111.rrr	CWR-QUAD	Campus QUAD NET	[128,JAG3]
R*192.005.112.rrr	CWR-CAISR	CAISR LOCAL NET	[128,JAG3]
R*192.005.113.rrr	CWR-CES	CES LOCAL NET	[JAG3]
C*192.005.114.rrr	I2-RING-1	INTERMETRICS PRONET	[128,NXH]
C*192.005.115.rrr	I2-ETHER-1	INTERMETRICS ETHER	[128,NXH]
R 192.005.116.rrr	BRAGGNET-1	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.117.rrr	BRAGGNET-2	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.118.rrr	BRAGGNET-3	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.119.rrr	BRAGGNET-4	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.120.rrr	BRAGGNET-5	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.121.rrr	BRAGGNET-6	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.122.rrr	BRAGGNET-7	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.123.rrr	BRAGGNET-8	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.124.rrr	BRAGGNET-9	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.125.rrr	BRAGGNET-10	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.126.rrr	BRAGGNET-11	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.127.rrr	BRAGGNET-12	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.128.rrr	BRAGGNET-13	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.129.rrr	BRAGGNET-14	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.130.rrr	BRAGGNET-15	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.131.rrr	BRAGGNET-16	BRAGG/ADDCOMPE	[128,BG25]
R 192.005.132.rrr	BRAGGNET-17	BRAGG/ADDCOMPE	[128,BG25]
R*192.005.133.rrr	PERCEPT-AI	Perceptronics, AI Div.	[KXC]
192.005.134.rrr-192.005.255.rrr		Unassigned	[JBP]
C*192.006.000.rrr-192.006.255.rrr		Hewlett Packard	[AXG]
C*192.007.000.rrr-192.007.255.rrr		Computer Consoles, Inc.	[RA11]
C*192.008.000.rrr-192.008.255.rrr		Spartacus Computers, Inc.	[SXM]
C*192.009.000.rrr-192.009.255.rrr		SUN Microsystems, Inc.	[BN4]
C*192.010.000.rrr-192.010.040.rrr		Symbolics, Inc.	[CH2]
R 192.010.041.rrr	T SCRC-ETHERNET	SCRC ETHERNET	[128,CH2]
C*192.010.042.rrr-192.010.255.rrr		Symbolics, Inc.	[CH2]
C*192.011.000.rrr-192.011.255.rrr		ATT, Bell Labs	[MH12]

C*192.012.000.rrr	CADMUS-ETHERNET	CADMUS-NET	[MS9]
C*192.012.001.rrr	CADMUS-EXP-1	CADMUS-NET-EXP-1	[MS9]
C*192.012.002.rrr	CADMUS-EXP-2	CADMUS-NET-EXP-2	[MS9]
C*192.012.003.rrr	FLAIR	Fairchild AI Lab Net	[128,AMS1]
C*192.012.004.rrr	SCG-NET	Hughes SCG Net	[122,MXP]
R 192.012.005.rrr	AIC-LISPMS	SRI-AIC-LispMachNet	[128,PM4]
R 192.012.006.rrr	NPS-C2	NPS-C2	[128,AW9]
R 192.012.007.rrr	NYU-CS-ETHER	NYU CompSci Ethernet	[128,LOU]
D 192.012.008.rrr	PICANET1	Picatinny Arsenal LAN1	[128,RFD1]
R 192.012.009.rrr	CADRE-NET	Decision Systems Lab	[SM6]
R 192.012.010.rrr	CORNELL-ENG	Cornell-Engineering	[128,BN9]
R 192.012.011.rrr	MIT-TEST	MIT Gateway TEST NET	[128,NC3]
R 192.012.012.rrr	WISC-ETHER	Wisconsin Ether Net	[128,CBP]
R 192.012.013.rrr	JHU-NET1	JHU-NET1	[128,MO14]
R 192.012.014.rrr	JHU-NET2	JHU-NET2	[128,MO14]
R 192.012.015.rrr	BROOKNET	BNL Brooknet III	[128,GC]
R 192.012.016.rrr	PRMNET	SRI-SURAN-EN	[128,BP17]
G 192.012.017.rrr	LLL-TIS-NET	LLL-TIS-NET	[119,123,GP10]
R 192.012.018.rrr	CIT-CS-10NET	Caltech 10Meg EtherNet	[126,AD22]
R 192.012.019.rrr	CIT-NET	Caltech Campus Net	[126,AD22]
R 192.012.020.rrr	CIT-SUN-NET	Caltech Sun Net	[126,AD22]
R 192.012.021.rrr	CIT-PHYSCOMP	Caltech Phys Comp Net	[126,AD22]
R 192.012.022.rrr	UTCSRES	UTCS Net Research	[128,JSQ1]
R 192.012.023.rrr	UTCSTTY	UTCS TTY Kludgenet	[128,JSQ1]
R 192.012.024.rrr	MICANET	MITRE (Experimental)	[WDL]
R 192.012.025.rrr	CSS-GRAMINAE	CSS Workstation Net	[62,RR2]
R 192.012.026.rrr	NOSC-NETR	Net-R Testbed at BBN	[106,CP10]
R 192.012.027.rrr	UR-LASER	UR Laser Energetics	[128,WXL]
R*192.012.028.rrr	RIACS-X-NET	RIACS-Experimental-Net	[DG28]
D 192.012.029.rrr	RF-EVANS	ADDCOMPE DC3 LAN1	[120,MB31]
D 192.012.030.rrr	RF-HEX-A	ADDCOMPE DC3 LAN2	[120,MB31]
D 192.012.031.rrr	USNA-ENET	USNA Engineering Net	[120,TXS]
R*192.012.032.rrr	CMU-VINEYARD	CMU File Cluster Net	[128,MXK]
R 192.012.033.rrr	SRI-CSL-NET	SRI-CSL 10MB Ethernet	[GEOF]
C*192.012.034.rrr-192.012.043.rrr		Schlumberger PA Net	[128,RXB]
R 192.012.044.rrr	NRTC-NET	Northrop Research Net	[128,RSM1]
R 192.012.045.rrr	ACC-SB-IMP-NET	ACC Santa Barbara IMP	[AB20]
R 192.012.046.rrr	ACC-SB-ETHER	ACC Santa Barbara Ethernet	[AB20]
R 192.012.047.rrr	UMN-UCC-NET	Univ. of Minnesota	[RG12]
G 192.012.048.rrr	AMES-ED-EXPNET	Code ED Exp. Net.	[128,MSM1]
G 192.012.049.rrr	AMES-ED-NET	Code ED IP Net	[128,MSM1]
G 192.012.050.rrr	AMES-DB-NET	Ames DBridge Net	[128,MSM1]
R 192.012.051.rrr	THINK-CHAOS	TMC Chaos	[128,BJN1]
R*192.012.052.rrr	NEURO-NET	NEURO-NET	[128,JXB]
R*192.012.053.rrr	PU-LCA	Princeton U. LCA	[128,CXH]
R 192.012.054.rrr	WISC-MADISON	Univ Wisc - MACC	[128,JXD]
R 192.012.055.rrr	HAZ-LPR-BETA	Hazeltine LPR Net	[128,KXK]
R 192.012.056.rrr	UTAH-AP-NET	Utah-Appolo-Ring-Net	[JL15]

R 192.012.057.rrr	MCC-CAD-NET	MCC AI Subnet	[128,CBD]
R 192.012.058.rrr	MCC-PP-NET	MCC CAD Subnet	[128,CBD]
R 192.012.059.rrr	MCC-DB-NET	MCC DB Subnet	[128,CBD]
R 192.012.060.rrr	MCC-HI-NET	MCC HI Subnet	[128,CBD]
R 192.012.061.rrr	MCC-SW-NET	MCC SW Subnet	[128,CBD]
R 192.012.062.rrr	DREA-ENET	DREA Lispm & Vaxen	[128,GLH5]
R 192.012.063.rrr	CYPRESS	CYPRESS Serial Net	[CAK]
D 192.012.064.rrr	LOGNET	Logistics Net GW	[62,JXR]
D 192.012.065.rrr	HELNET1	HELNET1	[128,MJM2]
D 192.012.066.rrr	HELNET2	HELNET2	[128,MJM2]
D 192.012.067.rrr	HELNET3	HELNET3	[MJM2]
G 192.012.068.rrr	ORNL-MSRNET	ORNL Local Area Net	[62,HD]
R 192.012.069.rrr	UA-CS-NET	UNIV. OF ARIZ-CS DEPT	[128,BXM]
R 192.012.070.rrr	NPRDC-IPD	NPRDC-IPD REMOTE ETHERNET	[LRB]
R 192.012.071.rrr	NPRDC-ISG	NPRDC-ISG REMOTE ETHERNET	[LRB]
R 192.012.072.rrr	ULCC	UK.AC.ULCC	[RHC3]
R 192.012.073.rrr	BTRL	UK.CO.BT-RESEARCH-LABS	[RHC3]
R*192.012.074.rrr	APPLE-ETHER	APPLE COMPUTER ETHER	[128,RXJ]
R*192.012.075.rrr	PASC-RING	IBM PASC TOKEN RING	[GXL]
R*192.012.076.rrr	UQ-NET	UNIV. OF QLD NETWORK	[128,AXH]
C*192.012.077.rrr	PRIME	PRIME COMPUTER, INC.	[FXS]
C*192.012.078.rrr	GENNET	GENENTECH NET	[128,SXM]
C*192.012.079.rrr	SLI	SOFTWARE LEVERAGE INC.	[MXG]
R 192.012.080.rrr	CAEN	UMICH-CAEN	[HWB]
R 192.012.081.rrr	YALE-RING-NET	YALE RESEARCH RING	[RC77]
C 192.012.082.rrr	CU-CC-NET	Columbia CC Net	[128,BC14]
G*192.012.083.rrr	UCDLA-EXNET	UCDLA EXPERIMENTAL NET	[CXL]
G*192.012.084.rrr	UCDLA-PCNET	UCDLA PERSONAL NET	[CXL]
G*192.012.085.rrr	UCDLA-OPNET	UCDLA OPTICAL DISK	[CXL]
G*192.012.086.rrr	UCDLA-RADNET	UCDLA PACKET RADIO	[CXL]
G*192.012.087.rrr	UCDLA-CSLNET	UCDLA STATE LIBRARY	[CXL]
R*192.012.088.rrr	RUTGERS-NWK	RUTGERS, NEWARK	[DXB]
R 192.012.089.rrr	SBCS-CSDEPT-1	SB Computer Science	[JXS]
R 192.012.090.rrr	SBCS-CSDEPT-2	SB Computer Science	[JXS]
R*192.012.091.rrr	RPICSNET0	RPICS-LOCALNET-0	[MS9]
R*192.012.092.rrr	RPICSNET1	RPICS-LOCALNET-1	[MS9]
R*192.012.093.rrr	RPICSNET2	RPICS-LOCALNET-2	[MS9]
R*192.012.094.rrr	RPICSNET3	RPICS-LOCALNET-3	[MS9]
R*192.012.095.rrr	RPICSNET4	RPICS-LOCALNET-4	[MS9]
R*192.012.096.rrr	RPICSNET5	RPICS-LOCALNET-5	[MS9]
R*192.012.097.rrr	RPICSNET6	RPICS-LOCALNET-6	[MS9]
R*192.012.098.rrr	RPICSNET7	RPICS-LOCALNET-7	[MS9]
R*192.012.099.rrr	RPICSNET8	RPICS-LOCALNET-8	[MS9]
R*192.012.100.rrr	RPICSNET9	RPICS-LOCALNET-9	[MS9]
R*192.012.101.rrr	OSU-CGRG	OSU Computer Graphics	[128,KXS]
G 192.012.102.rrr	AMES-NAS-HY	AMES NAS HY NET	[MF31]
R*192.012.103 rrr-192.012.118.rrr		Colorado State Univ Nets	[RXB1]
G 192.012.119.rrr	ICST	ICST Network	[128,J[illegible]]

D	192.012.120.rrr	MITRE-B-NET	MITRE BEDFORD ETHER	[BSW]
R*	192.012.121.rrr	FSUCS	FSU COMPUTER SCIENCE 1	[TXB]
R*	192.012.122.rrr	FSUCS2	FSU COMPUTER SCIENCE 2	[TXB]
G	192.012.123.rrr	AMES-CCF-NET	AMES CCF NETWORK	[128,MSM1]
D	192.012.124.rrr	ETL-LAN	ETL LOCAL AREA NET	[128,WWS]
D	192.012.125.rrr	CRDC-NET1	CRDC-NET1	[128,JXY]
D	192.012.126.rrr	CRDC-NET2	CRDC-NET2	[128,JXY]
R	192.012.127.rrr	LL-MI-NET	LL-Machine Intell.	[128,GAA]
R	192.012.128.rrr	AITAC-ADMIN	SRI-AITAC ADMIN NET	[128,DVC]
C*	192.012.129.rrr	SYM-CAN	Symbolics/Canada	[MXH]
R	192.012.130.rrr	SDC-SM	SDC Santa Monica	[CAS]
R	192.012.131.rrr	SAC-ADMIN	SRI-SAC ADMIN NET	[128,KMC3]
R	192.012.132.rrr	LLL-MON	LLL Open Labnet-1	[128,BANDY]
R	192.012.133.rrr	LLL-TUES	LLL Open Labnet-2	[128,BANDY]
R	192.012.134.rrr	LLL-WED	LLL Open Labnet-3	[128,BANDY]
R	192.012.135.rrr	LLL-THU	LLL Open Labnet-4	[128,BANDY]
R	192.012.136.rrr	LLL-FRI	LLL Open Labnet-5	[128,BANDY]
R	192.012.137.rrr	LLL-SAT	LLL Open Labnet-6	[128,BANDY]
R	192.012.138.rrr	LLL-SUN	LLL Open Labnet-7	[128,BANDY]
D	192.012.139.rrr	JTELS-BEN-GW	JUMPS Teleprocessing	[RR26]
R*	192.012.140.rrr	INFERENCE	INFERENCE	[DXT]
R	192.012.141.rrr	CSS-ETHER	CSS Workstation Net 2	[RA11]
C*	192.012.142.rrr	SENTRY	Sentry Adv. Prod. Net	[LXL]
C*	192.012.143.rrr	VHSIC-NET	Sentry VHSIC Test	[LXL]
R*	192.012.144.rrr	ECRCNET	ECRC Internet	[128,PXD]
C*	192.012.145 rrr-192.012.154.rrr		RCA-CADNET	[128,RXG]
C*	192.012.155 rrr-192.012.170.rrr		MTCS-CUST	[SXF]
D	192.012.171.rrr	PICANET2	Picatinny Arsenal 2	[RFD1]
R	192.012.172.rrr	ROCKWELLENET	ROCKWELL ETHERNET	[NG]
D	192.012.173.rrr	JTELS-BEN1-GW	JUMPS Teleprocessing	[RR26]
R*	192.012.174 rrr-192.012.183.rrr		TORONTO	[128,BXD]
	192.012.184 rrr-192.012.255.rrr		Unassigned	[JBP]
D	192.013.000.rrr-192.014.255.rrr		DODIIS Subnetworks	[AY5]
C*	192.015.000.rrr-192.015.255.rrr		NBINET	[WW2]
G	192.016.000.rrr-192.016.049.rrr		LANLLAN	[128,JC11]
	192.016.050.rrr-192.016.255.rrr		Unassigned	[JBP]
R*	192.017.000.rrr-192.017.255.rrr		NIBELUNG	[MXA]
C*	192.018.000.rrr-192.018.255.rrr		SUN Microsystems, Inc.	[BN4]
C*	192.019.000.rrr-192.019.255.rrr		SYSNET-2	[EXY]
C*	192.020.000.rrr-192.020.255.rrr		ATT-MD-NET	[128,MH12]
	192.021.000.rrr-223.255.254.rrr		Unassigned	[JBP]
	223.255.255.rrr		Reserved	[JBP]

Other Reserved Internet Addresses

*	Internet Address	Name	Network	References
	224.000.000.000-255.255.255.255	Reserved		[JBP]

Network Totals

Assigned for the ARPA-Internet and the DDN-Internet

Class	A	B	C	Total
Research	7	63	911	981
Defense	8	15	536	559
Government	0	2	59	61
Commercial	2	1	4	7
Total	17	81	1510	1608

Allocated for Internet and Independent Uses

Class	A	B	C	Total
Research	7	68	1764	1838
Defense	8	15	536	559
Government	1	3	64	68
Commercial	2	5	2357	2364
Total	18	91	4721	4829

Maximum Allowed

Class	A	B	C	Total
Research	8	1024	65536	66568
Defense	24	3072	458752	461848
Government	24	3072	458752	461848
Commercial	74	9214	1114137	1123394
Total	126	16382	2097150	2113658

ASSIGNED VERSION NUMBERS

In the Internet Protocol (IP) [39,92] there is a field to identify the version of the internetwork general protocol. This field is 4 bits in size.

Assigned Internet Version Numbers

Decimal	Keyword	Version	References
0		Reserved	[JBP]
1-3		Unassigned	[JBP]
4	IP	Internet Protocol	[37,85,JBP]
5	ST	ST Datagram Mode	[40,JWF]
6-14		Unassigned	[JBP]
15		Reserved	[JBP]

ASSIGNED PROTOCOL NUMBERS

In the Internet Protocol (IP) [39,92] there is a field, called Protocol, to identify the the next level protocol. This is an 8 bit field.

Assigned Internet Protocol Numbers

Decimal	Keyword	Protocol	References
0		Reserved	[JBP]
1	ICMP	Internet Control Message	[84,JBP]
2		Unassigned	[JBP]
3	GGP	Gateway-to-Gateway	[51,MB]
4		Unassigned	[JBP]
5	ST	Stream	[43,JWF]
6	TCP	Transmission Control	[39,93,JBP]
7	UCL	UCL	[PK]
8	EGP	Exterior Gateway Protocol	[108,DLM1]
9	IGP	any private interior gateway	[JBP]
10	BBN-RCC-MON	BBN RCC Monitoring	[SGC]
11	NVP-II	Network Voice Protocol	[21,SC3]
12	PUP	PUP	[15,HGM]
13	ARGUS	ARGUS	[RWS4]
14	EMCON	EMCON	[BN7]
15	XNET	Cross Net Debugger	[49,JFH2]
16	CHAOS	Chaos	[NC3]
17	UDP	User Datagram	[39,91,JBP]
18	MUX	Multiplexing	[22,JBP]
19	DCN-MEAS	DCN Measurement Subsystems	[DLM1]
20	HMP	Host Monitoring	[6,RH6]
21	PRM	Packet Radio Measurement	[ZSU]
22	XNS-IDP	XEROX NS IDP	[129,LLG]
23	TRUNK-1	Trunk-1	[SA2]
24	TRUNK-2	Trunk-2	[SA2]
25	LEAF-1	Leaf-1	[SA2]
26	LEAF-2	Leaf-2	[SA2]
27	RDP	Reliable Data Protocol	[125,RH6]
28	IRTP	Internet Reliable Transaction	[68,TXM]
29	ISO-TP4	ISO Transport Protocol Class 4	[57,RC7]
30-60		Unassigned	[JBP]
61		any host internal protocol	[JBP]
62	CFTP	CFTP	[44,HCF2]
63		any local network	[JBP]
64	SAT-EXPAK	SATNET and Backroom EXPAK	[SHB]
65	MIT-SUBNET	MIT Subnet Support	[NC3]
66	RVD	MIT Remote Virtual Disk Protocol	[MBG]
67	IPPC	Internet Pluribus Packet Core	[SHB]

68		any distributed file system	[JBP]
69	SAT-MON	SATNET Monitoring	[SHB]
70		Unassigned	[JBP]
71	IPCV	Internet Packet Core Utility	[SHB]
72-75		Unassigned	[JBP]
76	BR-SAT-MON	Backroom SATNET Monitoring	[SHB]
77		Unassigned	[JBP]
78	WB-MON	WIDEBAND Monitoring	[SHB]
79	WB-EXPAK	WIDEBAND EXPAK	[SHB]
80-254		Unassigned	[JBP]
255		Reserved	[JBP]

ASSIGNED PORT NUMBERS

Ports are used in the TCP [39,93] to name the ends of logical connections which carry long term conversations. For the purpose of providing services to unknown callers, a service contact port is defined. This list specifies the port used by the server process as its contact port. The contact port is sometimes called the "well-known port".

To the extent possible, these same port assignments are used with the UDP [39,91].

To the extent possible, these same port assignments are used with the ISO-TP4 [57].

The assigned ports use a small portion of the possible port numbers. The assigned ports have all except the low order eight bits cleared to zero. The low order eight bits are specified here.

Port Assignments:

Decimal	Keyword	Description	References
0		Reserved	[JBP]
1-4		Unassigned	[JBP]
5	RJE	Remote Job Entry	[17,40,JBP]
7	ECHO	Echo	[82,JBP]
9	DISCARD	Discard	[80,JBP]
11	USERS	Active Users	[76,JBP]
13	DAYTIME	Daytime	[79,JBP]
15	NETSTAT	Who is up or NETSTAT	[JBP]
17	QUOTE	Quote of the Day	[87,JBP]
19	CHARGEN	Character Generator	[78,JBP]
20	FTP-DATA	File Transfer [Default Data]	[39,83,JBP]
21	FTP	File Transfer [Control]	[39,83,JBP]
23	TELNET	Telnet	[99,JBP]
25	SMTP	Simple Mail Transfer	[39,89,JBP]
27	NSW-FE	NSW User System FE	[23,RHT]
29	MSG-ICP	MSG ICP	[74,RHT]
31	MSG-AUTH	MSG Authentication	[74,RHT]
33	DSP	Display Support Protocol	[MLC]
35		any private printer server	[JBP]
37	TIME	Time	[95,JBP]
39	RLP	Resource Location Protocol	[1,MA]
41	GRAPHICS	Graphics	[40,115,JBP]
42	NAMESERVER	Host Name Server	[39,86,JBP]
43	NICNAME	Who Is	[39,48,JAKE]
44	MPM-FLAGS	MPM FLAGS Protocol	[JBP]

45	MPM	Message Processing Module [recv]	[85,JBP]
46	MPM-SND	MPM [default send]	[91,JBP]
47	NI-FTP	NI FTP	[122,SK]
49	LOGIN	Login Host Protocol	[PHD1]
51	LA-MAINT	IMP Logical Address Maintenance	[66,AGM]
53	DOMAIN	Domain Name Server	[81,71,PM1]
55	ISI-GL	ISI Graphics Language	[14,RB6]
57		any private terminal access	[JBP]
59		any private file service	[JBP]
61	NI-MAIL	NI MAIL	[12,SK]
63	VIA-FTP	VIA Systems - FTP	[DXD]
65	TACACS-DS	TACACS-Database Service	[11,RHT]
67	BOOTPS	Bootstrap Protocol Server	[35,WJC2]
68	BOOTPC	Bootstrap Protocol Client	[35,WJC2]
69	TFTP	Trivial File Transfer	[39,102,DDC1]
71	NETRJS-1	Remote Job Service	[16,40,RTB]
72	NETRJS-2	Remote Job Service	[16,40,RTB]
73	NETRJS-3	Remote Job Service	[16,40,RTB]
74	NETRJS-4	Remote Job Service	[16,40,RTB]
75		any private dial out service	[JBP]
77		any private RJE service	[JBP]
79	FINGER	Finger	[40,46,KLH]
81	HOSTS2-NS	HOSTS2 Name Server	[EAK1]
83	MIT-ML-DEV	MIT ML Device	[DPR]
85	MIT-ML-DEV	MIT ML Device	[DPR]
87		any private terminal link	[JBP]
89	SU-MIT-TG	SU/MIT Telnet Gateway	[MRC]
91	MIT-DOV	MIT Dover Spooler	[EBM]
93	DCP	Device Control Protocol	[DT15]
95	SUPDUP	SUPDUP	[26,MRC]
97	SWIFT-RVF	Swift Remote Vitural File Protocol	[MXR]
98	TACNEWS	TAC News	[FRAN]
99	METAGRAM	Metagram Relay	[GEOF]
101	HOSTNAME	NIC Host Name Server	[39,47,JAKE]
103		Unassigned	[JBP]
105	CSNET-NS	Mailbox Name Nameserver	[113,MHS1]
107	RTELNET	Remote Telnet Service	[88,JBP]
109	POP-2	Post Office Protocol - Version 2	[19,JKR1]
111	SUNRPC	SUN Remote Procedure Call	[DXG]
113	AUTH	Authentication Service	[116,MCSJ]
115	SFTP	Simple File Transfer Protocol	[60,MKL1]
117	UUCP-PATH	UUCP Path Service	[38,MAE]
119	UNTP	USENET News Transfer Protocol	[61,PL4]
121	ERPC	HYDRA Expedited Remote Procedure Call	[118,JXO]
123	NTP	Network Time Protocol	[70,DLM1]
125	LOCUS-MAP	Locus PC-Interface Net Map Server	[124,BXG]
127	LOCUS-CON	Locus PC-Interface Conn Server	[124,BXG]
129		Unassigned	[JBP]

131		Unassigned	[JBP]
133-223		Reserved	[JBP]
224-241		Unassigned	[JBP]
243	SUR-MEAS	Survey Measurement	[13,AV]
245	LINK	LINK	[18,RDB2]
247-255		Unassigned	[JBP]

ASSIGNED AUTONOMOUS SYSTEM NUMBERS

The Exterior Gateway Protocol (EGP) [108,105] specifies that groups of gateways may form autonomous systems. The EGP provides a 16-bit field for identifying such systems. The values of this field are registered here.

Autonomous System Numbers:

Decimal	Name	References
0	Reserved	[JBP]
1	The BBN Core Gateways	[MB]
2	DCN-AS	[DLM1]
3	The MIT Gateways	[LM8]
4	ISI-AS	[JKR1]
5	Symbolics	[CH2]
6	HIS-Multics	[BIM,JLM23]
7	UK-MOD	[RNM1]
8	RICE-AS	[PGM]
9	CMU-ROUTER	[MA]
10	CSNET-PDN-AS	[RDR4]
11	HARVARD	[SB28]
12	NYU-DOMAIN	[EF5]
13	BRL-AS	[RBN1]
14	COLUMBIA-GW	[BC14]
15	NET DYNAMICS EXP	[ZSU]
16	LBL	[WG]
17	PURDUE-CS	[KCS1]
18	UTEXAS	[JSQ1]
19	CSS-DOMAIN	[RR2]
20	UR	[LB16]
21	RAND	[JDG]
22	NOSC	[RLB3]
23	RIACS-AS	[DG28]
24	AMES-NAS-GW	[MF31]
25	UCB	[MK17]
26	CORNELL	[BN9]
27	UMDNET	[JWO1]
28	DFVLR-SYS	[HDC1]
29	YALE-AS	[JG46]
30	SRI-AICNET	[PM4]
31	CIT-CS	[AD22]
32	STANFORD	[PA5]
33	DEC-WRL-AS	[RKJ2]
34	UDEL-EECIS	[NMM]
35	MICATON	[WDL]
36	EGP-TESTOR	[BP17]

Autonomous System Numbers

37	NSWC	[MXP1]
38	UIUC	[AKC]
39	NRL-ITD	[AP]
40	MIT-TEST	[NC3]
41	AMES	[MSM1]
42	THINK-AS	[BJN1]
43	BNL-AS	[GC]
44	S1-DOMAIN	[LWR]
45	LLL-TIS-AS	[GP10]
46	RUTGERS	[RM8]
47	USC-OBERON	[DRS4]
48	NRL-AS	[WF3]
49	ICST-AS	[JCN2]
50	ORNL-MSRNET	[THD]
51	USAREUR-EM-AS	[WXD]
52	UCLA	[BXL]
53-65534	Unassigned	[JBP]
65535	Reserved	[JBP]

DOMAIN SYSTEM PARAMETERS

The Internet Domain Naming System (DOMAIN) includes several parameters. These are documented in RFC 883 [72]. The CLASS parameter is listed here. The per CLASS parameters are defined in separate RFCs as indicated.

Domain System Parameters:

Decimal	Name	References
0	Reserved	[PM1]
1	Internet	[72,PM1]
2	Unassigned	[PM1]
3	Chaos	[PM1]
4-65534	Unassigned	[PM1]
65535	Reserved	[PM1]

ASSIGNED ARPANET LOGICAL ADDRESSES

The ARPANET facility for "logical addressing" is described in RFC 878 [65]. A portion of the possible logical addresses are reserved for standard uses.

There are 49,152 possible logical host addresses. Of these, 256 are reserved for assignment to well-known functions. Assignments for well-known functions are made by Joyce Reynolds. Assignments for other logical host addresses are made by the NIC.

Logical Address Assignments:

Decimal	Description	References
0	Reserved	[JBP]
1	The BBN Core Gateways	[MB]
2-255	Unassigned	[JBP]
256	Reserved	[JBP]

ASSIGNED ARPANET LINK NUMBERS

The word "link" here refers to a field in the original ARPANET Host/IMP interface leader. The link was originally defined as an 8-bit field. Later specifications defined this field as the "message-id" with a length of 12 bits. The name link now refers to the high order 8 bits of this 12-bit message-id field. The Host/IMP interface is defined in BBN Report 1822 [10].

The low-order 4 bits of the message-id field are called the sub-link. Unless explicitly specified otherwise for a particular protocol, there is no sender to receiver significance to the sub-link. The sender may use the sub-link in any way he chooses (it is returned in the RFNM by the destination IMP), the receiver should ignore the sub-link.

Link Assignments:

Decimal	Description	References
0	Reserved	[JBP]
1-149	Unassigned	[JBP]
150	Xerox NS IDP	[129,LLG]
151	Unassigned	[JBP]
152	PARC Universal Protocol	[15,HGM]
153	TIP Status Reporting	[JGH]
154	TIP Accounting	[JGH]
155	Internet Protocol [regular]	[39,92,JBP]
156-158	Internet Protocol [experimental]	[39,92,JBP]
159	Figleaf Link	[JBW1]
160-194	Unassigned	[JBP]
195	ISO-IP	[58,RXM]
196-247	Experimental Protocols	[JBP]
248-255	Network Maintenance	[JGH]

IEEE 802 SAP NUMBERS OF INTEREST

Some of the networks of all classes are IEEE 802 Networks. These systems may use a Service Access Point field in much the same way the ARPANET uses the "link" field. For further information and SAP number assignments, please contact: Mr. Maris Graube, Chairman, IEEE 802, Route 1, 244 H, Forest Grove, Oregon, 97116.

Assignments:

Service Access Point		Description	References
decimal	binary		
127	01111111	ISO DIS 8473	[JXJ]
96	01100000	DOD IP	[39,91,JBP]

The IEEE 802.3 header does not have a type field to indicate what protocol is used at the next level. As a work around for this problem, one can put the Ethernet type field value in the IEEE 802.3 header's length field and use the following test to determine the appropriate processing on receipt.

If the value in the length field of the IEEE 802.3 header is greater than the Ethernet maximum packet length, then interpret the value as an Ethernet type field. Otherwise, interpret the packet as an IEEE 802.3 packet.

The proposed standard for transmission of IP datagrams over IEEE 802.3 networks is specified in RFC 948 [127].

ETHERNET NUMBERS OF INTEREST

Many of the networks of all classes are Ethernets (10Mb) or Experimental Ethernets (3Mb). These systems use a message "type" field in much the same way the ARPANET uses the "link" field.

If you need an Ethernet number, contact the XEROX Corporation, Office Products Division, Network Systems Administration Office, 333 Coyote Hill Road, Palo Alto, California, 94304.

Assignments:

Ethernet		Exp. Ethernet		Description	References
decimal	Hex	decimal	octal		
512	0200	512	1000	XEROX PUP	[1,HGM]
513	0201	-	-	PUP Addr. Trans.	[HGM]
1536	0600	1536	3000	XEROX NS IDP	[128,HGM]
2048	0800	513	1001	DOD IP	[39,91,JBP]
2049	0801	-	-	X.75 Internet	[HGM]
2050	0802	-	-	NBS Internet	[HGM]
2051	0803	-	-	ECMA Internet	[HGM]
2052	0804	-	-	Chaosnet	[HGM]
2053	0805	-	-	X.25 Level 3	[HGM]
2054	0806	-	-	ARP	[74,JBP]
2055	0807	-	-	XNS Compatability	[HGM]
2076	081C	-	-	Symbolics Private	[DCP1]
32771	8003	-	-	Cronus VLN	[116,DT15]
32772	8004	-	-	Cronus Direct	[116,DT15]
32774	8006	-	-	Nestar	[HGM]
32784	8010	-	-	Excelan	[HGM]
32821	8035	-	-	Reverse ARP	[42,JCM]
36864	9000	-	-	Loopback	[HGM]

The standard for transmission of IP datagrams over Ethernets and Experimental Ethernets is specified in RFC 894 [54] and RFC 895 [76] respectively.

ASSIGNED ADDRESS RESOLUTION PROTOCOL PARAMETERS

The Address Resolution Protocol (ARP) specified in RFC 826 [75] has several parameters. The assigned values for these parameters are listed here.

Assignments:

Operation Code (op)

1	REQUEST
2	REPLY

Hardware Type (hrd)

Type	Description	References
1	Ethernet (10Mb)	[JBP]
2	Experimental Ethernet (3Mb)	[JBP]
3	Amateur Radio AX.25	[PXK]
4	Proton ProNET Token Ring	[JBP]
5	Chaos	[GXP]

Protocol Type (pro)

Use the same codes as listed in the section called "Ethernet Numbers of Interest" (all hardware types use this code set for the protocol type).

ASSIGNED PUBLIC DATA NETWORK NUMBERS

One of the Internet Class A Networks is the international system of Public Data Networks. This section lists the mapping between the Internet Addresses and the Public Data Network Addresses (X.121).

Assignments:

Internet	Public Data Net	Description	References
014.000.000.000		Reserved	[JBP]
014.000.000.001	3110-317-00035 00	PURDUE-TN	[CAK]
014.000.000.002	3110-608-00027 00	UWISC-TN	[CAK]
014.000.000.003	3110-302-00024 00	UDEL-TN	[CAK]
014.000.000.004	2342-192-00149 23	UCL-VTEST	[PK]
014.000.000.005	2342-192-00300 23	UCL-TG	[PK]
014.000.000.006	2342-192-00300 25	UK-SATNET	[PK]
014.000.000.007	3110-608-00024 00	UWISC-IBM	[MHS1]
014.000.000.008	3110-213-00045 00	RAND-TN	[MO2]
014.000.000.009	2342-192-00300 23	UCL-CS	[PK]
014.000.000.010	3110-617-00025 00	BBN-VAN-GW	[JD21]
014.000.000.011	2405-015-50300 00	CHALMERS	[UXB]
014.000.000.012	3110-713-00165 00	RICE	[PAM6]
014.000.000.013	3110-415-00261 00	DECWRL	[PAM6]
014.000.000.014	3110-408-00051 00	IBM-SJ	[SA1]
014.000.000.015	2041-117-01000 00	SHAPE	[JFW]
014.000.000.016	2628-153-90075 00	DFVLR4-X25	[HDC1]
014.000.000.017	3110-213-00032 00	ISI-VAN-GW	[JD21]
014.000.000.018	2624-522-80900 52	DFVLR5-X25	[HDC1]
014.000.000.019	2041-170-10000 00	SHAPE-X25	[JFW]
014.000.000.020	5052-737-20000 50	UQNET	[AXH]
014.000.000.021	3020-801-00057 50	DMC-CRC1	[JR17]
014.000.000.022-014.255.255.254		Unassigned	[JBP]
014.255.255.255		Reserved	[JBP]

The standard for transmission of IP datagrams over the Public Data Network is specified in RFC 877 [60].

ASSIGNED TELNET OPTIONS

The Telnet Protocol has a number of options that may be negotiated. These options are listed here. "Official ARPA-Internet Protocols" [104] provides more detailed information.

Options	Name	References
0	Binary Transmission	[97,JBP]
1	Echo	[98,JBP]
2	Reconnection	[7,JBP]
3	Suppress Go Ahead	[101,JBP]
4	Approx Message Size Negotiation	[40,JBP]
5	Status	[100,JBP]
6	Timing Mark	[102,JBP]
7	Remote Controlled Trans and Echo	[94,JBP]
8	Output Line Width	[5,JBP]
9	Output Page Size	[6,JBP]
10	Output Carriage-Return Disposition	[27,JBP]
11	Output Horizontal Tab Stops	[31,JBP]
12	Output Horizontal Tab Disposition	[30,JBP]
13	Output Formfeed Disposition	[28,JBP]
14	Output Vertical Tabstops	[33,JBP]
15	Output Vertical Tab Disposition	[32,JBP]
16	Output Linefeed Disposition	[29,JBP]
17	Extended ASCII	[123,JBP]
18	Logout	[24,MRC]
19	Byte Macro	[34,JBP]
20	Data Entry Terminal	[37,JBP]
22	SUPDUP	[26,25,MRC]
22	SUPDUP Output	[45,MRC]
23	Send Location	[59,EAK1]
24	Terminal Type	[114,MHS1]
25	End of Record	[89,JBP]
26	TACACS User Identification	[3,BA4]
27	Output Marking	[110,SXS]
28	Terminal Location Number	[73,RN6]
255	Extended-Options-List	[96,JBP]

OFFICIAL MACHINE NAMES

These are the Official Machine Names as they appear in the NIC Host Table. Their use is described in RFC 810 [41].

ALTO
AMDAHL-V7
APOLLO
ATT-3B20
BBN-C/60
BURROUGHS-B/29
BURROUGHS-B/4800
BUTTERFLY
C/30
C/70
CADLINC
CADR
CDC-170
CDC-170/750
CDC-173
CELERITY-1200
COMTEN-3690
CP8040
CTIWS-117
DANDELION
DEC-10
DEC-1050
DEC-1077
DEC-1080
DEC-1090
DEC-1090B
DEC-1090T
DEC-2020T
DEC-2040
DEC-2040T
DEC-2050T
DEC-2060
DEC-2060T
DEC-2065
DEC-FALCON
DEC-KS10
DORADO
DPS8/70M
ELXSI-6400
FOONLY-F2
FOONLY-F3
FOONLY-F4
GOULD

```
   GOULD-6050
   GOULD-6080
   GOULD-9050
   GOULD-9080
   H-316
   H-60/68
   H-68
   H-68/80
   H-89
   HONEYWELL-DPS-6
   HONEYWELL-DPS-8/70
   HP3000
   HP3000/64
   IBM-158
   IBM-360/67
   IBM-370/3033
   IBM-3081
   IBM-3084QX
   IBM-3101
   IBM-4331
   IBM-4341
   IBM-4361
   IBM-4381
   IBM-4956
   IBM-PC
   IBM-PC/AT
   IBM-PC/XT
   IBM-SERIES/1
   IMAGEN
   IMAGEN-8/300
   IMSAI
   INTEGRATED-SOLUTIONS
   INTEGRATED-SOLUTIONS-68K
   INTEGRATED-SOLUTIONS-CREATOR
   INTEGRATED-SOLUTIONS-CREATOR-8
   INTEL-IPSC
   IRIS
   IRIS-1400
   IS-1
   IS-68010
   LMI
   LSI-11
   LSI-11/2
   LSI-11/23
   LSI-11/73
   M-6800
   M68000
   MASSCOMP
```

```
MC500
MC68000
MICROVAX
MICROVAX-I
MV/8000
NAS3-5
NCR-COMTEN-3690
NOW
ONYX-Z8000
PDP-11
PDP-11/3
PDP-11/23
PDP-11/24
PDP-11/34
PDP-11/40
PDP-11/44
PDP-11/45
PDP-11/50
PDP-11/70
PDP-11/73
PE-7/32
PE-3205
PERQ
PLEXUS-P/60
PLI
PLURIBUS
PYRAMID-90
PYRAMID-90MX
PYRAMID-90X
RIDGE
RIDGE-32
RIDGE-32C
ROLM-1666
S1-MKIIA
SMI
SEQUENT
SEQUENT-BALANCE-8000
SGI-IRIS
SIEMENS
SILICON-GRAPHICS
SILICON-GRAPHICS-IRIS
SPERRY-DCP/10
SUN
SUN-2
SUN-2/50
SUN-2/100
SUN-2/120
SUN-2/140
```

```
   SUN-2/150
   SUN-2/160
   SUN-2/170
   SUN-3/160
   SUN-3/75
   SUN-50
   SUN-100
   SUN-120
   SUN-130
   SUN-150
   SUN-170
   SUN-68000
   SYMBOLICS-3600
   SYMBOLICS-3670
   TANDEM-TXP
   TEK-6130
   TI-EXPLORER
   TP-4000
   TRS-80
   UNIVAC-1100
   UNIVAC-1100/60
   UNIVAC-1100/62
   UNIVAC-1100/63
   UNIVAC-1100/64
   UNIVAC-1100/70
   UNIVAC-1160
   VAX-11/725
   VAX-11/730
   VAX-11/750
   VAX-11/780
   VAX-11/785
   VAX-11/790
   VAX-11/8600
   VAX-8600
   WANG-PC002
   WANG-VS100
   WANG-VS400
   XEROX-1100
   XEROX-1108
   XEROX-8010
```

OFFICIAL SYSTEM NAMES

These are the Official System Names as they appear in the NIC Host Table. Their use is described in RFC 810 [41].

AEGIS
APOLLO
BS-2000
CEDAR
CGW
CHRYSALIS
CMOS
CMS
COS
CPIX
CTOS
DCN
DDNOS
DOMAIN
EDX
ELF
EMBOS
EMMOS
EPOS
FOONEX
FUZZ
GCOS
GPOS
HDOS
IMAGEN
INTERCOM
IMPRESS
INTERLISP
IOS
ITS
LISP
LISPM
LOCUS
MINOS
MOS
MPE5
MSDOS
MULTICS
MVS
MVS/SP
NEXUS
NMS
NONSTOP

```
   NOS-2
   OS/DDP
   OS4
   OS86
   OSX
   PCDOS
   PERQ-OS
   PLI
   PSDOS/MIT
   RMX/RDOS
   ROS
   RSX11M
   SATOPS
   SCS
   SIMP
   SWIFT
   TAC
   TANDEM
   TENEX
   TOPS-10
   TOPS-20
   TP3010
   TRSDOS
   ULTRIX
   UNIX
   UT2D
   V
   VM
   VM/370
   VM/CMS
   VM/SP
   VMS
   VMS/EUNICE
   VRTX
   WAITS
   WANG
   XDE
   XENIX
```

OFFICIAL PROTOCOL AND SERVICE NAMES

These are the Official Protocol Names. Their use is described in greater detail in RFC 810 [41].

ARGUS	- ARGUS Protocol
AUTH	- Authentication Service
BBN-RCC-MON	- BBN RCC Monitoring
BOOTPC	- Bootstrap Protocol Client
BOOTPS	- Bootstrap Protocol Server
BR-SAT-MON	- Backroom SATNET Monitoring
CFTP	- CFTP
CHAOS	- CHAOS Protocol
CHARGEN	- Character Generator Protocol
CLOCK	- DCNET Time Server Protocol
CSNET-NS	- CSNET Mailbox Nameserver Protocol
DAYTIME	- Daytime Protocol
DCN-MEAS	- DCN Measurement Subsystems Protocol
DCP	- Device Control Protocol
DISCARD	- Discard Protocol
DOMAIN	- Domain Name Server
ECHO	- Echo Protocol
EGP	- Exterior Gateway Protocol
EMCON	- Emission Control Protocol
FINGER	- Finger Protocol
FTP	- File Transfer Protocol
GGP	- Gateway Gateway Protocol
GRAPHICS	- Graphics Protocol
HMP	- Host Monitoring Protocol
HOST2-NS	- Host2 Name Server
HOSTNAME	- Hostname Protocol
ICMP	- Internet Control Message Protocol
IGP	- Interior Gateway Protocol
IP	- Internet Protocol
IPCU	- Internet Packet Core Utility
IPPC	- Internet Pluribus Packet Core
IRTP	- Internet Reliable Transaction Protocol
ISI-GL	- ISI Graphics Language Protocol
ISO-TP4	- ISO Transport Protocol Class 4
LA-MAINT	- IMP Logical Address Maintenance
LEAF-1	- Leaf-1 Protocol
LEAF-2	- Leaf-2 Protocol
LINK	- Link Protocol
LOGIN	- Login Host Protocol
METAGRAM	- Metagram Relay
MIT-ML-DEV	- MIT ML Device
MIT-SUBNET	- MIT Subnet Support
MIT-DOV	- MIT Dover Spooler

MPM	- Internet Message Protocol (Multimedia Mail)
MPM-FLAGS	- MP Flags Protocol
MSG-AUTH	- MSG Authentication Protocol
MSG-ICP	- MSG ICP Protocol
MUX	- Multiplexing Protocol
NAMESERVER	- Host Name Server
NETED	- Network Standard Text Editor
NETRJS	- Remote Job Service
NI-FTP	- NI File Transfer Protocol
NI-MAIL	- NI Mail Protocol
NICNAME	- Who Is Protocol
NSW-FE	- NSW User System Front End
NTP	- Network Time Protocol
NVP-II	- Network Voice Protocol
POP2	- Post Office Protocol - Version 2
PRM	- Packet Radio Measurement
PUP	- PUP Protocol
QUOTE	- Quote of the Day Protocol
RDP	- Reliable Data Protocol
RJE	- Remote Job Entry
RLP	- Resource Location Protocol
RTELNET	- Remote Telnet Service
RVD	- Remote Virtual Disk Protocol
SAT-EXPAK	- Satnet and Backroom EXPAK
SAT-MON	- SATNET Monitoring
SFTP	- Simple File Transfer Protocol
SMTP	- Simple Mail Transfer Protocol
ST	- Stream Protocol
SU-MIT-TG	- SU/MIT Telnet Gateway Protocol
SUNRPC	- SUN Remote Procedure Call
SUPDUP	- SUPDUP Protocol
SUR-MEAS	- Survey Measurement
SWIFT-RVF	- Remote Virtual File Protocol
TACACS-DS	- TACACS-Database Service
TACNEWS	- TAC News
TCP	- Transmission Control Protocol
TELNET	- Telnet Protocol
TFTP	- Trivial File Transfer Protocol
TIME	- Time Server Protocol
TRUNK-1	- Trunk-1 Protocol
TRUNK-2	- Trunk-2 Protocol
UCL	- University College London Protocol
UDP	- User Datagram Protocol
UNTP	- USENET News Transfer Protocol
USERS	- Active Users Protocol
UUCP-PATH	- UUCP Path Service
VIA-FTP	- VIA Systems-File Transfer Protocol
WB-EXPAK	- Wideband EXPAK

WB-MON	- Wideband Monitoring
XNET	- Cross Net Debugger
XNS-IDP	- Xerox NS IDP

OFFICIAL TERMINAL TYPE NAMES

These are the Official Terminal Type Names. Their use is described in RFC 930 [114]. The maximum length of a name is 40 characters.

```
ADDS-CONSUL-980
ADDS-REGENT-100
ADDS-REGENT-20
ADDS-REGENT-200
ADDS-REGENT-25
ADDS-REGENT-40
ADDS-REGENT-60
AMPEX-DIALOGUE-80
ANDERSON-JACOBSON-630
ANDERSON-JACOBSON-832
ANDERSON-JACOBSON-841
ANN-ARBOR-AMBASSADOR
ARDS
BITGRAPH
BUSSIPLEXER
CALCOMP-565
CDC-456
CDI-1030
CDI-1203
CLNZ
COMPUCOLOR-II
CONCEPT-100
CONCEPT-104
CONCEPT-108
DATA-100
DATA-GENERAL-6053
DATAGRAPHIX-132A
DATAMEDIA-1520
DATAMEDIA-1521
DATAMEDIA-2500
DATAMEDIA-3025
DATAMEDIA-3025A
DATAMEDIA-3045
DATAMEDIA-3045A
DATAMEDIA-DT80/1
DATAPOINT-2200
DATAPOINT-3000
DATAPOINT-3300
DATAPOINT-3360
DEC-DECWRITER-I
DEC-DECWRITER-II
DEC-GT40
DEC-GT40A
```

DEC-GT42
DEC-LA120
DEC-LA30
DEC-LA36
DEC-LA38
DEC-VT05
DEC-VT100
DEC-VT132
DEC-VT50
DEC-VT50H
DEC-VT52
DELTA-DATA-5000
DELTA-TELTERM-2
DIABLO-1620
DIABLO-1640
DIGILOG-333
DTC-300S
EDT-1200
EXECUPORT-4000
EXECUPORT-4080
GENERAL-TERMINAL-100A
GSI
HAZELTINE-1500
HAZELTINE-1510
HAZELTINE-1520
HAZELTINE-2000
HP-2621
HP-2621A
HP-2621P
HP-2626
HP-2626A
HP-2626P
HP-2640
HP-2640A
HP-2640B
HP-2645
HP-2645A
HP-2648
HP-2648A
HP-2649
HP-2649A
IBM-3101
IBM-3101-10
IBM-3275-2
IBM-3276-2
IBM-3276-3
IBM-3276-4
IBM-3277-2

```
   IBM-3278-2
   IBM-3278-3
   IBM-3278-4
   IBM-3278-5
   IBM-3279-2
   IBM-3279-3
   IMLAC
   INFOTON-100
   INFOTONKAS
   ISC-8001
   LSI-ADM-3
   LSI-ADM-31
   LSI-ADM-3A
   LSI-ADM-42
   MEMOREX-1240
   MICROBEE
   MICROTERM-ACT-IV
   MICROTERM-ACT-V
   MICROTERM-MIME-1
   MICROTERM-MIME-2
   NETRONICS
   NETWORK-VIRTUAL-TERMINAL
   OMRON-8025AG
   PERKIN-ELMER-1100
   PERKIN-ELMER-1200
   PERQ
   PLASMA-PANEL
   QUME-SPRINT-5
   SOROC
   SOROC-120
   SOUTHWEST-TECHNICAL-PRODUCTS-CT82
   SUPERBEE
   SUPERBEE-III-M
   TEC
   TEKTRONIX-4010
   TEKTRONIX-4012
   TEKTRONIX-4013
   TEKTRONIX-4014
   TEKTRONIX-4023
   TEKTRONIX-4024
   TEKTRONIX-4025
   TEKTRONIX-4027
   TELERAY-1061
   TELERAY-3700
   TELERAY-3800
   TELETEC-DATASCREEN
   TELETERM-1030
   TELETYPE-33
```

TELETYPE-35
TELETYPE-37
TELETYPE-38
TELETYPE-43
TELEVIDEO-912
TELEVIDEO-920
TELEVIDEO-920B
TELEVIDEO-920C
TELEVIDEO-950
TERMINET-1200
TERMINET-300
TI-700
TI-733
TI-735
TI-743
TI-745
TYCOM
UNIVAC-DCT-500
VIDEO-SYSTEMS-1200
VIDEO-SYSTEMS-5000
VISUAL-200
XEROX-1720
ZENITH-H19
ZENTEC-30

DOCUMENTS

[1] Accetta, M., "Resource Location Protocol", RFC 887, Carnegie-Mellon University, December 1983.

[2] Aerospace, Internal Report, ATM-83(3920-01)-3, 1982.

[3] Anderson, B., "TACACS User Identification Telnet Option", RFC 927, BBN, December 1984.

[4] Apollo Computer, Inc., "Domain TCP/IP Reference", Order No. 003247, Chelmsford, Ma.

[5] ARPANET Protocol Handbook, "Telnet Output Line Width Option", NIC 20196, November 1973.

[6] ARPANET Protocol Handbook, "Telnet Output Page Size Option", NIC 20197, November 1973.

[7] ARPANET Protocol Handbook, "Telnet Reconnection Option", NIC 15391, August 1973.

[8] Aupperle, E. M., "Merit's Evolution - Statistically Speaking", IEEE Transaction on Computers, Vol. C-32, No. 10, October 1983, pp. 881-902.

[9] BBN Proposal No. P83-COM-40, "Packet Switched Overlay to Tactical Multichannel/Satellite Systems".

[10] BBN, "Specifications for the Interconnection of a Host and an IMP", Report 1822, Bolt Beranek and Newman, Cambridge, Massachusetts, revised, December 1981.

[11] BBN, "User Manual for TAC User Database Tool", Bolt Beranek and Newman, September 1984.

[12] Bennett, C., "A Simple NIFTP-Based Mail System", IEN 169, University College, London, January 1981.

[13] Bhushan, A., "A Report on the Survey Project", RFC 530, NIC 17375, June 1973.

[14] Bisbey, R., D. Hollingworth, and B. Britt, "Graphics Language (version 2.1)", ISI/TM-80-18, Information Sciences Institute, July 1980.

[15] Boggs, D., J. Shoch, E. Taft, and R. Metcalfe, "PUP: An Internetwork Architecture", XEROX Palo Alto Research Center, CSL-79-10, July 1979; also in IEEE Transactions on Communication, Volume COM-28, Number 4, April 1980.

[16] Braden, R., "NETRJS Protocol", RFC 740, NIC 42423, November 1977.

[17] Bressler, B., "Remote Job Entry Protocol", RFC 407, NIC 12112, October 72.

[18] Bressler, R., "Inter-Entity Communication -- An Experiment", RFC 441, NIC 13773, January 1973.

[19] Butler, M., J. Postel, D. Chase, J. Goldberger, and J. K. Reynolds, "Post Office Protocol - Version 2", RFC 937, Information Sciences Institute, February 1985.

[20] Clark, D., "Revision of DSP Specification", Local Network Note 9, Laboratory for Computer Science, MIT, June 1977.

[21] Cohen, D., "Specifications for the Network Voice Protocol", RFC 741, ISI/RR 7539, Information Sciences Institute, March 1976.

[22] Cohen, D. and J. Postel, "Multiplexing Protocol", IEN 90, Information Sciences Institute, May 1979.

[23] COMPASS, "Semi-Annual Technical Report", CADD-7603-0411, Massachusetts Computer Associates, 4 March 1976. Also as, "National Software Works, Status Report No. 1," RADC-TR-76-276, Volume 1, September 1976. And COMPASS. "Second Semi-Annual Report," CADD-7608-1611, Massachusetts Computer Associates, August 1976.

[24] Crispin, M., "Telnet Logout Option", Stanford University-AI, RFC 727, April 1977.

[25] Crispin, M., "Telnet SUPDUP Option", Stanford University-AI, RFC 736, October 1977.

[26] Crispin, M., "SUPDUP Protocol", RFC 734, NIC 41953, October 1977.

[27] Crocker, D., "Telnet Output Carriage-Return Disposition Option", RFC 652, October 1974.

[28] Crocker, D., "Telnet Output Formfeed Disposition Option", RFC 655, October 1974.

[29] Crocker, D., "Telnet Output Linefeed Disposition", RFC 658, October 1974.

[30] Crocker, D., "Telnet Output Horizontal Tab Disposition Option", RFC 654,

[31] Crocker, D., "Telnet Output Horizontal Tabstops Option", RFC 653, October 1974.

[32] Crocker, D., "Telnet Output Vertical Tab Disposition Option", RFC 657, October 1974.

[33] Crocker, D., "Telnet Output Vertical Tabstops Option", RFC 656, October 1974.

[34] Crocker, D. H. and R. H. Gumpertz, "Revised Telnet Byte Marco Option", RFC 735, November 1977.

[35] Croft, B., and J. Gilmore, "BOOTSTRAP Protocol (BOOTP)", RFC 951, Stanford and SUN Microsytems, September 1985.

[36] Croft, W. J., "Unix Networking at Purdue", USENIX Conference, 1980.

[37] Day, J., "Telnet Data Entry Terminal Option", RFC 732, September 1977.

[38] Elvy, M., and R. Nedved, "Network Mail Path Service", RFC 915, Harvard and CMU, December 1984.

[39] Feinler, E., "Internet Protocol Transition Workbook", Network Information Center, SRI International, March 1982.

[40] Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[41] Feinler, E., K. Harrenstien, Z. Su, and V. White, "DoD Internet Host Table Specification", RFC 810, SRI International, March 1982.

[42] Finlayson, R., T. Mann, J. Mogul, and M. Theimer, "A Reverse Address Resolution Protocol", RFC 903, Stanford University, June 1984.

[43] Forgie, J., "ST - A Proposed Internet Stream Protocol", IEN 119, MIT Lincoln Laboratory, September 1979.

[44] Forsdick, H., "CFTP", Network Message, Bolt Beranek and Newman, January 1982.

[45] Greenberg, B., "Telnet SUPDUP-OUTPUT Option", RFC 749, MIT-Multics, September 1978.

[46] Harrenstien, K., "Name/Finger", RFC 742, NIC 42758, SRI International, December 1977.

[47] Harrenstien, K., V. White, and E. Feinler, "Hostnames Server", RFC 811, SRI International, March 1982.

[48] Harrenstien, K., and V. White, "Nicname/Whois", RFC 812, SRI International, March 1982.

[49] Haverty, J., "XNET Formats for Internet Protocol Version 4", IEN 158, October 1980.

[50] Hinden, R. M., "A Host Monitoring Protocol", RFC 869, Bolt Beranek and Newman, December 1983.

[51] Hinden, R., and A. Sheltzer, "The DARPA Internet Gateway", RFC 823, September 1982.

[52] Honeywell CISL, Internal Document, "AFSDSC Hyperchannel RPQ Project Plan".

[53] Honeywell CISL, Internal Document, "Multics MR11 PFS".

[54] Hornig, C., "A Standard for the Transmission of IP Datagrams over Ethernet Networks, RFC 894, Symbolics, April 1984.

[55] Hwang, K., W. J. Croft and G. H. Goble, "A Unix-Based Local Computer Network with Load Balancing", IEEE Computer, April 1982.

[56] IBM Corporation, "Technical Reference Manual for the IBM PC Network", 6322505, IBM, Boca Raton, Florida, 1984.

[57] International Standards Organization, "ISO Transport Protocol Specification - ISO DP 8073", RFC 905, April 1984.

[58] International Standards Organization, "Protocol for Providing the Connectionless-Mode Network Services", RFC 926, ISO, December 1984.

[59] Killian, E., "Telnet Send-Location Option", RFC 779, April 1981.

[60] Korb, J. T., "A Standard for the Transmission of IP Datagrams Over Public Data Networks", RFC 877, Purdue University, September 1983.

[61] Lapsley, P., and B. Kantor, "USENET News Transfer Protocol", Draft Memo, April 1985.

[62] Leffler, S. J., et al., "4.2bsd Network Implementation Notes", University of California, Berkeley, July 1983.

[63] Lottor, M. K., "Simple File Transfer Protocol", RFC 913, MIT, September 1984.

[64] Macgregor, W., and D. Tappan, "The CRONUS Virtual Local Network", RFC 824, Bolt Beranek and Newman, August 1982.

[65] Malis, A., "The ARPANET 1822L Host Access Protocol", RFC 878, BBN-CC, Cambridge, December 1983.

[66] Malis, A., "Logical Addressing Implementation Specification", BBN Report 5256, pp 31-36, May 1983.

[67] Metcalfe, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks", Communications of the ACM, 19 (7), pp 395-402, July 1976.

[68] Miller, T., "Internet Reliable Transaction Protocol", RFC 938, ACC, February 1985.

[69] Mills, D., "DCN Local Network Protocols", RFC 891, Linkabit, December 1983.

[70] Mills, D., "Network Time Protocol", RFC 958, M/A-COM Linkabit, September 1985.

[71] Mockapetris, P., "Domain Names - Concepts and Facilities", RFC 882, ISI, November 1983.

[72] Mockapetris, P., "Domain Names - Implementation and Specification", RFC 883, ISI, November 1983.

[73] Nedved, R., "Telnet Terminal Location Number Option", RFC 946, Carnegie-Mellon University, May 1985.

[74] NSW Protocol Committee, "MSG: The Interprocess Communication Facility for the National Software Works", CADD-7612-2411, Massachusetts Computer Associates, BBN 3237, Bolt Beranek and Newman, Revised December 1976.

[75] Plummer, D., "An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48-bit Ethernet Addresses for Transmission on Ethernet Hardware", RFC 826, MIT-LCS, November 1982.

[76] Postel, J., "Active Users", RFC 866, Information Sciences Institute, May 1983.

[77] Postel, J., "A Standard for the Transmission of IP Datagrams over Experimental Ethernet Networks, RFC 895, Information Sciences Institute, April 1984.

[78] Postel, J., "Character Generator Protocol", RFC 864, Information Sciences Institute, May 1983.

[79] Postel, J., "Daytime Protocol", RFC 867, Information Sciences Institute, May 1983.

[80] Postel, J., "Discard Protocol", RFC 863, Information Sciences Institute, May 1983.

[81] Postel, J., "The Domain Names Plan and Schedule", RFC 881, ISI, November 1983.

[82] Postel, J., "Echo Protocol", RFC 862, Information Sciences Institute, May 1983.

[83] Postel, J., "File Transfer Protocol", RFC 765, IEN 149, Information Sciences Institute, June 1980.

[84] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification", RFC 792, Information Sciences Institute, September 1981.

[83] Postel, J., "Internet Message Protocol", RFC 759, IEN 113, Information Sciences Institute, August 1980.

[84] Postel, J., "Name Server", IEN 116, Information Sciences Institute, August 1979.

[85] Postel, J., "Quote of the Day Protocol", RFC 865, Information Sciences Institute, May 1983.

[86] Postel, J., "Remote Telnet Service", RFC 818, Information Sciences Institute, November 1982.

[87] Postel, J., "Simple Mail Transfer Protocol", RFC 821, Information Sciences Institute, August 1982.

[90] Postel, J., "Telnet End of Record Option", RFC 885, Information Sciences Institute, December 1983.

[91] Postel, J., "User Datagram Protocol", RFC 768 Information Sciences Institute, August 1980.

[92] Postel, J., ed., "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, Information Sciences Institute, September 1981.

[93] Postel, J., ed., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, Information Sciences Institute, September 1981.

[94] Postel, J. and D. Crocker, "Remote Controlled Transmission and Echoing Telnet Option", RFC 726, March 1977.

[95] Postel, J., and K. Harrenstien, "Time Protocol", RFC 868, Information Sciences Institute, May 1983.

[96] Postel, J. and J. Reynolds, "Telnet Extended Options - List Option", RFC 861, Information Sciences Institute, May 1983.

[97] Postel, J. and J. Reynolds, "Telnet Binary Transmission", RFC 856, Information Sciences Institute, May 1983.

[98] Postel, J. and J. Reynolds, "Telnet Echo Option", RFC 857, Information Sciences Institute, May 1983.

[99] Postel, J., and J. Reynolds, "Telnet Protocol Specification", RFC 854, Information Sciences Institute, May 1983.

[100] Postel, J. and J. Reynolds, "Telnet Status Option", RFC 859, Information Sciences Institute, May 1983.

[101] Postel, J. and J. Reynolds, "Telnet Suppress Go Ahead Option", RFC 858, Information Sciences Institute, May 1983.

[102] Postel, J. and J. Reynolds, "Telnet Timing Mark Option", RFC 860, Information Sciences Institute, May 1983.

[103] Reed, D., "Protocols for the LCS Network", Local Network Note 3, Laboratory for Computer Science, MIT, November 1976.

[104] Reynolds, J. and J. Postel, "Official ARPA-Internet Protocols", RFC 961, Information Sciences Institute, November 1985.

[105] Rosen, E., "Exterior Gateway Protocol" RFC 827, Bolt Beranek and Newman, October 1982.

[106] Saltzer, J. H., "Design of a Ten-megabit/sec Token Ring Network", MIT Laboratory for Computer Science Technical Report.

[107] Scott, W. S., "2.9bsd/TIS Network Implementation", Lawrence Livermore National Laboratory, September 1984.

[108] Seamonson, L. J., and E. C. Rosen, "STUB" Exterior Gateway Protocol", RFC 888, BBN Communications Corporation, January 1984.

[109] Shuttleworth, B., "A Documentary of MFENet, a National Computer Network", UCRL-52317, Lawrence Livermore Labs, Livermore, California, June 1977.

[110] Silverman, S., "Output Marking Telnet Option", RFC 933, MITRE, January 1985.

[111] Skelton, A., S. Holmgren, and D. Wood, "The MITRE Cablenet Project", IEN 96, April 1979.

[112] Sollins, K., "The TFTP Protocol (Revision 2)", RFC 783, MIT/LCS, June 1981.

[113] Solomon, M., L. Landweber, and D. Neuhengen, "The CSNET Name Server", Computer Networks, v.6, n.3, pp. 161-172, July 1982.

[114] Solomon, M., and E. Wimmers, "Telnet Terminal Type Option", RFC 930, Supercedes RFC 884, University of Wisconsin, Madison, January 1985.

[115] Sproull, R., and E. Thomas, "A Networks Graphics Protocol", NIC 24308, August 1974.

[116] StJohns, M., "Authentication Service", RFC 931, TPSC, January 1985.

[117] Tappan, D. C., "The CRONUS Virtual Local Network", RFC 824, Bolt Beranek and Newman, August 1982.

[118] Taylor, J., "ERPC Functional Specification", Version 1.04, HYDRA Computer Systems, Inc., July 1984.

[119] "The Ethernet, a Local Area Network: Data Link Layer and Physical Layer Specification", AA-K759B-TK, Digital Equipment Corporation, Maynard, MA.

[120] "The Ethernet - A Local Area Network", Version 1.0, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, September 1980.

[121] "The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications", Digital, Intel and Xerox, November 1982.

[122] The High Level Protocol Group, "A Network Independent File Transfer Protocol", INWG Protocol Note 86, December 1977.

[123] Tovar, "Telnet Extended ASCII Option", RFC 698, Stanford University-AI, July 1975.

[124] Uttal, J, J. Rothschild, and C. Kline, "Transparent Integration of UNIX and MS-DOS", Locus Computing Corporation.

[125] Velten, D., R. Hinden, and J. Sax, "Reliable Data Protocol", RFC 908, BBN Communications Corporation, July 1984.

[126] Whelan, D., "The Caltech Computer Science Department Network", 5052:D F:82, Caltech Computer Science Department, 1892.

[127] Winston, I., "Two Methods for the Transmission of IP Datagrams Over IEEE 802.3 Networks", RFC 948, University Of Pennsylvania, June 1985.

[128] XEROX, "The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specification", X3T51/80-50, Xerox Corporation, Stamford, CT., October 1980.

[129] XEROX, "Internet Transport Protocols", XSIS 028112, Xerox Corporation, Stamford, Connecticut, December 1981.

PEOPLE

[AB13]	Alison Brown	CORNELL	alison@CORNELL.ARPA
[AB20]	Art Berggreen	ACC	ART@ACC.ARPA
[AD22]	Arlene DesJardins	CIT	arlene@CIT-20.ARPA
[AG22]	Alfred Ganz	YALE	GANZ@YALE.ARPA
[AGM]	Andy Malis	BBN	Malis@BBNCCS.ARPA
[AKC]	Albert Cheng	UIUC	acheng@UIUC.ARPA
[AL6]	Alexis Layton	CCA	alex@CCA-UNIX.ARPA
[AP]	Alan Parker	NRL	parker@NRL-CSS.ARPA
[AV]	Al Vezza	MIT	AV@MIT-XX.ARPA
[AW34]	Albert Wong	NPS	AWong@NPS-CS.ARPA
[AXG]	Atul Garg	HP	---none---
[AXH]	Arthur Hartwig	UQNET	---none---
[AY5]	Akiharu Yasuda	DODIIS	dia@PAXRV-NES.ARPA
[BA4]	Brian Anderson	BBN	baanders@BBNCCQ.ARPA
[BANDY]	Andrew S. Beals	LLNL	bandy@LLL-CRG.ARPA
[BC14]	Robert Cattani	COLUMBIA	Cattani@COLUMBIA-20.ARPA
[BG5]	Bob Gilligan	SRI	Gilligan@SRI-SPAM.ARPA
[BG25]	Bryan L. Gorman	SRI	GORMAN@SRI-SPAM.ARPA
[BIM]	Benson I. Margulies	HONEYWELL	Margulies@CISL.ARPA
[BJL5]	Barry J. Lustig	UCLA	barry@LOCUS.UCLA.EDU
[BJN1]	Bruce Nemnich	TMC	BJN@THINK.ARPA
[BN4]	Bill Nowicki	SUN	Nowicki@SU-GLACIER.ARPA
[BN7]	Bich T. Nguyen	SRI	btn@SRI-TSC.ARPA
[BN9]	Bill Nesheim	CORNELL	bill@CORNELL.ARPA
[BP17]	Bobbi Phillips	SRI	bobbi@SRI-TSC.ARPA
[BSW]	Barbara Seber-Wagner	MITRE	bnsw@MITRE-BEDFORD.ARPA
[BXA]	Bobby W. Allen	YPG	WYMER@OFFICE.ARPA
[BXD]	Brian Down	TORONTO	bdown%TORONTO@CSNET-RELAY.ARPA
[BXG]	Barry Lustig	UCLA	BARRY@LOCUS.UCLA.EDU
[BXL]	Barry Greenberg	LOCUS	---none---
[BXM]	Bill Mitchell	U OF ARIZ	---none---
[CAK]	Chris Kent	PURDUE	CAK@PURDUE.EDU
[CAS]	Carl Sunshine	SDC	Sunshine@USC-ISIB.ARPA
[CBD]	Clive B. Dawson	MCC	Clive@MCC.ARPA
[CBP]	Brian Pinkerton	WISCONSON	Brian@WISC-RSCH.ARPA
[CJC3]	Chase Cotton	UDEL	Cotton@UDEL-EE.ARPA
[CH2]	Charles Hornig	SYMBOLICS	CAH@MIT-MC.ARPA
[CJW2]	Cliff Weinstein	LL	cjw@LL-SST.ARPA
[CLH3]	Charles Hedrick	RUTGERS	Hedrick@RUTGERS.EDU
[CMR]	Craig Rogers	ISI	Rogers@USC-ISIB.ARPA
[CP10]	Craig Partridge	BBN	craig@BBN-UNIX.ARPA
[CXH]	Chien Y. Huang	PRINCETON	6026959%PUCC.BINET@WISCVM.ARPA
[CXL]	Clifford A. Lynch	BERKELEY	udcla%ucbtopaz.cc@UCBARPA.BERKELEY.EDU
[DAM1]	David A. Mosher	BERKELEY	Mosher@UCBARPA.BERKELEY.EDU

[DAVE]	David Roode	IntelliCorp	Roode@SUMEX-AIM.ARPA
[DBJ]	David B. Johnson	DRILLTECH	DBJ@RICE.ARPA
[DCP1]	David Plummer	MIT	DCP@SYMBOLICS.ARPA
[DDC1]	David Clark	MIT	DClark@BBN-UNIX.ARPA
[DT15]	Dan Tappan	BBN	Tappan@BBNG.ARPA
[DG28]	David L. Gehrt	RIACS	Dave@RIACS.ARPA
[DH17]	Douglas Hirsch	BBN	hirsch@BBNCCS.ARPA
[DHH]	Doug Hunt	BBN	DHunt@BBNCCJ.ARPA
[DJF]	David J. Farber	UDEL	Farber@UDEL-EE.ARPA
[DJV1]	Darrel J. Van Buer	SDC	vanbuer@USC-ECL.ARPA
[DK2]	Dean B. Krafft	CORNELL	Dean@CORNELL.ARPA
[DLM1]	David Mills	LINKABIT	Mills@USC-ISID.ARPA
[DPR]	David Reed	MIT-LCS	Reed@MIT-MULTICS.ARPA
[DRP]	Don Provan	LLNL	Provan@LLL-MFE.ARPA
[DRS4]	Dennis R. Smith	USC	Smith@USC-ECLC.ARPA
[DSW]	Dan Whelan	CALTECH	Dan@CIT-20.ARPA
[DVC]	Don Cone	SRI	CONE@SRI-SPAM.ARPA
[DXB]	David Bloom	RUTGERS	andromeda!bloom@RUTGERS.EDU
[DXD]	Dennis J.W. Dube	VIA SYSTEMS	---none---
[DXG]	David Goldberg	SMI	sun!dg@UCBARPA.BERKELEY.EDU
[DXS]	Don Scelza	PERQ	---none---
[DXT]	Dave Taylor	INFERENCE	---none---
[EAK1]	Earl Killian	LLL	EAK@S1-C.ARPA
[EBM]	Eliot Moss	MIT	EBM@MIT-XX.ARPA
[EC5]	Ed Cain	DCEC	cain@EDN-UNIX.ARPA
[EF5]	Ed Franceschini	NYU	Franceschini@NYU.ARPA
[EHP]	Ed Perry	SRI	Perry@SRI-KL.ARPA
[EXY]	Elaine Yamin	ATT	---none---
[FAS]	Fred Segovich	GSWD	fred@GSWD-VMS.ARPA
[FLM2]	F. Lee Maybaum	MILNET	Maybaum@DDN1.ARPA
[FRAN]	Francine Perillo	SRI	Perillo@SRI-NIC.ARPA
[FXS]	Frank Solensky	PRIME	---none---
[GEOF]	Geoff Goodfellow	SRI	Geoff@SRI-CSL.ARPA
[GAA]	Glenn A. Adams, Jr.	MIT/LL	glenn@LL-XN.ARPA
[GC]	Graham Campbell	BNL	gc@BNL.ARPA
[GH29]	Gregory Hidley	UCSD	hidley@UCSD.ARPA
[GIH]	Glenn I. Hastie II	SRI	Hastie@SRI-SPAM.ARPA
[GLH5]	Gavin L. Hamphill	DREA	Hemphill@DREA-XX.ARPA
[GP10]	George Pavel	LLNL	liaison@LLL-TIS.ARPA
[GW22]	Grant Weiler	UTAH	Weiler@UTAH-20.ARPA
[GXL]	Guillermo A. Loyola	IBM	Loyola%ibm-sj@CSNET-RELAY.ARPA
[GXP]	Gill Pratt	MIT	gill%mit-ccc@MIT-MC.ARPA
[HCF2]	Harry Forsdick	BBN	Forsdick@BBNA.ARPA
[HDC1]	Horst Clausen	DFVLR	Clausen@USC-ISID.ARPA
[HDW2]	Howard Wactlar	CMU	Wactlar@CMU-CS-A.ARPA
[HGM]	Hallam Murray	XEROX	Murray.PA@XEROX.ARPA
[HM]	Hank Magnuski	---	JOSE@XEROX.PA.ARPA
[HWB]	Hans-Werner Braun	MICHIGAN	HWB@UMICH1.ARPA

[JA1]	Jules P. Aronson	NLM	Aronson@NLM-MCS.ARPA
[JAG3]	Jeff Gumpf	CWRU	G.Gumpf@COLUMBIA-20.ARPA
[JAKE]	Jake Feinler	SRI	Feinler@SRI-NIC.ARPA
[JAR4]	Jim Rees	WASHINGTON	JIM@WASHINGTON.ARPA
[JBP]	Jon Postel	ISI	Postel@USC-ISIB.ARPA
[JBW1]	Joseph Walters, Jr.	BBN	JWalters@BBNCCX.ARPA
[JC11]	Jim Clifford	LANL	jrc@LANL.ARPA
[JCN2]	John C. Nunn	NBS	NUNN@NBS-VMS.ARPA
[JD21]	Jonathan Dreyer	BBN	JDreyer@BBNCCV.ARPA
[JDG]	Jim Guyton	RAND	guyton@RAND-UNIX.ARPA
[JEM]	Jim Mathis	SRI	Mathis@SRI-KL.ARPA
[JFH2]	Jack Haverty	BBN	Haverty@BBNCCV.ARPA
[JFW]	Jon F. Wilkes	STC	Wilkes@STC.ARPA
[JGH]	Jim Herman	BBN	Herman@BBNCCJ.ARPA
[JG46]	Jonathan Goodman	YALE	Goodman@YALE.ARPA
[JKR1]	Joyce K. Reynolds	ISI	JKREYNOLDS@USC-ISIB.ARPA
[JL15]	Jay Lepreau	UTAH	Lepreau@UTAH-CS.ARPA
[JLM23]	John L. Mills	HONEYWELL	Mills@CISL-SERVICE-MULTICS.ARPA
[JO5]	John O'Donnell	YALE	ODonnell@YALE.ARPA
[JR15]	John Rhodes	LOGNET	JRhodes@LOGNET2.ARPA
[JR17]	John L. Robinson	CANADA	Robinson@DMC-CRC.ARPA
[JRM1]	John Mullen	MITRE	Mullen@MITRE.ARPA
[JRS8]	Jeffrey R. Schwab	PURDUE	jrs@PURDUE.EDU
[JS38]	Joseph Sventek	LBL	JSSventek@LBL.ARPA
[JSG5]	Jon Goodridge	BBN	jsg@BBNCCM.ARPA
[JSQ1]	John S. Quarterman	UT	jsq@UT-SALLY.ARPA
[JW1]	Jill Westcott	BBN	Westcott@BBNA.ARPA
[JWF]	Jim Forgie	LL	jwf@LL-EN.ARPA
[JWO1]	James W. O'Toole	UMD	james@MARYLAND.ARPA
[JXB]	John Blair	NEOCM	cbosgd!neoucom!johnb@UCBARPA.BERKELEY.EDU
[JXD]	Jean Darling	WISC-MADI	Darling@UWISC.ARPA
[JXJ]	Jackie Jones	NBS	----none----
[JXO]	Jack O'Neil	ENCORE	----none----
[JXS]	J. Simonetti	SUNY	joes@SBCS.ARPA
[JXY]	Joe Yancone	USARMY	Yancone@CRDC.ARPA
[KCS1]	Kevin C. Smallwood	PURDUE	kcs@PURDUE.EDU
[KFD]	Ken Dove	AIDS	kfd@AID-UNIX.ARPA
[KLH]	Ken Harrenstien	SRI	KLH@SRI-NIC.ARPA
[KMC3]	Kenneth M. Crepea	SRI	Crepea@SRI-SPAM.ARPA
[KO11]	Kevin O'Keefe	HAZELTINE	Hazeltine@USC-ISI.ARPA
[KRS]	Karen Sollins	MIT	Sollins@MIT-XX.ARPA
[KTP]	Kenneth T. Pogran	BBN	Pogran@BBNBBNCCQ.ARPA
[KWP]	Kevin W. Paetzold	DEC	Paetzold@DEC-MARLBORO.ARPA
[KXC]	Ken Chen	Perceptronics	----none----
[KXS]	Kathy Simpson	OSU	----none----
[LB3]	Len Bosack	STANFORD	Bosack@SU-SCORE.ARPA

[LB16]	Liudvikas Bukys	ROCHESTER	Bukys@ROCHESTER.ARPA
[LCN]	Lou Nelson	AEROSPACE	Lou@AEROSPACE.ARPA
[LCS]	Lou Schreier	SRI	Schreier@USC-ISID.ARPA
[LH2]	Lincoln Hu	COLUMBIA	Hu@COLUMBIA-20.ARPA
[LOU]	Lou Salkind	NYU	Salkind@NYU.ARPA
[LM8]	Liza Martin	MIT-LCS	Martin@MIT-XX.ARPA
[LRB]	Larry Bierma	NPRDC	Bierma@NPRDC.ARPA
[LWR]	Larry Robinson	LLNL	lwr@S1-C.ARPA
[LXL]	Len Lattanzi	SENTRY	----none----
[MA]	Mike Accetta	CMU	MIKE.ACCETTA@CMU-CS-A.ARPA
[MAB4]	Mark Brown	USC	Mark@USC-ECLB.ARPA
[MAE]	Marc A. Elvy	HARVARD	elvy@HARVARD.EDU
[MBG]	Michael Greenwald	MIT-LCS	Greenwald@MIT-MULTICS.ARPA
[MB]	Michael Brescia	BBN	Brescia@BBNCCV.ARPA
[MB31]	Michael Bereschinsky	USARMY	Bereschinsky@USC-ISID.ARPA
[MCA1]	Mary C. Akers	FISG	MCAkers@TPSC-T.ARPA
[MCSJ]	Mike StJohns	TPSC	StJohns@MIT-MULTICS.ARPA
[MDC]	Martin D. Connor	MIT AI	Marty@MIT-HTVAX.ARPA
[MF31]	Martin J. Fouts	NASA-AMES	fouts@AMES-NAS.ARPA
[MH12]	Mark Horton	ATT	mark@UCBARPA.BERKELEY.EDU
[MJM2]	Mike Muuss	BRL	Mike@BRL.ARPA
[MK17]	Mike Karels	BERKELEY	Karels@UCBARPA.BERKELEY.EDU
[MKL1]	Mark Lottor	MIT	MKL@SRI-NIC.ARPA
[MLC]	Mike Corrigan	DDN	Corrigan@DDN1.ARPA
[MO2]	Michael O'Brien	RAND	OBrien@RAND-UNIX.ARPA
[MO14]	Michele Olivant	JHU	Olivant@HAWAII-EMH.ARPA
[MRC]	Mark Crispin	STANFORD	Admin.MRC@SU-SCORE.ARPA
[MS9]	Martin Schoffstall	RPI	schoff%rpi@CSNET-RELAY.ARPA
[MS56]	Marvin Solomon	WISC	Solomon@UWISC.ARPA
[MSM1]	Milo S. Medin	AMES	medin@AMES.ARPA
[MTR]	Marshall Rose	IRVINE	MRose.UCI@RAND-RELAY.ARPA
[MXA]	Melanie Anderson	UIUC	Melanie%UIUCVMD.BITNET@WISCVM.ARPA
[MXA1]	M. Aziza	INRIA	----none----
[MXG]	Mike Gilbert	SLI	Software-Leverage@USC-ECLB.ARPA
[MXH]	Martin Hayman	Symbolics	----none----
[MXK]	Michael Kazar	CMU	Mike.Kazar@CMU-CS-K.ARPA
[MXM]	Marc M. Meilleur	COINS	COINS@USC-ISI.ARPA
[MXP]	Michael K. Peterson	HUGHES	scgvaxd!mkp@CIT-VAX.ARPA
[MXP1]	Mark C. Powers	NSWC	mpowers@NSWC-G.ARPA
[MXR]	Mark A. Rosenstein	MIT	mar@MIT-BORAX.ARPA
[MXS]	Marc Shapiro	INRIA	Marc.Shapiro@C.CS.CMU.EDU
[NC3]	J. Noel Chiappa	MIT	JNC@MIT-XX.ARPA
[NG]	Neil Gower	ROCKWELL	GOWER@USC-ISID.ARPA
[NMM]	Mike Minnich	UDELEE	MMinnich@UDEL-HUEY.ARPA
[NXH]	Nat Howard	IM	nrh@DDN1.ARPA
[NXK]	Neil Katin	HP	hpda.neil@UCBARPA.BERKELEY.EDU
[PA5]	Philip Almquist	STANFORD	Almquist@SU-SCORE.ARPA
[PAM6]	Paul McNabb	RICE	pam@PURDUE.EDU

[PFS2]	Paul Sass	CECOM	Sass@USC-ISID.ARPA
[PGM]	Paul G. Milazzo	RICE	Milazzo@RICE.ARPA
[PHD1]	Pieter Ditmars	BBN	pditmars@BBNCCX.ARPA
[PK]	Peter Kirstein	UCL	Kirstein@USC-ISI.ARPA
[PK28]	Philip R. Karn, Jr.	BCR	Karn@BELLCORE-CS-GW.ARPA
[PL4]	Phil Lapsley	BERKELEY	phil@UCBARPA.BERKELEY.EDU
[PM1]	Paul Mockapetris	ISI	Mockapetris@USC-ISIB.ARPA
[PM4]	Paul Martin	SRI	PMartin@SRI-AI.ARPA
[PS27]	Paal Spilling	NTA	Spilling@USC-ISID.ARPA
[PXA]	Phillip G. Apley	BITSTREAM	PGA@MIT-OZ.ARPA
[PXB]	Pat Boyle	UBC	boyle.ubc@CSNET-RELAY.ARPA
[PXD]	Pete Delaney	ECRC	pete%ecrcvax@CSNET-RELAY.ARPA
[PXM]	Pat Marques	NSRDC	marques@DTRC.ARPA
[PXN]	Peter Nellessen	SIEMENS	crtvax!pn@CMU-CS-SPICE.ARPA
[RA11]	Rick Adams	CCI	Rick@SEISMO.CSS.GOV
[RA17]	Bob Albrightson	WASHINGTON	BOB@WASHINGTON.ARPA
[RB9]	Richard Bisbey	ISI	Bisbey@USC-ISIB.ARPA
[RBN1]	Ronald Natalie, Jr.	BRL	ron@BRL-TGR.ARPA
[RBW]	Richard B. Wales	UCLA	WALES@LOCUS.UCLA.EDU
[RHC3]	Robert Cole	UCL	robert@UCL-CS.ARPA
[RC77]	Robert Carey	YALE	CAREY@YALE.ARPA
[RDB2]	Robert Bressler	BBN	Bressler@BBNCCW.ARPA
[RDR4]	Dennis Rockwell	BBN	DRockwell@CSNET-SH.ARPA
[RFD1]	Robert F. Donnelly	ARDC	donnelly@ARDC.ARPA
[RG12]	Roger L. Gulbranson	UMINN	ROGERG@UMN-UCC-VA.ARPA
[RH6]	Robert Hinden	BBN	Hinden@BBN-CCV.ARPA
[RH60]	Roger Hale	MIT	Roger@LL-SST.ARPA
[RHC3]	Robert Cole	UCL	Robert@USC-CS.ARPA
[RHT]	Robert Thomas	BBN	BThomas@BBNF.ARPA
[RKJ2]	Richard Johnsson	DEC	johnsson@DECWRL.ARPA
[RL2]	Randy C. Lee	HONEYWELL	RCLee@HI-MULTICS.ARPA
[RLB3]	Ronald L. Broersma	NOSC	Ron@NOSC.ARPA
[RLH2]	Ronald L. Hartung	NSWC	ron@NSWC-WO.ARPA
[RLS6]	Ronald L. Smith	COINS	COINS@USC-ISI.ARPA
[RM8]	Roy Marantz	RUTGERS	Marantz@RUTGERS.EDU
[RN6]	Rudy Nedved	CMU	Rudy.Nedved@CMU-CS-A.ARPA
[RNM1]	Neil MacKenzie	RSRE	CLE%RSRE@UCL-CS.ARPA
[RR2]	Raleigh Romine	TELEDYNE	romine@SEISMO.CSS.GOV
[RR18]	Ron Reisor	UDEL	ron@UDEL-EE.ARPA
[RR26]	William R. Reilly	USARMY	RREILLY@JPL-MILVAX.ARPA
[RS23]	Russel Sandberg	WISC	root@UWISC.ARPA
[RSM1]	Robert S. Miles	NRTC	RSM@BRL.ARPA
[RTB3]	Bob Braden	UCLA	Braden@UCLA-CCN.ARPA
[RWS4]	Robert W. Scheifler	ARGUS	RWS@MIT-XX.ARPA
[RXB]	Rafael Bracho	SPAR	RXB@SRI-KL.ARPA
[RXB1]	Randolph Bentson	CSU	Bentson%ColoState@CSNET-RELAY.ARPA
[RXG]	Richard Gopstein	RCA	Gopstein@RUTGERS.EDU

[RXJ]	Ronald Johnson	APPLE	rlj%apple@CSNET-RELAY.ARPA
[RXM]	Robert Myhill	BBN	Myhill@BBNCCS.ARPA
[SA1]	Sten Andler	ARPA	andler.ibm-sj@RAND-RELAY.ARPA
[SA2]	Saul Amarel	ARPA	Amarel@USC-ISI.ARPA
[SC3]	Steve Casner	ISI	Casner@USC-ISIB.ARPA
[SGC]	Steve Chipman	BBN	Chipman@BBNF.ARPA
[SHB]	Steven Blumenthal	BBN	BLUMENTHAL@BBN-VAX.ARPA
[SK8]	Steve Kille	UCL	Steve@UCL-CS.ARPA
[SM6]	Sean McLinden	DSL	McLinden@RUTGERS.EDU
[SMF]	Steven M. Feldman	TYMNET	ARPAVAX.feldman@UCBARPA.BERKELEY.EDU
[SXA]	Skip Addison	GATECH	Skip!gatech.csnet@CSNET-RELAY.ARPA
[SXB]	Steve Byrne	TARTAN	Byrne@CMU-CS-C.ARPA
[SB28]	Scott Bradner	HARVARD	sob@HARVARD.EDU
[SXF]	Steve Fogel	MTCS	SFogel!mtcs!mtxinu@UCBARPA.BERKELEY.EDU
[SXM]	Scott Marcus	SPARTACUS	---none---
[SXM1]	Scooter Morris	GENENTECH	scooter@UCSF-CGL.ARPA
[SXS]	Steve Silverman	MITRE	Blankert@MITRE-GATEWAY.ARPA
[TBS]	Claude S. Steffey	WSMR	csteffey@WSMRCAS1.ARPA
[TC4]	Tony Cincotta	DTNSRDC	tony@NALCON.ARPA
[TF6]	Thomas Ferrin	UCSF	Ferrin@UCSF-CGL.ARPA
[THD]	Thomas Dunigan	ORNL	dunigan@ORNL-MSR.ARPA
[TML]	T. Michael Louden	MITRE	Louden@MITRE-GW.ARPA
[TW11]	Tom Wadlow	LLL	TAW@S1-C.ARPA
[TXB]	Ted Baker	FSU	baker@WASHINGTON.ARPA
[TXM]	Trudy Miller	ACC	Trudy@ACC.ARPA
[TXN]	Todd Nugent	U CHICAGO	Nugent@ANL-MCS.ARPA
[UXB]	Ulf Bilting	CHALMERS	bilting@PURDUE.EDU
[WDL]	Walter Lazear	MITRE	Lazear@MITRE.ARPA
[WG]	Wayne Graves	LBL	WLGraves@LBL.ARPA
[WF3]	William E. Fink	NRLRCD	bill@NRL.ARPA
[WIM]	William Macgregor	BBN	macg@BBN.ARPA
[WJC2]	Bill Croft	STANFORD	Croft@SUMEX-AIM.ARPA
[WPJ]	William Jones	USRA	Jones@AMES-VMSB.ARPA
[WW2]	Wally Wedel	NBI	wedel@UT-NGP.ARPA
[WWS]	Bill Seemuller	USARMY	bill@ETL.ARPA
[WXL]	William Lampeter	UR	bill@ROCHESTER.ARPA
[ZSU]	Zaw-Sing Su	SRI	ZSu@SRI-TSC.ARPA

APPENDIX A

Network Numbers

The network numbers in class A, B, and C network addresses are allocated among Research, Defense, Government (Non-Defense) and Commercial uses.

Class A (highest-order bit 0)

```
   Research allocation:           8
   Defense allccation:           24
   Government allocation:        24
   Commercial allocation:        94
   Reserved Addresses:    (0, 127)
   Total                        128
```

Class B (highest-order bits 1-0)

```
   Research allocation:        1024
   Defense allocation:         3072
   Government allocation:      3072
   Commercial allocation:     12286
   Reserved Addresses:  (0, 16383)
   Total                      16384
```

Class C (highest-order bits 1-1-0)

```
   Research allocation:          65536
   Defense allocation:          458725
   Government allocation:       458725
   Commercial allocation:      1572862
   Reserved Addresses:  (0, 2097151)
   Total                       2097152
```

Class D (highest-order bits 1-1-1)

All addresses in this class are reserved for future use.

Within the Research community, network identifiers will only be granted to applicants who show evidence that they are acquiring standard Bolt Beranek and Newman gateway software or have implemented or are acquiring a gateway meeting the Exterior Gateway Protocol requirements. Acquisition of the Berkeley BSD 4.2 UNIX software might be considered evidence of the latter.

Experimental networks which later become operational need not be renumbered. Rather, the identifiers could be moved from Research to Defense, Government or Commercial status. Thus, network identifiers may change state among Research, Defense, Government and Commercial, but the number of identifiers allocated to each use must remain within the limits indicated above. To make possible this fluid assignment, the network identifier spaces are not allocated by simple partition, but rather by specific assignment.

Protocol Identifiers

These assignments are shared by the four communities.

Port Numbers

These assignments are shared by the four communities.

ARPANET Link Numbers

These assignments are shared by the four communities.

IP Version Numbers

These assignments are shared by the four communities.

TCP, IP and Telnet Option Identifiers

These assignments are shared by the four communities.

Implementation:

Joyce Reynolds is the coordinator for all number assignments.

Network Working Group
Request for Comments: 794
Replaces: IEN 125

V. Cerf
ARPA
September 1981

PRE-EMPTION

In circuit-switching systems, once a user has acquired a circuit, the communication bandwidth of that circuit is dedicated, even if it is not used. When the system saturates, additional circuit set-up requests are blocked. To allow high precedence users to gain access to circuit resources, systems such as AUTOVON associate a precedence with each telephone instrument. Those instruments with high precedence can pre-empt circuit resources, causing lower precedence users to be cut off.

In message switching systems such as AUTODIN I, incoming traffic is stored on disks (or drums or tape) and processed in order of precedence. If a high precedence message is entered into the system, it is processed and forwarded as quickly as possible. When the high precedence message arrives at the destination message switch, it may pre-empt the use of the output devices on the switch, interrupting the printing of a lower precedence message.

In packet switching systems, there is little or no storage in the transport system so that precedence has little impact on delay for processing a packet. However, when a packet switching system reaches saturation, it rejects offered traffic. Precedence can be used in saturated packet switched systems to sort traffic queued for entry into the system.

In general, precedence is a tool for deciding how to allocate resources when systems are saturated. In circuit switched systems, the resource is circuits; in message switched systems the resource is the message switch processor; and in packet switching the resource is the packet switching system itself.

This capability can be realized in AUTODIN II without adding any new mechanisms to TCP (except to make precedence of incoming connection requests visible to the processes which use TCP). To allow pre-emptive access to a particular terminal, the software (i.e., THP) which supports terminal access to the TAC can be configured so as to always have a LISTEN posted for that terminal, even if the terminal has a connection in operation. For example in the ARPANET TENEX systems, the user TELNET permits a user to have many connections open at one time - the user can switch among them at will. To the extent that this can be done without violating security requirements, one could imagine a multi-connection THP which always leaves a LISTEN pending for incoming connection requests. If a connection is established, the THP can decide, based on its precedence, whether to pre-empt any existing connection and to switch the user to the high precedence one.

If the user is working with several connections of different precedence at the same time, the THP would close or abort the lowest precedence connection in favor of the higher precedence pre-empting one. Then the THP would do a new LISTEN on that terminal's port in case a higher precedence connection is attempted.

One of the reasons for suggesting this model is that processes are the users of TCP (in general) and that TCP itself cannot cause processes to be created on behalf of an incoming connection request. Implementations could be realized in which TCPs accept incoming connection requests and, based on the destination port number, create appropriate server processes. In terms of pre-empting access to a remote terminal, however, it seems more sensible to let the process which interfaces the terminal to the system mediate the pre-emption. If the terminal is not connected or is turned off, there is no point in creating a process to serve the incoming high precedence connection request.

For example, suppose a routine FTP is in operation between Host X and Host Y. Host Z decides to do a flash-override FTP to Host X. It opens a high precedence connection via its TCP and the "SYN" goes out to the FTP port on Host X.

FTP always leaves one LISTEN pending to pre-empt lower precedence remote users if it cannot serve one more user (and still keep a LISTEN pending). In this way, the FTP is naturally in a state permitting the high precedence connection request to be properly served, and the FTP can initiate any cleaning up that is needed to deal with the pre-emption.

In general, this strategy permits the processes using TCP to accommodate pre-emption in the context of the applications they support.

A non-pre-emptable process is one that does not have a LISTEN pending while it is serving one (or more) users.

The actions taken to deal with pre-emption of TCP connections will be application-process specific and this strategy of a second (or N+1st) LISTEN is well suited to the situation.

Pre-emption may also be necessary at the site initiating a high precedence connection request. Suppose there is a high precedence user who wants to open an FTP connection request from Host Z to Host X. But all FTP and/or TCP resources are saturated when this user tries to start the user FTP process. In this case, the operating system would have to know about the precedence of the user and would have to locally pre-empt resources on his behalf (e.g., by logging out lower precedence users). This is a system issue, not specific only to TCP. Implementation of pre-emption at the source could vary greatly. Precedence may be associated with a user or with a terminal. The TCP implementation may locally pre-empt resources to serve high precedence users. The operating system may make all pre-emption decisions.

Network Working Group J. Postel
Request for Comments: 795 ISI
September 1981

SERVICE MAPPINGS

This memo describes the relationship between the Internet Protocol (IP) [1] Type of Service and the service parameters of specific networks.

The IP Type of Service has the following fields:

```
Bits 0-2:  Precedence.
Bit    3:  0 = Normal Delay,      1 = Low Delay.
Bits   4:  0 = Normal Throughput, 1 = High Throughput.
Bits   5:  0 = Normal Relibility, 1 = High Relibility.
Bit  6-7:  Reserved for Future Use.

   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|                 |     |     |     |     |     |
|   PRECEDENCE    |  D  |  T  |  R  |  0  |  0  |
|                 |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine
```

The individual networks listed here have very different and specific service choices.

AUTODIN II

The service choices are in two parts: Traffic Acceptance Catagories, and Application Type. The Traffic Acceptance Catagories can be mapped into and out of the IP TOS precedence reasonably directly. The Application types can be mapped into the remaining IP TOS fields as follows.

TA	DELAY	THROUGHPUT	RELIABILITY
I/A	1	0	0
Q/R	0	0	0
B1	0	1	0
B2	0	1	1

DTR	TA
000	Q/R
001	Q/R
010	B1
011	B2
100	I/A
101	I/A
110	I/A
111	error

ARPANET

The service choices are in quite limited. There is one priority bit that can be mapped to the high order bit of the IP TOS precedence. The other choices are to use the regular ("Type 0") messages vs. the uncontrolled ("Type 3") messages, or to use single packet vs. multipacket messages. The mapping of ARPANET parameters into IP TOS parameters can be as follows.

Type	Size	DELAY	THROUGHPUT	RELIABILITY
0	S	1	0	0
0	M	0	0	0
3	S	1	0	0
3	M	not allowed		

DTR	Type	Size
000	0	M
001	0	M
010	0	M
011	0	M
100	3	S
101	0	S
110	3	S
111	error	

PRNET

There is no priority indication. The two choices are to use the station routing vs. point-to-point routing, or to require acknowledgments vs. having no acknowledgments. The mapping of PRNET parameters into IP TOS parameters can be as follows.

Routing	Acks	DELAY	THROUGHPUT	RELIABILITY
ptp	no	1	0	0
ptp	yes	1	0	1
station	no	0	0	0
station	yes	0	0	1

DTR	Routing	Acks
000	station	no
001	station	yes
010	station	no
011	station	yes
100	ptp	no
101	ptp	yes
110	ptp	no
111	ptp	yes

SATNET

There is no priority indication. The four choices are to use the block vs. stream type, to select one of four delay catagories, to select one of two holding time strategies, or to request one of three reliability levels. The mapping of SATNET parameters into IP TOS parameters can thus quite complex there being 2*4*2*3=48 distinct possibilities.

References

[1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

Network Working Group J. Postel
Request for Comments: 796 ISI
Replaces: IEN 115 September 1981

ADDRESS MAPPINGS

Internet Addresses

This memo describes the relationship between address fields used in the Internet Protocol (IP) [1] and several specific networks.

An internet address is a 32 bit quantity, with several codings as shown below.

The first type (or class a) of address has a 7-bit network number and a 24-bit local address.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|   NETWORK   |                Local Address                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class A Address

The second type (or class b) of address has a 14-bit network number and a 16-bit local address.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0|           NETWORK         |         Local Address         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class B Address

The third type (or class c) of address has a 21-bit network number and a 8-bit local address.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 0|                    NETWORK              | Local Address |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class C Address

The local address carries information to address a host in the network identified by the network number. Since each network has a particular address format and length, the following section describes the mapping between internet local addresses and the actual address format used in the particular network.

Internet to Local Net Address Mappings

The following transformations are used to convert internet addresses to local net addresses and vice versa:

AUTODIN II

The AUTODIN II has 16 bit subscriber addresses which identify either a host or a terminal. These addresses may be assigned independent of location. The 16 bit AUTODIN II address is located in the 24 bit internet local address as shown below.

The network number of the AUTODIN II is 26 (Class A).

```
+-----------------+
|  HOST/TERMINAL  |   AUTODIN II
+-----------------+
        16

+--------+--------+--------+--------+
|   26   |  ZERO  |  HOST/TERMINAL  |   IP
+--------+--------+--------+--------+
    8        8           16
```

ARPANET

The ARPANET (with 96 bit leaders) has 24 bit addresses. The 24 bits are assigned to host, logical host, and IMP leader fields as illustrated below. These 24 bit addresses are used directly for the 24 bit local address of the internet address. However, the ARPANET IMPs do not yet support this form of logical addressing so the logical host field is set to zero in the leader.

The network number of the ARPANET is 10 (Class A).

```
+--------+--------+--------+
|  HOST  |  ZERO  |  IMP   |   ARPANET
+--------+--------+--------+
    8        8        8

+--------+--------+--------+--------+
|   10   |  HOST  |   LH   |  IMP   |   IP
+--------+--------+--------+--------+
    8        8        8        8
```

DCNs

The Distributed Computing Networks (DCNs) at COMSAT and UCL use 16 bit addresses divided into an 8 bit host identifier (HID), and an 8 bit process identifier (PID). The format locates these 16 bits in the low order 16 bits of the 24 bit internet address, as shown below.

The network number of the COMSAT-DCN is 29 (Class A), and of the UCL-DCN is 30 (Class A).

```
+--------+--------+
|  HID   |  PID   |   DCN
+--------+--------+
    8        8

+--------+--------+--------+--------+
|   18   |  ZERO  |  HID   |  PID   |   IP
+--------+--------+--------+--------+
    8        8        8        8
```

EDN

The Experimental Data Network at the Defense Communication Engineering Center (DCEC) uses the same type of addresses as the ARPANET (with 96 bit leaders) and has 24 bit addresses. The 24 bits are assigned to host, logical host, and IMP leader fields as illustrated below. These 24 bit addresses are used directly for the 24 bit local address of the internet address. However, the IMPs do not yet support this form of logical addressing so the logical host field is set to zero in the leader.

The network number of the EDN is 21 (Class A).

```
+--------+--------+--------+
|  HOST  |  ZERO  |  IMP   |   EDN
+--------+--------+--------+
    8        8        8

+--------+--------+--------+--------+
|   21   |  HOST  |   LH   |  IMP   |   IP
+--------+--------+--------+--------+
    8        8        8        8
```

LCSNET

The LCS NET at MIT's Laboratory for Computer Science uses 32 bit addresses of several formats. Please see [3] for more details. The most common format locates the low order 24 bits of the 32 bit LCS NET address in the 24 bit internet local address, as shown below.

The network number of the LCS NET is 18 (Class A).

```
+--------+--------+--------+
| SUBNET |RESERVED|  HOST  |   LCSNET
+--------+--------+--------+
    8        8        8

+--------+--------+--------+--------+
|   18   | SUBNET |RESERVED|  HOST  |   IP
+--------+--------+--------+--------+
    8        8        8        8
```

PRNET

The Packet Radio networks use 16 bit addresses. These are independent of location (indeed the hosts may be mobile). The 16 bit PRNET addresses are located in the 24 bit internet local address as shown below.

The network numbers of the PRNETs are:

BBN-PR	1	(Class A)
SF-PR-1	2	(Class A)
SILL-PR	5	(Class A)
SF-PR-2	6	(Class A)
BRAGG-PR	9	(Class A)
DC-PR	20	(Class A)

```
+--------+--------+
|      HOST       |  PRNET
+--------+--------+
        16

+--------+--------+--------+--------+
|  net   |  ZERO  |      HOST       |  IP
+--------+--------+--------+--------+
    8        8            16
```

SATNET

The Atlantic Satellite Packet Network has 16 bit `ddresses for hosts. These addresses may be assigned independent of location (i.e., ground station). It is also possible to assign several addresses to one physical host, so the addresses are logical addresses. The 16 bit SATNET address is located in the 24 bit internet local address as shown below.

The network number of the SATNET is 4 (Class A).

```
+--------+--------+
|      HOST       |   SATNET
+--------+--------+
        16

+--------+--------+--------+--------+
|   4    |  ZERO  |      HOST       |   IP
+--------+--------+--------+--------+
    8        8           16
```

WBCNET

The Wideband Communication Satellite Packet Network (WBCNET) Host Access Protocol (HAP) has 16 bit addresses for hosts. It is possible to assign several addresses to one physical host, so the addresses are logical addresses. The 16 bit WBCNET address is divided into a HAP Number field and a Local Address field, and is located in the 24 bit internet local address as shown below. Please see [2] for more details.

The network number of the WBCNET is 28 (Class A).

```
+--------+--------+
| HAP NUM| LCL ADD|   WBCNET
+--------+--------+
    8        8

+--------+--------+--------+--------+
|   28   | HAP NUM|  ZERO  | LCL ADD|   IP
+--------+--------+--------+--------+
    8        8        8        8
```

References

[1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

[2] Pershing J., "Addressing Revisited," Bolt Beranek and Newman Inc., W Note 27, May 1981.

[3] Noel Chiappa, David Clark, David Reed, "LCS Net Address Format," M.I.T. Laboratory for Computer Science Network Implementation, Note No.5, IEN 82, February 1979.

Network Working Group
Request for Comments: 952

Obsoletes: RFC 810, 608

K. Harrenstien (SRI)
M. Stahl (SRI)
E. Feinler (SRI)
October 1985

DOD INTERNET HOST TABLE SPECIFICATION

STATUS OF THIS MEMO

This RFC is the official specification of the format of the Internet Host Table. This edition of the specification includes minor revisions to RFC-810 which brings it up to date. Distribution of this memo is unlimited.

INTRODUCTION

The DoD Host Table is utilized by the DoD Hostname Server maintained by the DDN Network Information Center (NIC) on behalf of the Defense Communications Agency (DCA) [See RFC-953].

LOCATION OF THE STANDARD DOD ONLINE HOST TABLE

A machine-translatable ASCII text version of the DoD Host Table is online in the file NETINFO:HOSTS.TXT on the SRI-NIC host. It can be obtained via FTP from your local host by connecting to host SRI-NIC.ARPA (26.0.0.73 or 10.0.0.51), logging in as user = ANONYMOUS, password = GUEST, and retrieving the file "NETINFO:HOSTS.TXT". The same table may also be obtained via the NIC Hostname Server, as described in RFC-953. The latter method is faster and easier, but requires a user program to make the necessary connection to the Name Server.

ASSUMPTIONS

1. A "name" (Net, Host, Gateway, or Domain name) is a text string up to 24 characters drawn from the alphabet (A-Z), digits (0-9), minus sign (-), and period (.). Note that periods are only allowed when they serve to delimit components of "domain style names". (See RFC-921, "Domain Name System Implementation Schedule", for background). No blank or space characters are permitted as part of a name. No distinction is made between upper and lower case. The first character must be an alpha character. The last character must not be a minus sign or period. A host which serves as a GATEWAY should have "-GATEWAY" or "-GW" as part of its name. Hosts which do not serve as Internet gateways should not use "-GATEWAY" and "-GW" as part of their names. A host which is a TAC should have "-TAC" as the last part of its host name, if it is a DoD host. Single character names or nicknames are not allowed.

2. Internet Addresses are 32-bit addresses [See RFC-796]. In the

host table described herein each address is represented by four decimal numbers separated by a period. Each decimal number represents 1 octet.

3. If the first bit of the first octet of the address is 0 (zero), then the next 7 bits of the first octet indicate the network number (Class A Address). If the first two bits are 1,0 (one,zero), then the next 14 bits define the net number (Class B Address). If the first 3 bits are 1,1,0 (one,one,zero), then the next 21 bits define the net number (Class C Address) [See RFC-943].

This is depicted in the following diagram:

```
+-+------------+--------------+--------------+--------------+
|0|  NET <-7-> |        LOCAL ADDRESS <-24->                |
+-+------------+--------------+--------------+--------------+

+---+----------+--------------+--------------+--------------+
|1 0|      NET  <-14->        | LOCAL ADDRESS <-16->        |
+---+----------+--------------+--------------+--------------+

+-----+--------+--------------+--------------+--------------+
|1 1 0|            NET  <-21->              | LOCAL ADDRESS|
+-----+--------+--------------+--------------+--------------+
```

4. The LOCAL ADDRESS portion of the internet address identifies a host within the network specified by the NET portion of the address.

5. The ARPANET and MILNET are both Class A networks. The NET portion is 10 decimal for ARPANET, 26 decimal for MILNET, and the LOCAL ADDRESS maps as follows: the second octet identifies the physical host, the third octet identifies the logical host, and the fourth identifies the Packet Switching Node (PSN), formerly known as an Interface Message Processor (IMP).

```
+-+------------+--------------+--------------+--------------+
|0|  10 or 26  |     HOST     | LOGICAL HOST |   PSN (IMP)  |
+-+------------+--------------+--------------+--------------+
```

(NOTE: RFC-796 also describes the local address mappings for several other networks.)

6. It is the responsibility of the users of this host table to translate it into whatever format is needed for their purposes.

7. Names and addresses for DoD hosts and gateways will be negotiated and registered with the DDN PMO, and subsequently with the NIC,

before being used and before traffic is passed by a DoD host. Names and addresses for domains and networks are to be registered with the DDN Network Information Center (HOSTMASTER@SRI-NIC.ARPA) or 800-235-3155.

The NIC will attempt to keep similar information for non-DoD networks and hosts, if this information is provided, and as long as it is needed, i.e., until intercommunicating network name servers are in place.

EXAMPLE OF HOST TABLE FORMAT

```
NET : 10.0.0.0 : ARPANET :
NET : 128.10.0.0 : PURDUE-CS-NET :
GATEWAY : 10.0.0.77, 18.10.0.4 : MIT-GW.ARPA,MIT-GATEWAY : PDP-11 :
          MOS : IP/GW,EGP :
HOST : 26.0.0.73, 10.0.0.51 : SRI-NIC.ARPA,SRI-NIC,NIC : DEC-2060 :
       TOPS20 :TCP/TELNET,TCP/SMTP,TCP/TIME,TCP/FTP,TCP/ECHO,ICMP :
HOST : 10.2.0.11 : SU-TAC.ARPA,SU-TAC : C/30 : TAC : TCP :
```

SYNTAX AND CONVENTIONS

; (semicolon)	is used to denote the beginning of a comment. Any text on a given line following a ';' is a comment, and not part of the host table.
NET	keyword introducing a network entry
GATEWAY	keyword introducing a gateway entry
HOST	keyword introducing a host entry
DOMAIN	keyword introducing a domain entry
:(colon)	is used as a field delimiter
::(2 colons)	indicates a null field
,(comma)	is used as a data element delimiter
XXX/YYY	indicates protocol information of the type TRANSPORT/SERVICE.

where TRANSPORT/SERVICE options are specified as

"FOO/BAR"	both transport and service known

"FOO"	transport known; services not known
"BAR"	service is known, transport not known

NOTE: See "Assigned Numbers" for specific options and acronyms for machine types, operating systems, and protocol/services.

Each host table entry is an ASCII text string comprised of 6 fields, where

Field 1	KEYWORD indicating whether this entry pertains to a NET, GATEWAY, HOST, or DOMAIN. NET entries are assigned and cannot have alternate addresses or nicknames. DOMAIN entries do not use fields 4, 5, or 6.
Field 2	Internet Address of Network, Gateway, or Host followed by alternate addresses. Addresses for a Domain are those where a Domain Name Server exists for that domain.
Field 3	Official Name of Network, Gateway, Host, or Domain (with optional nicknames, where permitted).
Field 4	Machine Type
Field 5	Operating System
Field 6	Protocol List

Fields 4, 5 and 6 are optional. For a Domain they are not used.

Fields 3-6, if included, pertain to the first address in Field 2.

'Blanks' (spaces and tabs) are ignored between data elements or fields, but are disallowed within a data element.

Each entry ends with a colon.

The entries in the table are grouped by types in the order Domain, Net, Gateway, and Host. Within each type the ordering is unspecified.

Note that although optional nicknames are allowed for hosts, they are discouraged, except in the case where host names have been changed and both the new and the old names are maintained for a suitable period of time to effect a smooth transition. Nicknames are not permitted for NET names.

GRAMMATICAL HOST TABLE SPECIFICATION

A. Parsing grammar

```
<entry> ::= <keyword> ":" <addresses> ":" <names> [":" [<cputype>]
   [":" [<opsys>]  [":" [<protocol list>] ]]] ":"
<addresses> ::= <address> *["," <address>]
<address> ::= <octet> "." <octet> "." <octet> "." <octet>
<octet> ::= <0 to 255 decimal>
<names> ::= <netname> | <gatename> | <domainname> *[","
   <nicknames>]
   | <official hostname> *["," <nicknames>]
<netname>  ::= <name>
<gatename> ::= <hname>
<domainname> ::= <hname>
<official hostname> ::= <hname>
<nickname> ::= <hname>
<protocol list> ::= <protocol spec> *["," <protocol spec>]
<protocol spec> ::= <transport name> "/" <service name>
   | <raw protocol name>
```

B. Lexical grammar

```
<entry-field> ::= <entry-text> [<cr><lf> <blank> <entry-field>]
<entry-text>  ::= <print-char> *<text>
<blank> ::= <space-or-tab> [<blank>]
<keyword> ::= NET | GATEWAY | HOST | DOMAIN
<hname> ::= <name>*["."<name>]
<name>  ::= <let>[*[<let-or-digit-or-hyphen>]<let-or-digit>]
<cputype> ::= PDP-11/70 | DEC-1080 | C/30 | CDC-6400...etc.
<opsys>   ::= ITS | MULTICS | TOPS20 | UNIX...etc.
<transport name> ::= TCP | NCP | UDP | IP...etc.
<service name> ::= TELNET | FTP | SMTP | MTP...etc.
<raw protocol name> ::= <name>
<comment> ::= ";" <text><cr><lf>
<text>    ::= *[<print-char> | <blank>]
<print-char>  ::= <any printing char (not space or tab)>
```

Notes:

1. Zero or more 'blanks' between separators " , : " are allowed. 'Blanks' are spaces and tabs.

2. Continuation lines are lines that begin with at least one blank. They may be used anywhere 'blanks' are legal to split an entry across lines.

BIBLIOGRAPHY

1. Feinler, E., Harrenstien, K., Su, Z. and White, V., "Official DoD Internet Host Table Specification", RFC-810, Network Information Center, SRI International, March 1982.

2. Harrenstien, K., Stahl, M., and Feinler, E., "Hostname Server", RFC-953, Network Information Center, SRI International, October 1985.

3. Kudlick, M. "Host Names Online", RFC-608, Network Information Center, SRI International, January 1973.

4. Postel, J., "Internet Protocol", RFC-791, Information Sciences Institute, University of Southern California, Marina del Rey, September 1981.

5. Postel, J., "Address Mappings", RFC-796, Information Sciences Institute, University of Southern California, Marina del Rey, September 1981.

6. Postel, J., "Domain Name System Implementation Schedule", RFC-921, Information Sciences Institute, University of Southern California, Marina del Rey, October 1984.

7. Reynolds, J. and Postel, J., "Assigned Numbers", RFC-943, Information Sciences Institute, University of Southern California, Marina del Rey, April 1985.

Network Working Group J. Postel
Request for Comments: 678 (SRI-ARC)
NIC: 31524 19 December 1974

Standard File Formats

Introduction

In an attempt to provide online documents to the network community we have had many problems with the physical format of the final documents. Much of this difficulty lies in the fact that we do not have control or even knowledge of all the processing steps or devices that act on the document file. A large part of the difficulty in the past has been due to some assumptions we made about the rest of the world being approximately like our own environment. We now see that the problems are due to differing assumptions and treatment of files to be printed as documents. We therefore propose to define certain standard formats for files and describe the expected final form for printed copies of such files.

These standard formats are not additional File Transfer Protocol data types/modes/structures, but rather usage descriptions between the originator and ultimate receiver of the file. It may be useful or even necessary at some hosts to construct programs that convert files between common local formats and the standard formats specified here.

The intent is that the author of a document may prepare his/her text and store it in an online file, then advertise that file by name and format (as specified here), such that interested individuals may copy and print the file with full understanding of the characteristics of the format controls and the logical page size.

Standardization Elements

The elements or aspects of a file to be standardized are the character or code set used, the format control procedures, the area of the page to be used for text, and the method to describe overstruck or underlined characters.

The area of the page to be used for text can be confusing to discuss, in an attempt to be clear we define a physical page and a logical page. Please note that the main emphasis of this note is to describe the standard formats in terms of the logical page, and that it is up to each site to map the logical page onto the physical page of each of their devices.

Physical Page

The physical page is the medium that carries the text, the height and width of its area are measured in inches.

The typical physical page is a piece of paper eleven inches high and eight and one half inches wide.

Typical print density is 10 characters per inch horizontally and 6 characters per inch vertically. This results in the typical physical page having a maximum capacity of 66 lines and 85 characters per line. It is often the case that printing devices limit the area of the physical page by enforcing margins.

Logical Page

The logical page is the area that can contain text, the height of this area is measured in lines and the width is measured in characters.

A typical logical page is 60 lines high and 72 characters wide.

Code Set

The character encoding will be the network standard Network Virtual Terminal (NVT) code as used in Telnet and File Transfer protocols, that is ASCII in an eight bit byte with the high order bit zero.

Format Control

The format will be controlled by the ASCII format effectors:

Form Feed <FF>

Moves the printer to the top of the next logical page keeping the same horizontal position.

Carriage Return <CR>

Moves the printer to the left edge of the logical page remaining on current line.

Line Feed <LF>

Moves the printer to the next print line, keeping the same horizontal position.

Horizontal Tab <HT>

Moves the printer to the next horizontal tab stop.

The conventional stops for horizontal tabs are every eight characters, that is character positions 9, 17, 25, ... within the logical page.

Note that it is difficult to enforce these conventions and it is therefore recommended that horizontal tabs not be used in document files.

Vertical Tab <VT>

Moves the printer to the next vertical tab stop.

The conventional stops for vertical tabs are every eight lines starting at the first printing line on each logical page, that is lines 1, 9, 17, ... within the logical page.

Note that it is difficult to enforce these conventions and it is therefore recommended that vertical tabs not be used in document files.

Back Space <BS>

Moves the printer one character position toward the left edge of the logical page.

Not all these effectors will be used in all format standards, any effectors which are not used in a format standard are ignored.

Page Length

The logical page length will be specified in terms of a number of lines of text.

Page Width

The logical page width will be specified as a number of characters.

Overstriking

Overstriking (note that underlining is a subset of overstriking) may be specified to be done in one or both of the following ways, or not at all:

By Line

The composite line is made up of text segments each terminated by the sequence <CR><NUL> except that the final segment is terminated by the sequence <CR><LF>.

By Character

Each character to be overstruck is to be immediately followed by a <BS> and the overstrike character.

End of Line

The end of line convention is the Telnet end of line convention which is the sequence <CR><LF>. It is recommended that use of <CR> and <LF> be avoided in other than the end of line context.

Standard Formats

Format 1 [Basic Document]

This format is designed to be used for documents to be printed on line printers, which normally have 66 lines to a physical page, but often have forced top and bottom margins of 3 lines each.

```
Active Format Effectors
   <FF>, <CR>, <LF>.
Page Length
   60 lines.
Page Width
   72 Characters.
Overstriking
   By Line.
```

Format 2 [Terminal]

This format is designed to be used with hard copy terminals, which in the normal case have 66 lines to a physical page. It is expected that there are no top or bottom margins enforced by the terminal or its local system, thus any margins around the physical page break must come from the file.

```
Active Format Effectors
   <FF>, <CR>, <LF>, <HT>, <VT>, <BS>.
Page Length
   66 lines.
Page Width
   72 Characters.
Overstriking
   By Character.
```

Format 3 [Line Printer]

This format is designed to be used with full width (11 by 14 inch paper) line printer output.

Active Format Effectors
<FF>, <CR>, <LF>.
Page Length
60 lines.
Page Width
132 Characters.
Overstriking
None.

Format 4 [Card Image]

This format is designed to be used for simulated card input. The page width is 80 characters, each card image is followed by <CR><LF>, thus each card is represented by between 2 and 82 characters in the file. Note that the trailing spaces of a card image need not be present in the file, and that the early occurence of the <CR><LF> sequence indicates that the remainder of the card image is to contain space characters.

Active Format Effectors
<CR>, <LF>.
Page Length
Infinite.
Page Width
80 Characters.
Overstriking
None.

Format 5 [Center Document]

This format is intended for use with documents to be printed on line printers which normally have 66 lines to the physical page but enforce top and bottom margins of 3 lines each. The text is expected to be centered on the paper. If the horizontal printing density is 10 characters per inch and the paper is 8 and 1/2 inches wide then there will be a one inch margin on each side.

Active Format Effectors
<FF>, <CR>, <LF>.
Page Length
60 Lines.
Page Width
65 Characters.
Overstriking
By Line.

Format 6 [Bound Document]

This format is intended for use with documents to be printed on line printers which normally have 66 lines to the physical page but enforce top and bottom margins of 3 lines each. If the horizontal printing density is 10 characters per inch and the paper is 8 and 1/2 inches wide then the text should be positioned such that there is a 1 and 1/2 inch left margin and a one inch right margin.

Active Format Effectors
<FF>, <CR>, <LF>.
Page Length
60 Lines.
Page Width
60 Characters.
Overstriking
By Line.

Implementation Suggestions

Overflow

Overflow can result from two causes, first if the physical page is smaller than the logical page, and second if the actual text in the file violates the standard under which it is being processed.

In either case the following suggestions are made to implementors of programs which process files in these formats.

Length

If more lines are processed than fit within the minimum of the physical page and the logical page length since the last <FF>, then the <FF> action should be forced.

Width

If more character positions are processed than fit on the minimum of the physical page width and the logical page width since the last <CR>, then characters are discarded up to the next <CR>.

or

If more character positions are processed than fit on the minimum of the physical page width and the logical page width since the last <CR>, then the <CR> and <LF> actions should be forced.

References

A. McKenzie "TELNET Protocol Specification," Aug-73, NIC 18639.

"USA Standard Code for Information Interchange," United States of America Standards Institute, 1968, NIC 11246.

INSTRUCTIONS FOR AUTHORS OF RFCs

RFCs are distributed online by being stored as public access files, and a short messages is sent to the distribution list indicating the availability of the memo.

The online files are copied by the interested people and printed or displayed at their site on their equipment. This means that the format of the online files must meet the constraints of a wide variety of printing and display equipment.

To meet these constraints the following rules are established for the format of RFCs:

The character codes are ASCII.

Each page must be limited to 58 lines followed by a form feed on a line by itself.

Each line must be limited to 72 characters followed by carriage return and line feed.

No overstriking (or underlining) is allowed.

These "height" and "width" constraints include any headers, footers, page numbers, or left side indenting.

Each RFC is to include on its title page or in the first or second paragraph a statement (titled "Status of this Memo") describing the intention of the RFC. There are several reasons for publishing a memo as an RFC, for example, to make available some information for interested people, or to begin or continue a discussion of an interesting idea, or to make available the specification of a protocol.

The following sample paragraphs may be used to satisfy this requirement:

Specification

This RFC specifies a standard for the DARPA Internet community. Hosts on the ARPA-Internet are expected to adopt and implement this standard.

Discussion

The purpose of this RFC is to focus discussion on particular problems in the ARPA-Internet and possible methods of solution. No proposed solutions this document are intended as standards for the ARPA-Internet. Rather, it is hoped that a general consensus will emerge as to the appropriate solution to such problems, leading eventually to the adoption of standards.

Information

This RFC is being distributed to members of the ARPA-Internet community in order to solicit their reactions to the proposals contained in it. While the issues discussed may not be directly relevant to the research problems of the ARPA-Internet, they may be interesting to a number of researchers and implementers.

Status

In response to the need for maintenance of current information about the status and progress of various projects in the ARPA-Internet community, this RFC is issued for the benefit of community members. The information contained in this document is accurate as of the date of publication, but is subject to change. Subsequent RFCs will reflect such changes.

Of course these paragraphs need not be followed word for word, but the general intent of the RFC must be made clear.

Each RFC is to also include a "distribution statement". In general RFCs have unlimited distribution. There may be a few cases in which it is appropriate to restrict the distribution in some way.

Typically the distribution statement will simply be the sentence "Distribution of this memo is unlimited." appended to the "status of this memo" section.

Network Working Group
Request for Comments: 797

A. Katz
ISI
September 1981

FORMAT FOR BITMAP FILES

This note describes a proposed format for storing simple bitmaps (one bit per pixel) in a file. These files may be very large and the intent is to use this format for short term storage and passing data between closely coupled programs. The data in the file should be stored in 8-bit bytes (octets). Bitmaps may be any size.

The first 4 octets of the file gives the width of each line (x direction), and the next 4 octets should give the number of lines of the display (length, y direction). After this is one octet for the x increment and one octet for the y increment. Following these 10 octets is the bitmap itself. The length and width fields are stored most significant octet first.

The x and y increment octets tell how much space is between pixels. For an ordinary display, both these would be one.

Each line of the display should be scanned from left to right. The lines should start at the top and work down. Each line in the bitmap should end on an octet boundary. If the width of the display is not divisable by 8, the rest of the last octet should be filled with zeros on the right.

Below is a representation of a bitmap file (each square is one octet):

```
-----------------------------------------------------
|    1    |    2    |    3    |    4     |    5     |
|  width  |  width  |  width  |  width   |  length  |
-----------------------------------------------------

-----------------------------------------------------
|    6    |    7    |    8    |    9     |    10    |
|  length |  length |  length |x-increment|y-increment|
-----------------------------------------------------

-----------------------------------------------------
|   11    |   12    |   13    |    14    |   15
|  data   |  data   |  data   |   data   |  data...
-----------------------------------------------------
```

For example, bitmaps from the RAPICOM 450 can be in Fine Detail, Quality, or Express Mode. In Fine Detail mode the x-increment and y-increment would be 1, for Quality mode, the x-increment would be 1 and the y-increment would be 2, and for Express mode, the x-increment would be 1 and the y-increment would be 3. For these bitmaps it is intended that each scan line be repeated y-increment times when they are displayed.

Network Working Group
Request for Comments: 769

J. Postel
ISI
26 September 1980

Rapicom 450 Facsimile File Format

Introduction:

Several organizations in the ARPA Internet community have RAPICOM 450 facsimile machines interfaced to computers. This allows these organizations to enter a facsimile representation of a page into a computer file, and to produce a page from stored facsimile data. These organizations can exchange stored facsimile data via file transfer and other protocols. The purpose of this note is to document the format used for these files so that other organizations with compatible facsimile devices can join in this information exchange procedure.

The Rapicom 450:

The Rapicom 450 has a built in encoding/decoding scheme. It produces data blocks of 585 bits. There are "set up" blocks and "data" blocks. The machine sends/receives several copies of the set up block, but since they are identical only one set up block is stored in the file.

Records:

Each 585 bit block is placed in a record of 8-bit bytes. The record format is a length byte, a command byte and the data bytes. Each record is an integral number of bytes. The length value includes the length byte and the command byte. The command describes the data in the data field.

```
0        1        2        3                              length
+--------+--------+--------+--------+---//---+--------+--------+
| length | command|   data                                     |
+--------+--------+--------+--------+---//---+--------+--------+
```

Rapicom 450 Facsimile Record

Commands:

56 - SET-UP

The command code 56 (70 octal) indicates the following data field is a set up block.

57 - DATA

The command code 57 (71 octal) indicates the following data field is a data block.

58 - END

The command code 58 (72 octal) indicates that this is the last record in the file. In this case the length may be 2, indicating that there is no data in this record.

Conventions:

In the files exchanged to date, each record contains one block. This means the data field is 74 bytes long (585/8=73.125), and the length field has the value 76 (114 octal), except the last record which may carry no data and have a length of 2.

The first record of a file is always a SET UP record, the following records are DATA records, until the last record which is an END record.

Details:

The 585 bit data block is encoded by the Rapicom 450 and so can not be used a bit map unless the encoding/decoding procedure is known and used.

The first 24 bits of the block is always a synchronization mark with the value 271 141 344 in octal or 101110010110000111100100 in binary.

The low order two bits of the next byte contain a sequence number (modulo 4). The sequence number bits cycle in the order 11, 01, 10, 00, starting with the first DATA record (not the SET UP record).

The line below represents a DATA record, where L represents a length bit, C represents a command bit, M represents the synchronization mark, S represents a sequence bit, F represents a fill bit, the dash represents 68 other data octets, and an D represents a data bit.

```
LLLLLLLLCCCCCCCCMMMMMMMMMMMMMMMMMMMMMMMMDDDDDDSSDDDDDDDD-DFFFFFFF
```

In the line below the normal values have been filled in for the length, the command, the synchronization mark and fill bits.

```
0100110000111001101110010110000111100100DDDDDDSSDDDDDDDD-D0000000
```

USA Standard Code for Information Interchange

1. Scope

2. Standard Code

b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	ROW ↓ / COLUMN →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	−	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

3. Character Representation and Code Identification

The standard 7-bit character representation, with b_7 the high-order bit and b_1 the low-order bit, is shown below:

EXAMPLE: The bit representation for the character "K," positioned in column 4, row 11, is

b_7	b_6	b_5	b_4	b_3	b_2	b_1
1	0	0	1	0	1	1

The code table position for the character "K" may also be represented by the notation "column 4, row 11" or alternatively as "4/11." The decimal equivalent of the binary number formed by bits b_7, b_6, and b_5, collectively, forms the column number, and the decimal equivalent of the binary number formed by bits b_4, b_3, b_2, and b_1, collectively, forms the row number.

The standard code may be identified by the use of the notation ASCII or USASCII.

The notation ASCII (pronounced as'-key) or USASCII (pronounced you-sas'-key) should ordinarily be taken to mean the code prescribed by the latest issue of the standard. To explicitly designate a particular (perhaps prior) issue, the last two digits of the year of issue may be appended, as, "ASCII 63" or "USASCII 63".

4. Legend

4.1 Control Characters

NUL	Null	DLE	Data Link Escape (CC)
SOH	Start of Heading (CC)	DC1	Device Control 1
STX	Start of Text (CC)	DC2	Device Control 2
ETX	End of Text (CC)	DC3	Device Control 3
EOT	End of Transmission (CC)	DC4	Device Control 4 (Stop)
ENQ	Enquiry (CC)	NAK	Negative Acknowledge (CC)
ACK	Acknowledge (CC)	SYN	Synchronous Idle (CC)
BEL	Bell (audible or attention signal)	ETB	End of Transmission Block (CC)
BS	Backspace (FE)	CAN	Cancel
HT	Horizontal Tabulation (punched card skip) (FE)	EM	End of Medium
LF	Line Feed (FE)	SUB	Substitute
VT	Vertical Tabulation (FE)	ESC	Escape
FF	Form Feed (FE)	FS	File Separator (IS)
CR	Carriage Return (FE)	GS	Group Separator (IS)
SO	Shift Out	RS	Record Separator (IS)
SI	Shift In	US	Unit Separator (IS)
		DEL	Delete[1]

NOTE: (CC) Communication Control
(FE) Format Effector
(IS) Information Separator

[1]In the strict sense, DEL is not a control character. (See 5.2.)

4.2 Graphic Characters

Column/Row	Symbol	Name
2/0	SP	Space (Normally Non-Printing)
2/1	!	Exclamation Point
2/2	"	Quotation Marks (Diaeresis[2])
2/3	#	Number Sign[3,4]
2/4	$	Dollar Sign
2/5	%	Percent
2/6	&	Ampersand
2/7	'	Apostrophe (Closing Single Quotation Mark; Acute Accent[2])
2/8	(	Opening Parenthesis
2/9	)	Closing Parenthesis
2/10	*	Asterisk
2/11	+	Plus
2/12	,	Comma (Cedilla[2])
2/13	-	Hyphen (Minus)
2/14	.	Period (Decimal Point)
2/15	/	Slant
3/10	:	Colon
3/11	;	Semicolon
3/12	<	Less Than
3/13	=	Equals
3/14	>	Greater Than
3/15	?	Question Mark
4/0	@	Commercial At[3]
5/11	[	Opening Bracket[3]
5/12	\	Reverse Slant[3]
5/13	]	Closing Bracket[3]
5/14	^	Circumflex[2,3]
5/15	_	Underline
6/0	`	Grave Accent[2,3] (Opening Single Quotation Mark)
7/11	{	Opening Brace[3]
7/12	¦	Vertical Line[3]
7/13	}	Closing Brace[3]
7/14	~	Overline[3] (Tilde[2]; General Accent[2])

[2] he use of the symbols in 2/2, 2/7, 2/12, 5/14, 6/0, and 7/14 as dia ritical marks is described in Appendix A, AS.2.

[3] These characters should not be used in international interchange without determining that there is agreement between sender and recipient. (See Appendix B4.)

[4] In applications where there is no requirement for the symbol #, the symbol £ may be used in position 2/3.

REPORT NO. 1822

INTERFACE MESSAGE PROCESSOR

Specifications for the Interconnection of a Host and an IMP

Developed for
the Advanced Research Projects Agency
by Bolt Beranek and Newman Inc.

bbn

Report No. 1822 Bolt Beranek and Newman Inc.

SPECIFICATIONS FOR THE INTERCONNECTION OF A HOST AND AN IMP

December 1983 Revision

Prepared for:

Defense Communications Agency

12/83

TABLE OF CONTENTS

LIST OF FIGURES

*Photographs by Hutchins Photography, Inc., Belmont, Mass.

1. INTRODUCTION

The ARPANET provides a capability for geographically separated computers, called Hosts, to communicate with each other The Host computers typically differ from one another in type, speed, word length, operating system, etc. Each Host computer is connected into the network through a local small computer, called an Interface Message Processor (IMP); a typical network section is shown in Figure 1-1. The complete network is formed by interconnecting these IMPs through wideband communication lines supplied by common carriers. Each IMP is then programmed to store and forward messages to the neighboring IMPs in the network. During a typical operation, a Host passes a message to its IMP; this message is then passed from IMP to IMP through the network until it finally arrives at the destination IMP, which in turn passes it along to the destination Host.

Several models of IMPs are currently in existence. All perform the basic function of a store and forward mode, but they have different physical configurations and data handling rates. The Model 516 (see Figure 1-2) is the original IMP. The Model 316 (see Figure 1-3) is a less expensive and somewhat slower version of the original IMP. The 316 Terminal IMP or TIP (see Figure 1-4) is a Model 316 IMP mounted in a double hi-boy rack along with a BBN Multi-Line Controller (MLC) The 316 Terminal

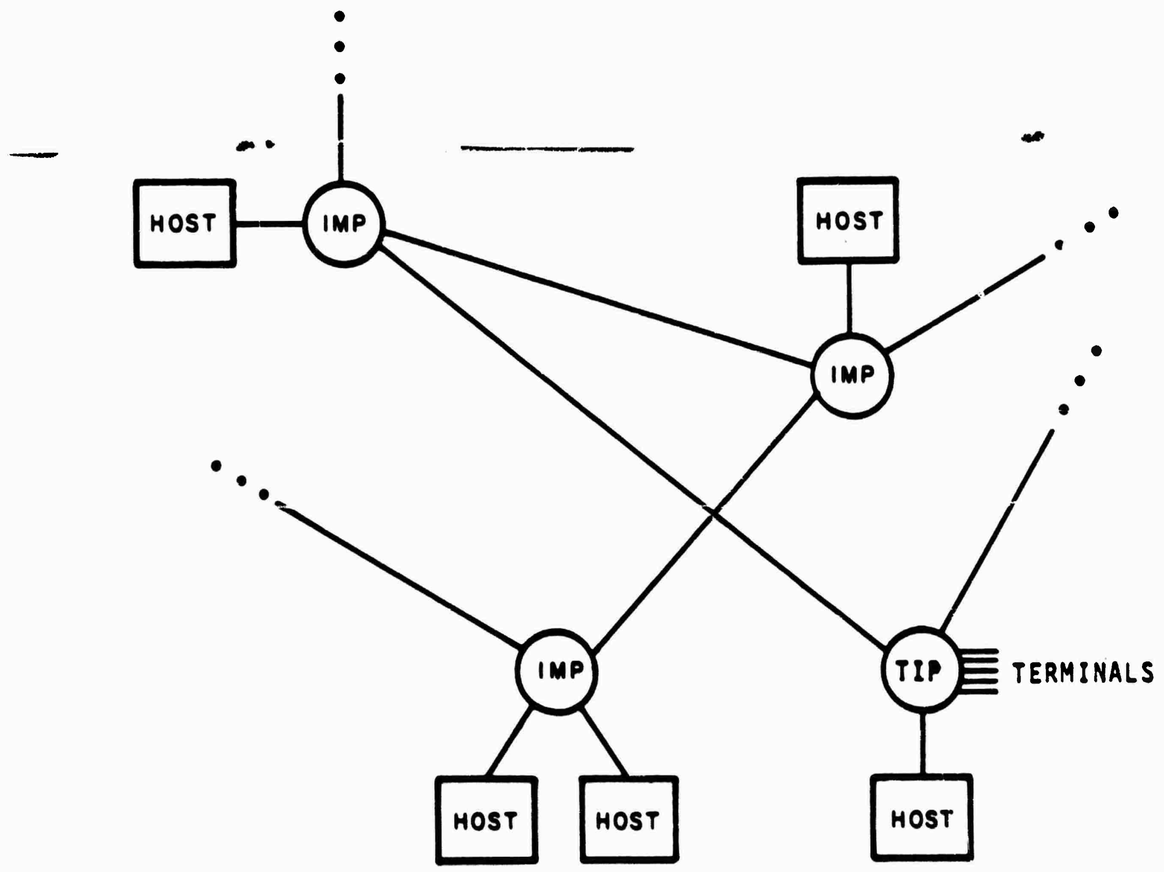


Figure 1-1 A Typical Section of the ARPANET

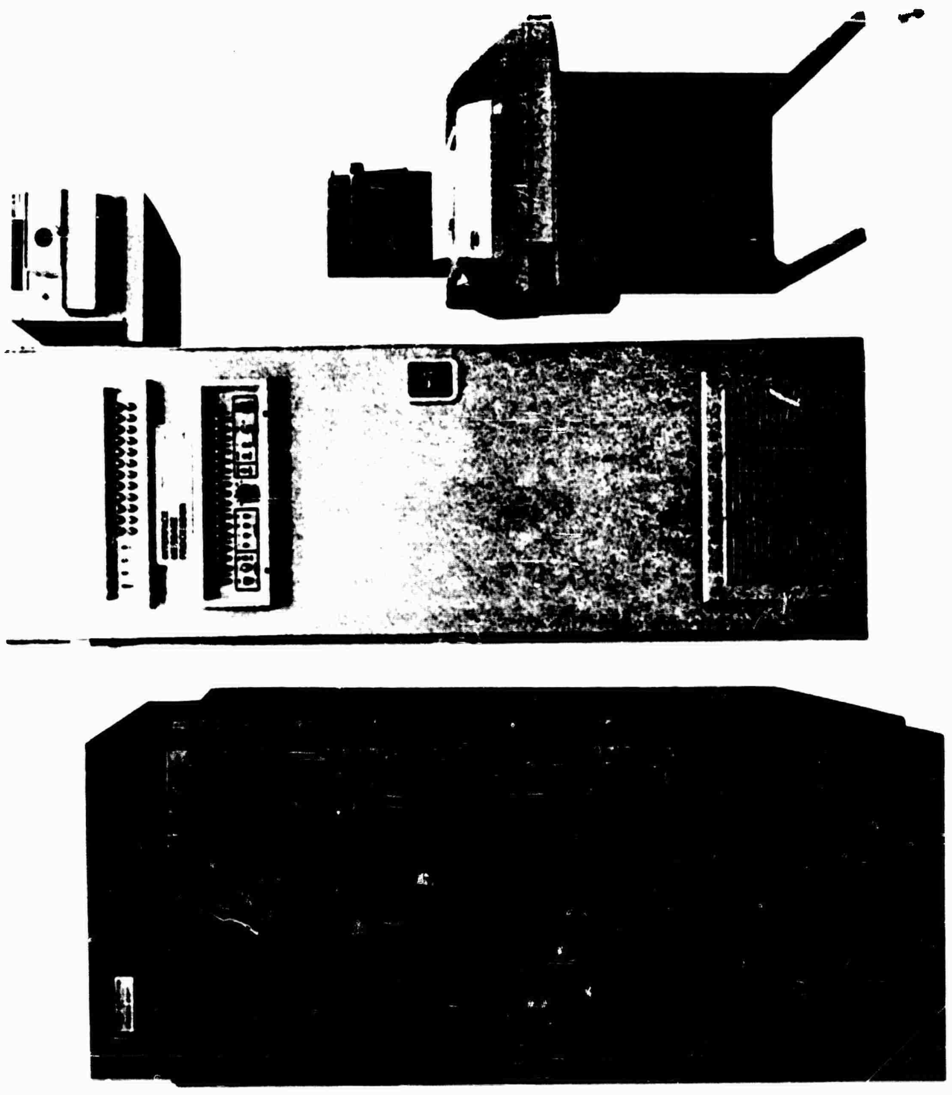


Figure 1-2 The Model 516 IMP, Modem Cabinet, and IMP Teletype

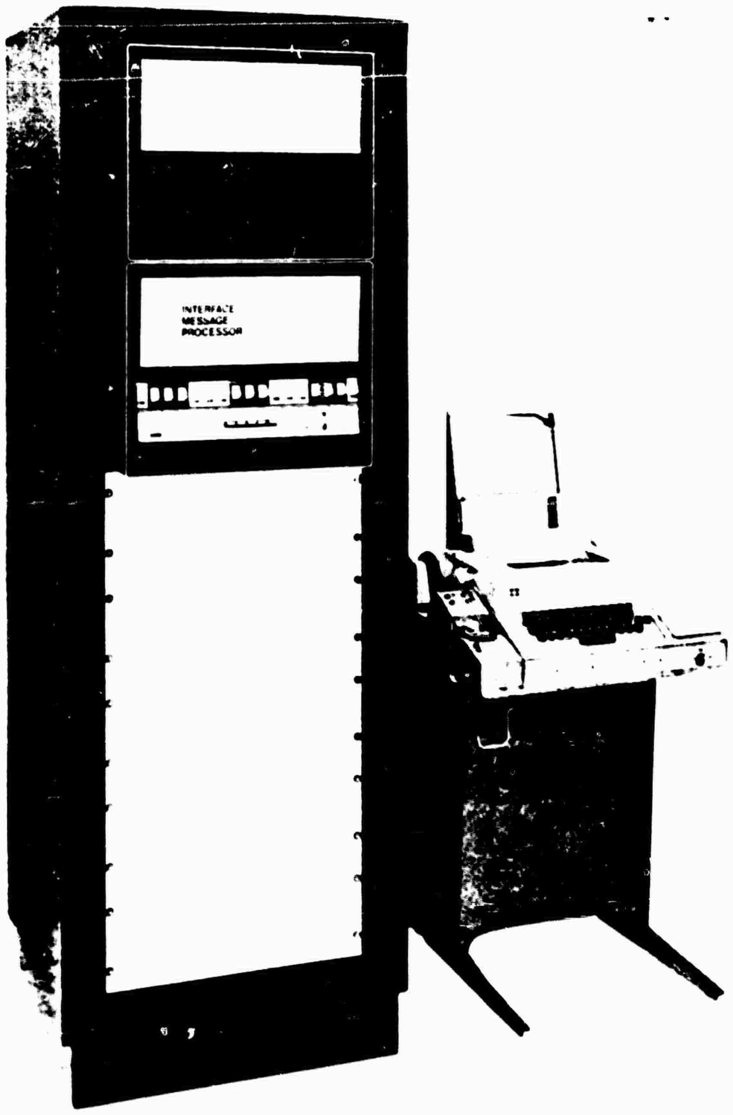


Figure 1-3 The Model 316 IMP and IMP Teletype

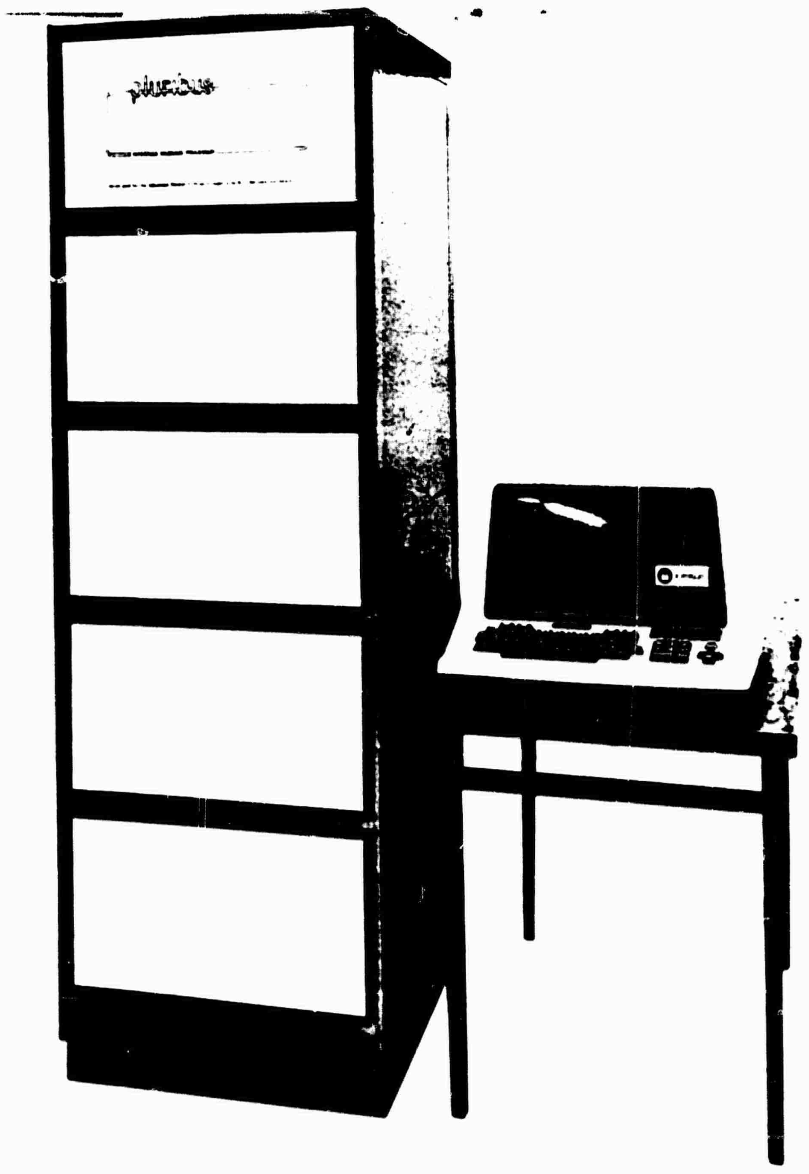


Figure 1-4 The Model 316 Terminal IMP and IMP Teletype

IMP is designed to connect both Hosts and up to 63 terminals to the (316 and 516) network; the terminals are given access to the network directly, without an intervening Host. The Honeywell-based IMPs and TIP are no longer being manufactured, but are occasionally re-deployed within the ARPANET.

The Pluribus IMP (see Figure 1-5) is based on a flexible multiprocessor design and is housed in from one to several (see Figure 1 6) racks, depending on precise speed and capacity. A Terminal IMP is also available in Pluribus form, and it can provide access to a much larger number of terminals than the 316 Terminal IMP. A front-end Pluribus system, called the Private Line Interface (see Appendix H) is available and provides a variety of alternative interfacing arrangements to the network.

The C/30 IMP, the most recent addition to the IMP family, is based on modern microprocessor technology and provides an inexpensive and flexible replacement for the 316 and 516 IMP series. Information on installation of and connection to the C/30 IMP is given in Appendix L.

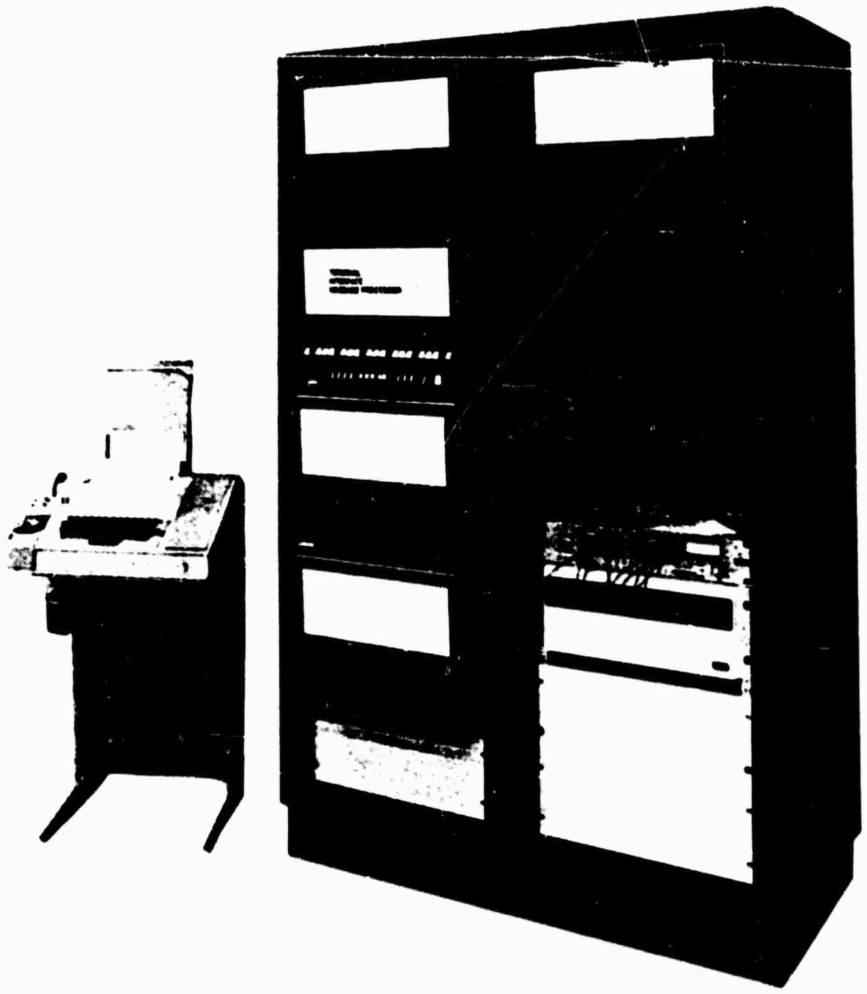


Figure 1-5 The Pluribus IMP and IMP Terminal

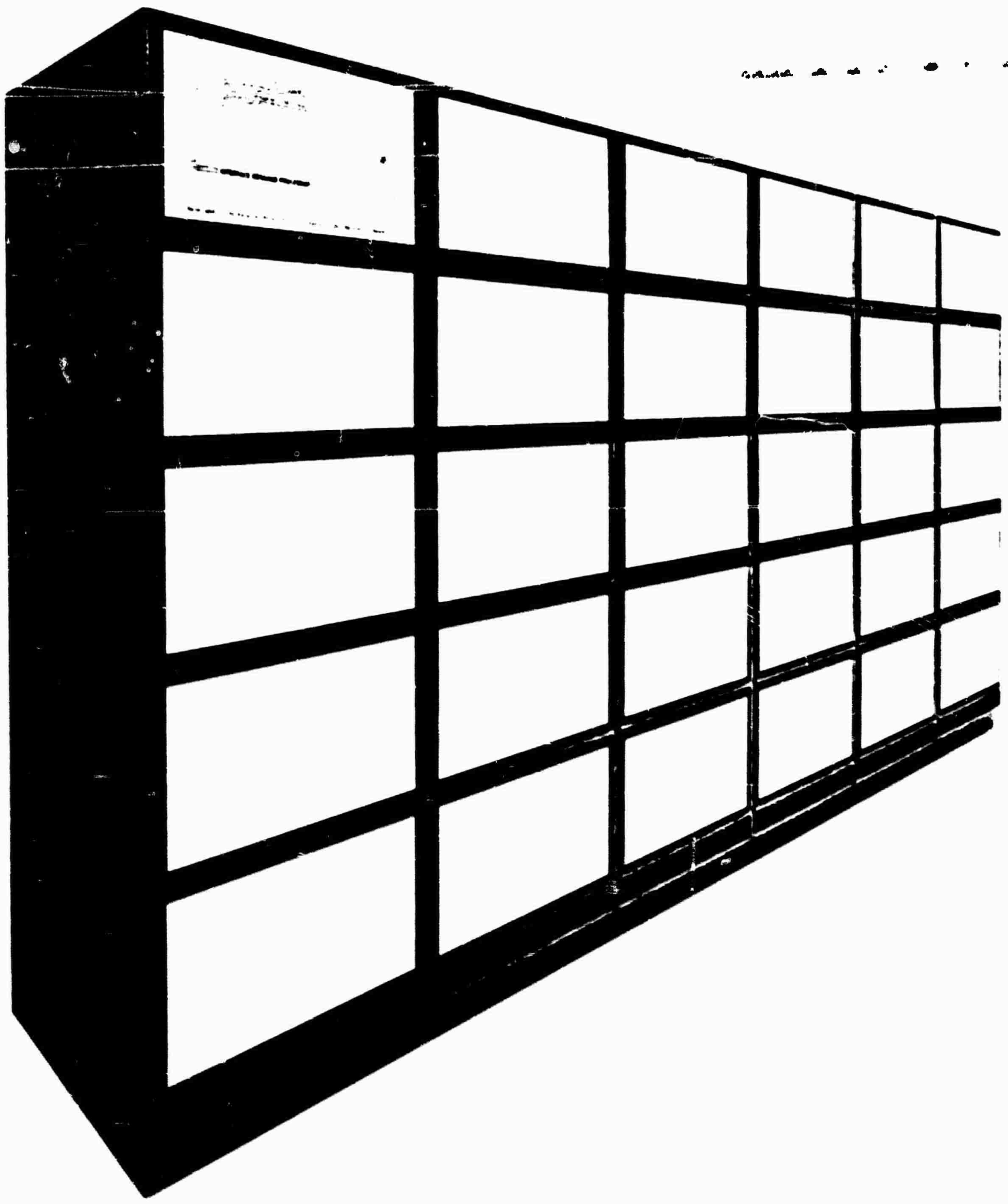


Figure 1-6 Pluribus TIP Configured to Support 378 Terminals

This document contains the specifications for interconnecting a Host and an IMP and may be subject to change. The interconnection of a Host and an IMP is a joint effort that requires the Host personnel to provide interfacing hardware and software. Although we have tried to provide sufficient information to assist the Host personnel in the design of the interface, problems and questions that we have not anticipated will undoubtedly arise. These questions should be addressed to:

Network Control Center
Bolt Beranek and Newman Inc.
10 Moulton Street
Cambridge, Massachusetts 02138

We strongly recommend that the personnel responsible for the design of the Host hardware and software interfaces visit in Cambridge with the technical staff of Bolt Beranek and Newman Inc. for a thorough review of the designs prior to implementation. We feel that this procedure will help to minimize the difficulties that will be encountered in connecting the Host and the IMP.

2. GENERAL REQUIREMENTS

In this section, we describe the physical configuration of the IMP, the space and power requirements, the equipment necessary to interconnect the IMP and Host and the facilities that must be provided by the IMP site to assist with installation and maintenance of the IMP.

2.1 Physical Configuration

As shown in Figure 2-1, four pieces of equipment are provided: the IMP itself, which is a modified Honeywell H-516R, Honeywell H-316, BBN Pluribus computer or BBN C/30 computer; an ASR-33 Teletype or Infoton Vistar;* a high-speed paper tape reader (optional); and a cabinet, approximately the same size as the Model 516R, that contains modems connecting the IMP to the communication lines. The telephone company will supply modems only for the communication lines actually installed. In addition, the telephone company usually supplies auxiliary equipment that may vary from site to site and need not be located near the modem cabinet or the IMP.

A Host is connected to an IMP by a Host cable.** The particular cabling scheme is determined by the distance between

*The Vistar is a keyboard/display type terminal used with the Pluribus. It performs the same functions as the ASR-33 Teletype.
**The cables in Figure 2-1 are drawn only schematically rather than in their actual positions.

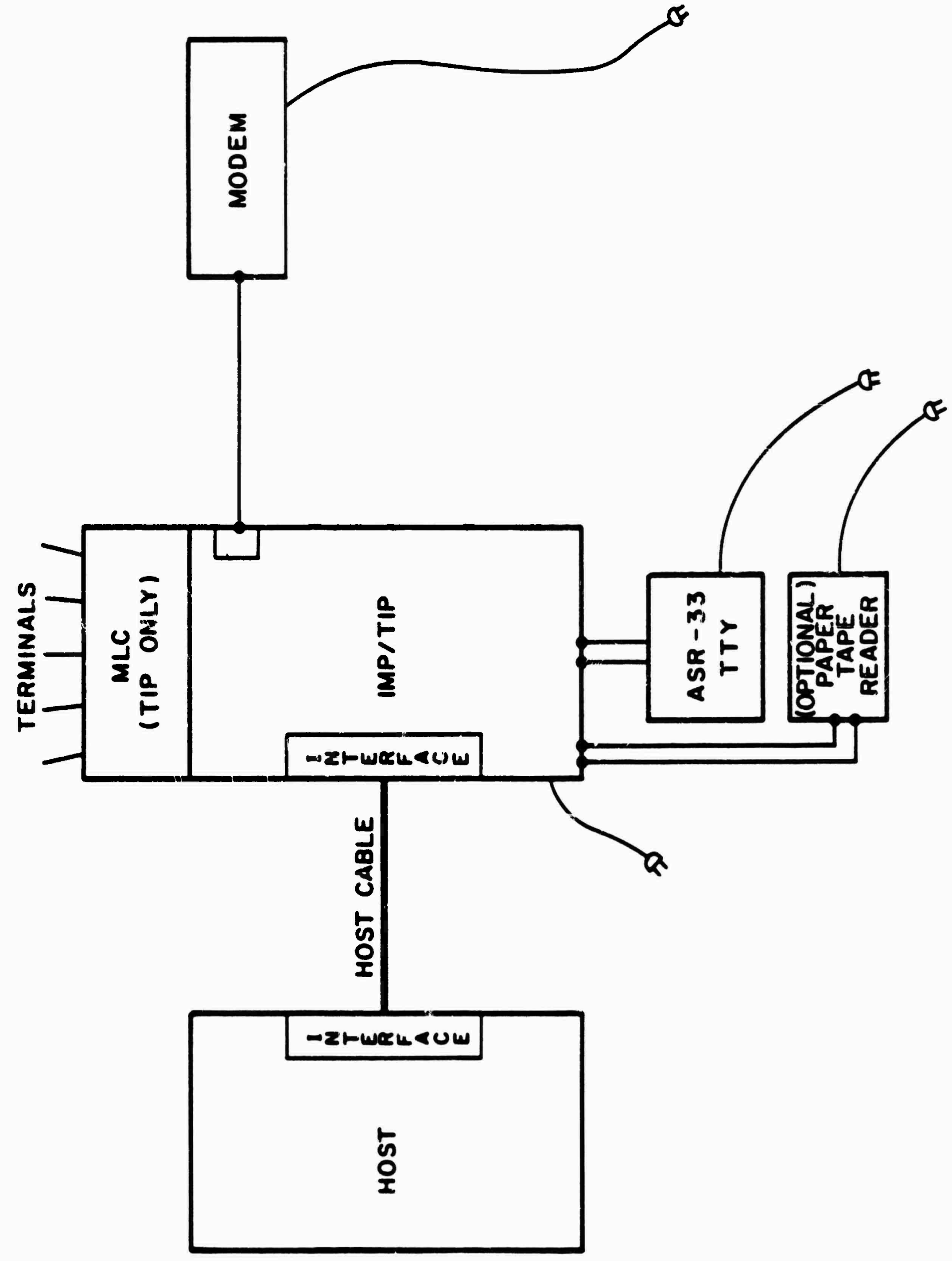


Figure 2-1 IMP Equipment

the Host and the IMP. A local Host (one close to the IMP) is connected by a 30-foot cable* that is supplied with the IMP. This cable connects a standard Host/IMP interface unit built into the IMP to a special interface provided by the Host.

A distant Host may be located up to 2000 feet from the IMP, but an addition to the standard Host/IMP interface is required to modify the line-driving scheme. The Host personnel must design a special interface that is compatible and must supply the connecting cable as specified in Section 4.5.2. Since additional IMP hardware must be supplied, the decision to connect a distant Host must be made known well in advance. A distant Host will usually be connected to an IMP which has one or more local Hosts.

A very distant Host may be located even farther from the IMP, using an entirely different interface arrangement which is described in Appendix F. Basically, the very distant Host interface is designed for use over communication circuits with speeds up to 230.4 kilobits/second and up to tens (perhaps hundreds) of miles long. The communication protocol used with this interface includes a 24-bit cyclic redundancy check and a positive acknowledgment scheme.

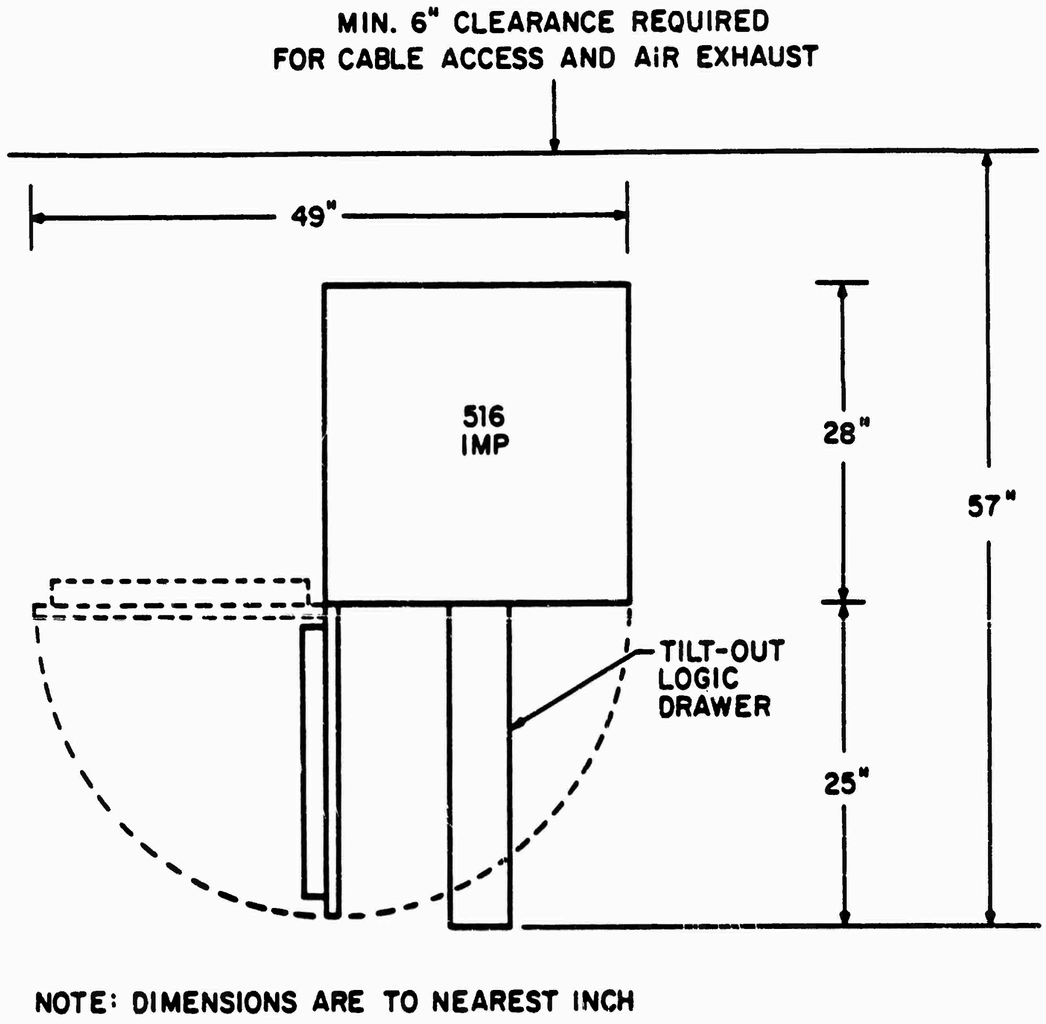
*The length of this cable is limited by the characteristics of the cable drivers in the IMP.

A separate 30-foot cable is provided with the IMP for the connection to each modem. In addition, cables are provided for connecting the terminal (Teletype or Vistar) and paper tape reader (if supplied) to the IMP. For the H-516R and H-316 IMPs, cables exit from the IMP through the bottom of the rear panel. Cables will exit from the modem unit through the bottom of the modem cabinet; if a site does not have a false floor, other modem cable arrangements are easily provided. Cables are connected to the Pluribus IMP via a fantail panel located at the rear of the machine.

Figures 2-2, 2-3, 2-4, and 2-5 depict the floor space requirements for the 516 IMP, the 316 IMP, the (maximum size) 316 TIP, and the (minimum size) Pluribus IMP respectively. Some configurations of the 316 TIP may only require the same floor space as a 316 IMP, and some Pluribus IMPs may require several racks side by side; the Network Control Center can furnish details for each installation.

With the Honeywell machines, provision should be made to place the ASR-33 Teletype close to the IMP. The ASR-33 occupies approximately 2' x 2' of floor space. (The optional paper tape reader must be placed nearby if it is supplied.* Its dimensions are 11 x 11 x 23 inches (WIDTH x HEIGHT x DEPTH). A convenient

*To determine whether a paper tape reader will be supplied, a site may contact the Network Control Center.


MIN. 6" CLEARANCE REQUIRED
FOR CABLE ACCESS AND AIR EXHAUST
49"
516
IMP
28"
57"
TILT-OUT
LOGIC
DRAWER
25"
NOTE: DIMENSIONS ARE TO NEAREST INCH

TOP VIEW

Figure 2-2 Minimum Floor Area Required for 516 IMP

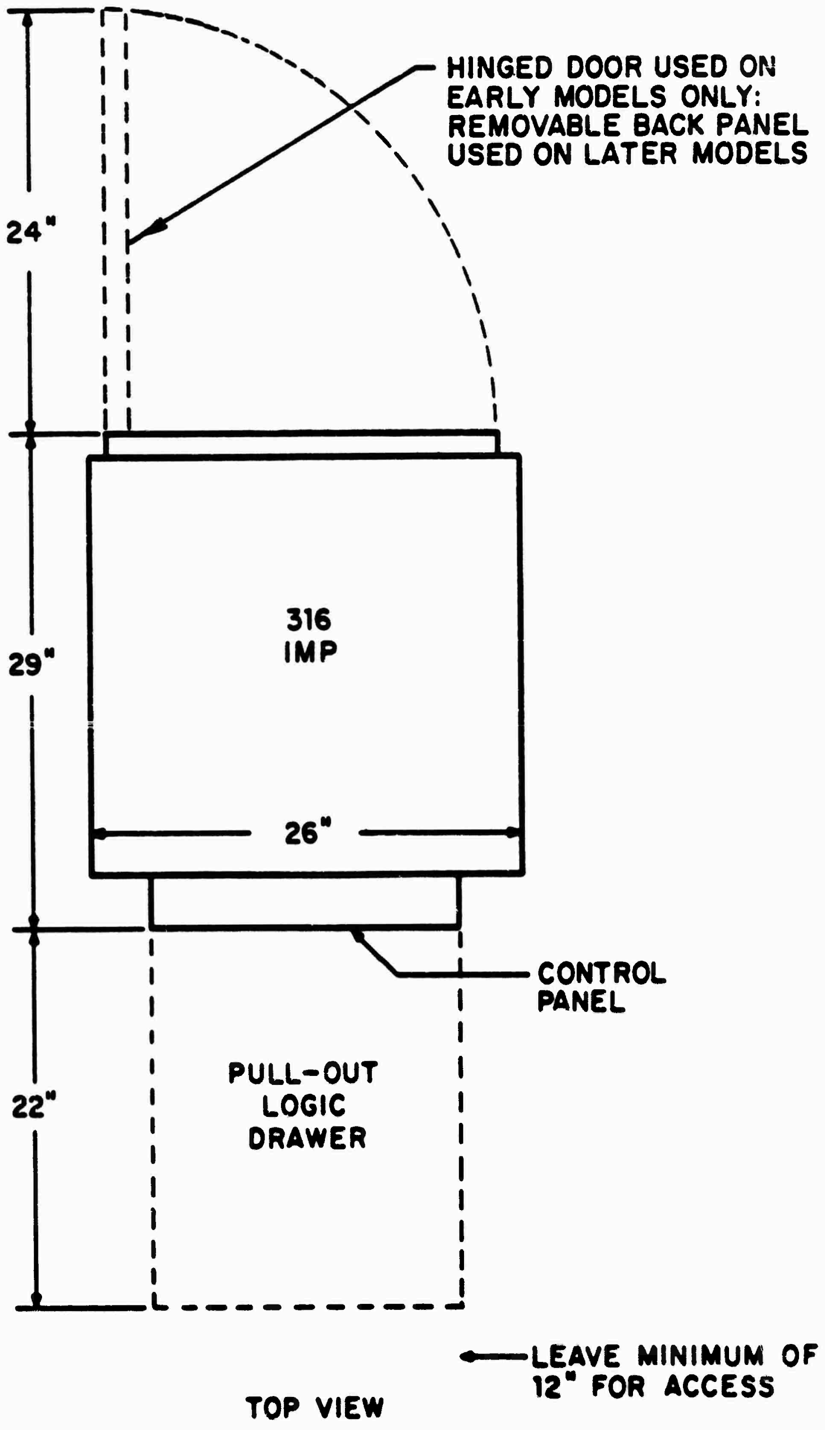

HINGED DOOR USED ON EARLY MODELS ONLY: REMOVABLE BACK PANEL USED ON LATER MODELS
24"
29"
316 IMP
26"
CONTROL PANEL
22"
PULL-OUT LOGIC DRAWER
LEAVE MINIMUM OF 12" FOR ACCESS
TOP VIEW

Figure 2-3 Minimum Floor Area Required for 316 IMP

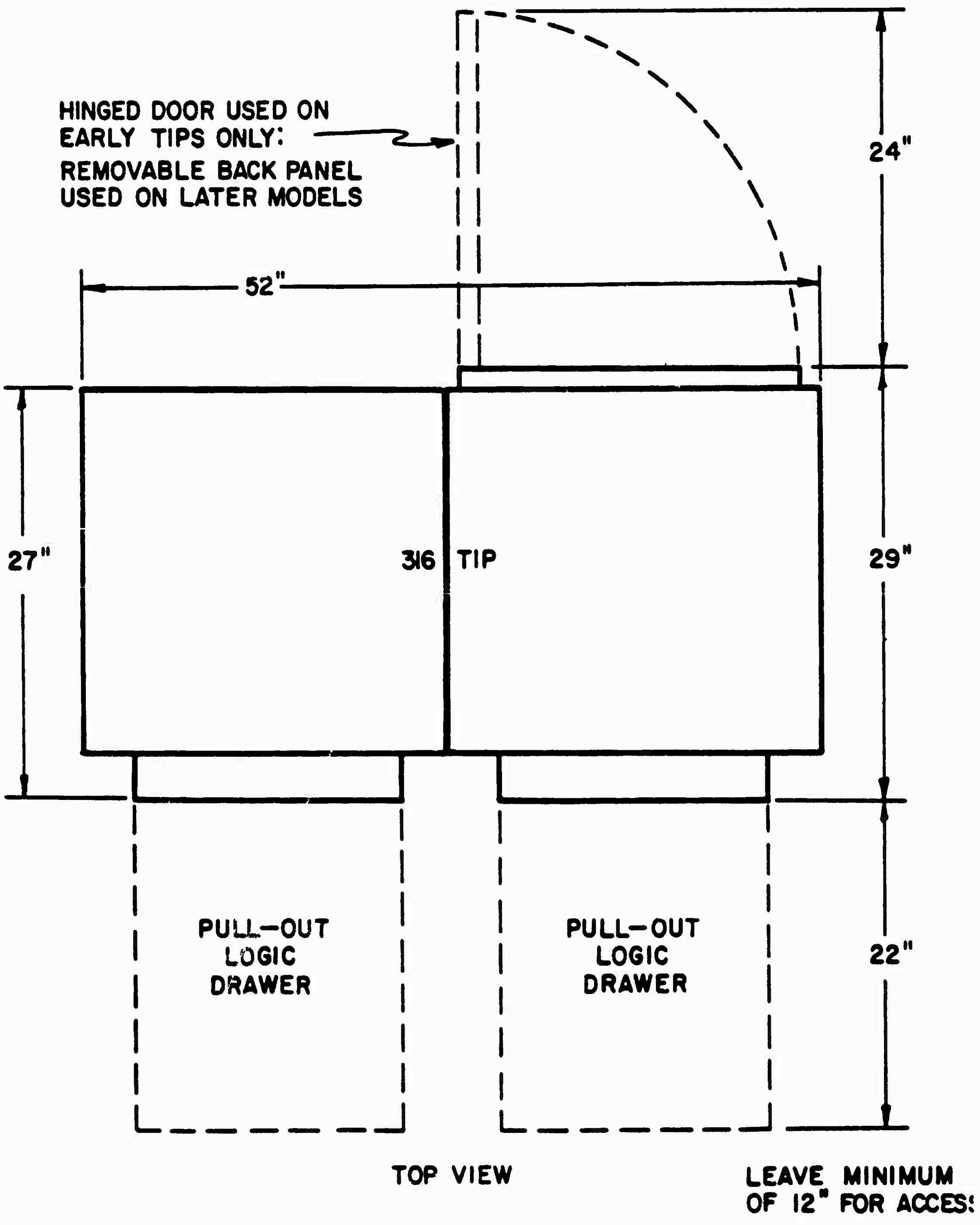
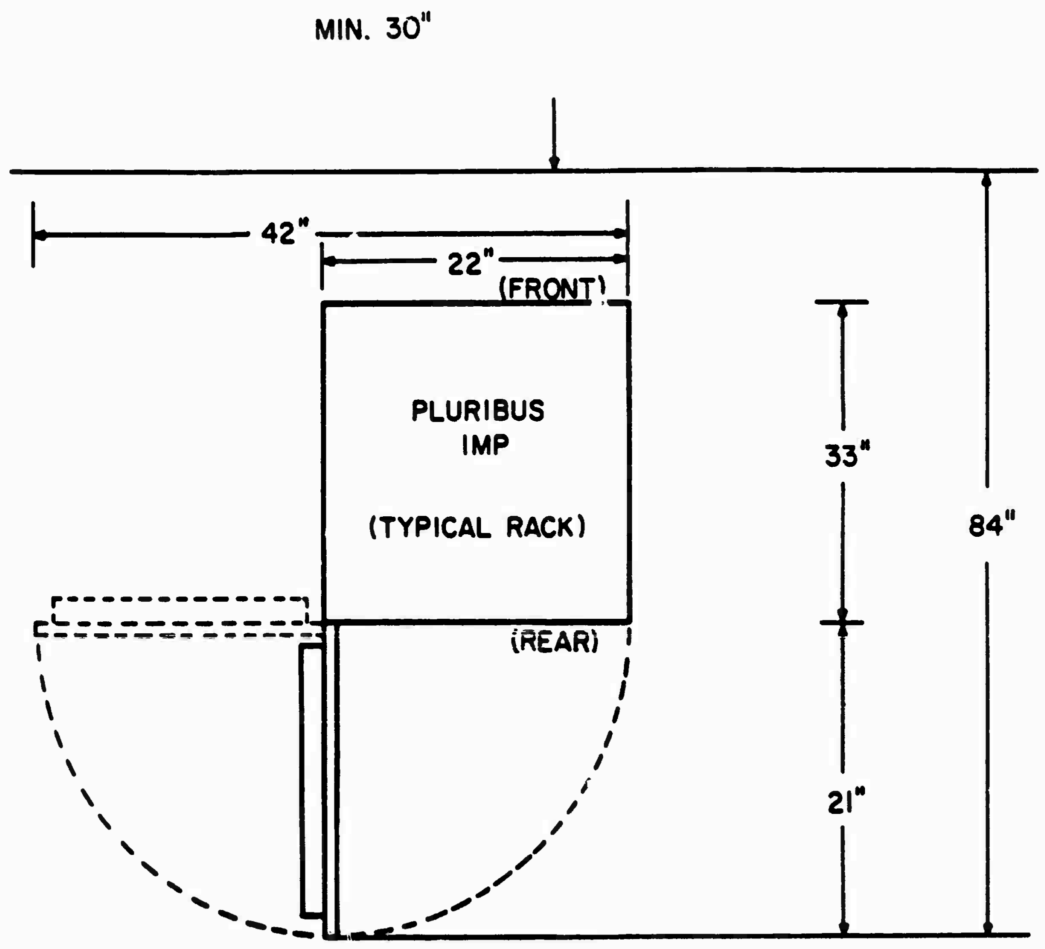


Figure 2-4 Minimum Floor Area Required for 316 TIP



NOTES: DIMENSIONS ARE TO NEAREST INCH;
CABLES CONNECT ON REAR (DOOR) SIDE

TOP VIEW

Figure 2-5 Minimum Floor Area Required for Pluribus IMP (Per Rack)

location is the top of the IMP cabinet, if overhead space permits.) With the Pluribus machine, table space should be provided nearby for the Infoton Vistar. Its dimensions are 20 x 13 x 24 inches. (Again, the optional paper tape reader must be placed nearby if it is supplied.* Its dimensions are 20 x 8 x 22 inches. It can be located on top of the IMP cabinet if overhead space permits.)

A small lockable cabinet is needed on the Host premises for the storage of IMP-related materials (e.g., manuals, test tapes, scope, tool box, etc.). Finally, a telephone should be located within reach of both the terminal and the operating panel of the IMP for use during diagnosis and debugging.

The locations of the IMP, modem cabinet, paper tape reader, and Teletype are to be selected by the Host personnel. These pieces of equipment should be placed within approximately eight feet of one another. A minimum of thirty square feet of floor space is required for the equipment, and additional space must be available for accessing the machine during maintenance and debugging. Access to the Model 516 IMP is via a full-length front door, which is hinged on the left side. Access to the 316 IMPs is via drawers which slide to the front. Access to the Pluribus IMP is via full-length rear doors and removable front

*To determine whether a paper tape reader will be supplied, a site may contact the Network Control Center.

panels. Access to the modem cabinet is via a removable front panel.

In addition to the modem cabinet, the telephone company may provide another cabinet to contain the auxiliary equipment. It is recommended that this auxiliary equipment be placed in an inconspicuous location on the Host premises, such as in a telephone company equipment room, since immediate access to this equipment is not necessary.

2.2 Description of Equipment

External dimensions, approximate weights, and power requirements of the various IMP models are given in Table 2-1. The paper tape reader weighs approximately 25 pounds, the ASR-33 Teletype weighs approximately 56 pounds, and the Infoton Vistar weighs approximately 55 pounds.

The Model 516 IMP is a ruggedized unit with EMI protection. All IMPs will operate in an ambient environment from 17 to 30 degrees centigrade and up to 95% humidity. However, these features have been included for reliability and, in general, an environment suitable for most digital computing equipment should be provided; i.e., air-conditioned and free from excessive dust and moisture.

Model	Size (inches)			Weight (lbs)	Power (watts)
	Height	Width	Depth		
516R IMP	74	24	28	990	2100
316 IMP	73	26	28	525	750
316 TIP	73	52	28	920	2200
expansion cabinet	39	25	28	100	0
Pluribus IMP (per rack)	68	22	26	550	3000 (approx)

Table 2-1

The power requirements for the Honeywell IMP equipment are as follows:

a) <u>IMP</u>: 115 VAC ± 10%; 60 Hz ± 5%, single phase. The line cord is 15 ft. long and contains 3-wire cable terminated by a 30-amp Hubbell 3331G (NEMA L5-30P) twistlock connector (for wiring convention, see Appendix G).

b) <u>High-speed reader</u> (optional): 115 VAC ± 10%; 60 Hz, single-phase at 125 watts. (The line must withstand 10-amp surges at 125 VAC.) The line cord is 6 ft. long and is terminated in a standard 3-wire grounded plug.

c) <u>ASR-33</u>: 115 VAC ± 10%; 60 Hz ± 0.45 Hz, single phase at 230 watts. The line cord is 8ft. long and is terminated in a standard 3-wire grounded plug.

Power for the Pluribus equipment is supplied via one 3-phase 208/110 volt wye 60 Hz connection per rack. Each power cord is 20 feet long and is terminated by a Hubbell 2811 (NEMA L21-30P) twistlock connector. Each circuit must supply 30 amps per leg. Sufficient convenience outlets for debugging equipment, the Infoton Vistar, and paper tape reader are provided on the Pluribus itself.

The Host must provide an appropriate power receptacle (located within 15 feet) for the IMP power plug and it is recommended that a separate fuse or circuit breaker be provided on the IMP's power line. (The Honeywell IMP normally draws about 20 amps, but the line must be capable of supplying up to 30 amps.) The IMP's chassis is connected to the ground (third) lead of the power plug, which is completely isolated from the signal return (i.e., "signal ground"). If at all feasible, the power to the IMP should be provided from the same transformer that delivers power to the Host in order to insure a common ground. For Honeywell equipment, three 115-VAC wall sockets (located within 5 feet of the IMP) are required to power the Teletype, paper tape reader, and an IMP debugging oscilloscope used during installation and maintenance. The line for these sockets should be fused for 20 amps and should be powered from the same transformer as the IMP, if feasible.

The modem cabinet dimensions are 68-1/8" x 28" x 28"; it weighs up to 750 lbs and requires up to 15 amps of standard 115 VAC power. The modem operates in an ambient environment of 40 degrees to 120 degrees Fahrenheit and up to 95% humidity. The Host must provide power for the modem from the same transformer that delivers power to the IMP. A standard 3-connector non-locking, non-twist plug is normally provided with the modem. The telephone company also recommends that a separate fuse or circuit breaker be provided on the power line to the modem. (The auxiliary equipment is a non-standard item that will vary from site to site; the size is generally no larger than the size of the modem cabinet and may be as small as a 2' x 3' wall mounting. A separate power outlet will also be needed for this equipment.)

In all, the Honeywell equipment requires six receptacles, and Pluribus machines require one receptacle per rack plus one for the modem cabinet. The site should plan to provide the power necessary for the phone company equipment after preliminary discussions with the local telephone company representatives and before the circuit installation date.

2.3 Interfacing

The Host/IMP interface is subdivided into two separate units, as illustrated in Figure 2-6.

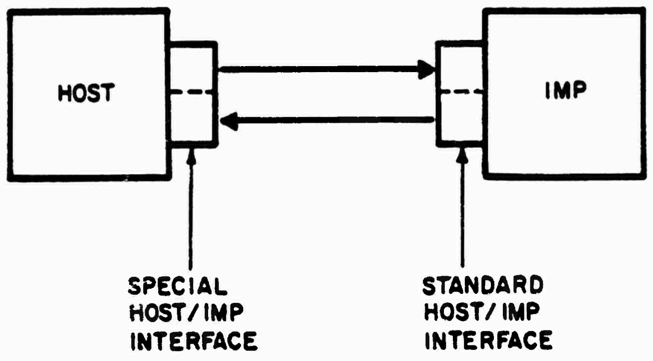


Figure 2-6 Host/IMP Interface

The right-hand (standard) unit is built into the IMP and contains logic that is standard for all Host/IMP interfaces. The left-hand unit contains the special equipment for interfacing directly to the particular Host. An addition to the standard Host/IMP interface is required for a distant Host. Standard signals pass on the host cable between these two halves; all special logic and signal adjustments (which vary from Host to

Host) are handled in the left-hand portion. Each participating Host will be responsible for the design and construction of its own special unit to mate to the standard Host/IMP interface unit. The logical operation of this unit will be the same, regardless of whether a Host is local or distant; however, a different electrical signaling scheme is required to handle a distant Host. A detailed description of the requirements for the special unit is given in Section 4. The very distant Host interface follows the same general philosophy of a standard interface unit at the IMP end and a special interface unit at the Host end, but uses a completely different signaling scheme as described in Appendix F. Still another Host interfacing scheme, making use of the Private Line Interface (PLI), is described in Appendix H.

The Host computer and the IMP communicate by transmitting messages over the Host cable. The format for this communication has been established and is described in Section 3. Each Host is responsible for providing the necessary Network Control Program in the Host computer.

An IMP test program is available for use during installation and testing. In addition to checking various functions in the IMP, this program provides a mechanism for checkout of the Host's special interface. The program repeatedly transmits a message to the Host, a copy of which it expects the Host to return with any Host padding, or data (Section 3.5). The Host should plan to

<u>provide an appropriate test program to operate in conjunction with this IMP test program</u>.

3. SYSTEM OPERATION

3.1 Messages and Message-ids

Hosts communicate with each other via <u>regular messages</u>. A regular message may vary in length from 96 up to 8159 bits, the first 96 of which are control bits called the <u>leader</u>. The leader is also used for sending control messages between the Host and its IMP, in which case only the first 80 bits are used. The remainder of the message is the <u>data</u>, or the <u>text</u>.

For each regular message, the Host specifies a <u>destination</u>, consisting of IMP, Host, and <u>handling type</u>. These three parameters uniquely specify a <u>connection</u> between source and destination Hosts. The handling type gives the connection specific characteristics, such as priority or non-priority transmission (see below). Additional leader space has been reserved for a fourth parameter, to be used in future inter-network addressing. For each connection, messages are delivered to the destination in the same order that they were transmitted by the source.

For each regular message, the Host also specifies a 12-bit identifier, the <u>message-id</u>.* The message-id, together with the destination of the message, is used as the "name" of the message.

*Until mid-1973 the first eight bits of the message-id field were called the "link".

The IMP will use this name to inform the Host of the disposition of the message. Therefore, if the Host refrains from re-using a particular message-id value (to a given destination) until the IMP has responded about that message-id, messages will remain uniquely identified and the Host can retransmit them in the event of a failure within the network.

After receiving a regular message from a Host connected to it, an IMP breaks the message into several packets (currently the maximum data bits/packet is 1008) and passes these through the network in the direction of the destination. Eventually, when all packets arrive at the destination, they are reassembled to form the original message and passed to the destination Host. The destination IMP returns a positive acknowledgment for receipt of the message to the source IMP, which in turn passes this acknowledgment to the source Host. This acknowledgment is called a Ready for Next Message (RFNM) and identifies the message being acknowledged by name. In some relatively rare cases, however, the message may be lost in the network due to an IMP failure; in such cases an Incomplete Transmission message will be returned to the source Host instead of a RFNM. Again, in this case, the message which was incompletely transmitted is identified by name.

If a response from the destination IMP (either RFNM or Incomplete Transmission) is itself lost in the network, this condition will be detected by the source IMP, which will automatically inquire of the destination IMP whether the original message was correctly transmitted or not, and repeat the inquiry until a response is received from the destination IMP. This inquiry mechanism is timeout-driven, and each timeout period may be as little as 30 or as much as 45 seconds in length.

When a message arrives at its destination, the leader is modified to indicate the source Host, but the message-id field is passed through unchanged. Thus, in addition to providing message identification between a Host and its local IMP, the message-id can provide a means for Hosts to identify messages between themselves. For example, the message-id can be used for multiplexing several independent data streams, or for keeping track of the portions of a single data stream being sent "in parallel" through the network.

If the priority bit of the handling type is set, the message will be expedited through the network by being placed at the front of the various transmission queues it will encounter along the way. This can be useful for transactions requiring minimal delay (e.g., remote echoing or the exchange of control information) but should be used judiciously, since the more it is used the less effect each further use will have.

In order to prevent various types of deadlocks within the network, a source IMP must guarantee that the destination IMP will have enough storage to accept the message it is about to send. This is done by preceding each message with a short "request for buffer space" message. When the destination has enough buffer space to receive another message, it returns an "allocation" to the source IMP, which can then send the message it has been holding.

There are several situations in which an IMP may temporarily block* the transmission of a message from the source Host to the source IMP. In general, any such blockage will last for only a few milliseconds, but in some cases the blockage may be indefinite. In at least one such case the IMP will be unable to accept the remainder of a message from its Host until it frees buffer space by delivering some message to the Host (it is for this reason that half-duplex Host-IMP interfaces are prohibited). In all such cases, in order to prevent permanently hanging up transmission between the Host and the IMP, the source IMP will discard the message after a wait of about fifteen seconds and return a type 9 (sub-type 4) message (see Section 3.4) to the Host, thus limiting the length of time that the interface will be blocked. Similarly, once a Host has begun to send the IMP a message, it must be prepared to deliver the entirety of that message to the IMP promptly. In particular, the IMP will discard any message that is not completely received from its Host in

*By failing to provide Ready-for-Next-Bit, see Section 4.1.

fifteen seconds and return a type 9 (sub-type 2) message to the Host (see Section 3.4).

One situation under which interface blocking will occur is when the source IMP must wait to receive an allocation from the destination IMP. Since a Host cannot send other messages into the network while its interface is blocked, it is desirable to expedite the "allocation" mechanism, and this is done in two different ways depending upon message length. For one-packet messages, the message itself is sent as its own request. Thus, if space is available, the message is immediately accepted and no additional delay is incurred. For multi-packet messages, when the destination IMP is about to return a RFNM it reserves storage in <u>anticipation</u> of the source Host's next message, and returns the allocation along with the acknowledgment. Thus, when the source IMP eventually sends its Host the RFNM, it is also implicitly informing it of the allocation now being available.* If the Host responds promptly with another message on that same connection (message-id is irrelevant), the message can be forwarded immediately, avoiding any set-up delay waiting for an allocation. If this allocation remains unused for about 125 ms, it is returned, unused, to the destination. Note that this

*In some (rare) cases the destination is unable to reserve storage immediately, and returns a RFNM without the reservation. Currently, the destination waits 1/2 second, attempting to reserve storage, before returning the RFNM without an accompanying reservation.

mechanism applies only for messages longer than one packet (about 1103 bits, including leader).

The message processing (reassembly of packets into messages, allocation of buffer space, detection of lost messages, etc.) requires the IMP to perform a certain amount of bookkeeping on the flow of messages between each pair of communicating Hosts. In order to keep the amount of required table space within manageable bounds, the following two restrictions are imposed.

1. The maximum number of messages which a Host is permitted to have "in transit" on any connection is eight. In other words, if a Host attempts to transmit nine messages on any connection, the interface will be blocked by the IMP during transmission of the ninth message until a RFNM (or Incomplete Transmission) is returned for the first message. However, this rule does not prohibit one Host from having eight messages in transit to Host "A", eight more in transit to Host "B", etc., simultaneously.

2. When a Host wishes to establish a new connection with another Host, both source and destination IMPs must acquire a block of table space from a pool of such blocks shared by all the Hosts local to each IMP. The source IMP must notify the destination of the need for

the new connection, and the destination must reply with a confirmation that it has also acquired the table space. This action may result in a small additional delay before Host communication can begin. The pool will be sufficiently large to seldom interfere with a pair of Hosts wishing to communicate. In no case will Hosts be prevented from communicating because of lack of these resources. In the event that the Hosts on an IMP desire to simultaneously communicate with so many other Hosts that the pool would be exhausted, the space in the pool is quickly multiplexed in time among all the desired Host/Host conversations so that none is stopped although all are possibly slowed.

Section 3.7 describes an optional mechanism available to Hosts that wish to keep interface blocking to a minimum.

3.2 Establishing and Breaking Host/IMP Communications

Each IMP and Host interface has its own hardware Ready indicator. The Ready indicator in the standard Host/IMP interface will be on whenever the IMP is powered on and both the IMP program and the IMP hardware are determined to be working properly. The Ready indicator in the special Host interface should be on whenever the Host is powered on, the hardware is working properly, and the Host's Network Control Program (NCP) is running. If the Host temporarily neglects communications with the IMP, the Host's hardware Ready indicator should not go off. An off indication should mean only that something is broken or that communications have been willfully cut off for an extended period (cable removed, power shut off, routine maintenance programs running, batch processing with no network program running etc.).

In addition to the Ready indicator, the standard interface has a flip-flop, called the Error flip-flop, which remembers a not-ready indication from the Host or the IMP. This flip-flop is used to detect any momentary off condition on either the Host's ready line or the IMP's ready line. The flip-flop is cleared by the IMP program each time the program enables (i.e., prepares to receive) a new input from the Host and is tested by the program when the input is completed. The input is discarded if the Error flip-flop is turned on.

To establish communication, a Host should simply send its message to the IMP. The operational IMP program will process any message transmitted from the Host. The Host must always send at least three NOP messages* to the IMP whenever either the Host or the IMP Ready line is turned on, for the reasons described below.

*See Section 3.3

One reason is that the Host-to-IMP NOP message contains information as to how much leader padding is to be contained in regular Host-to-IMP and IMP-to-Host messages. Also, until old-style leader formats (Appendix A) are no longer used, this NOP informs the IMP of the style of leader the Host is using.

Another reason is that in general, when the Host Ready indicator goes off, the IMP program will be either receiving or waiting (in an input command) to receive a message from the Host. Upon resumption of transmission by the Host, the IMP will unwittingly append the new information to the unfinished input. Upon completion of the message, the IMP program will note that the Error flip-flop is on and thus discard the entire message. To guarantee that a useful new message is not thereby discarded, the first message sent by the Host after its Ready indicator comes on should be a discardable NOP message. The special interface should have a similar Error flip-flop, and the Host's Network Control Program should be designed to use this flip-flop in a similar manner.

When the Host Ready indicator comes on, it will generally alternate a few times between on and off (due to relay contact bounce -- see Section 4.4) before setting solidly on. The Host should delay an appropriate period to permit its ready indicator to stabilize before starting output or preparing for input. Failure to do so may cause incorrect data to be taken from or sent to the IMP.

A Host may go down, thus halting network traffic to itself from other Hosts, in either of two ways: by turning off its ready indicator (hard down), or by failing to accept messages from the IMP (tardy down). In either case, the IMP will mark the Host as dead and see to it that any attempt to communicate with the Host results in a Destination Dead response.

The IMP program tests the Host Ready indicator (not the Error flip-flop) every half-second. If the program ever finds this ready indicator off, the Host will be marked dead (hard down) and the IMP will discard old messages for transmission to the Host and will set up 3 _NOP_ messages followed by a type 10 message for transmission to the Host. Both the IMP and the Host must discard any _NOP_ messages that are recognized as such. (A _NOP_ message that is appended to an unfinished message may not be recognized, but it will be discarded as discussed above.)

The IMP follows the above procedures when the Host Ready indicator is off momentarily or for an extended period. The following steps are taken by the IMP when its own indicator has gone off.

1. The Error flip-flop is turned on. This action will cause the first incoming message from the Host to be discarded.

2. Old messages for transmission to the Host are discarded.

3. The IMP Ready indicator is turned on.

4. Sufficient NOP messages are placed on the output queue to the Host to cover the period of relay bounce and insure correct transmission of at least one NOP.

5. A Type 10 message is placed on the output queue to the Host.

The Host should employ a similar procedure whenever its own Ready indicator has gone off, except that old messages for transmission to the IMP need not necessarily be discarded.

In order to not tie up network resources for an inordinate amount of time, Hosts must be prepared to accept messages from the network promptly. In particular, any given message will be discarded if it resides on a queue to the Host for more than thirty seconds. (With the current IMP system, this requires that the Host must read its interface at the rate of about 1,500 bits/second, averaged across about twenty seconds.) If the Host does not meet this constraint, the IMP will:

1. Declare the Host to be "tardy down".

2. Discard all messages pending on the queues to the Host.

3. Momentarily drop its ready line (thus setting the error flip-flop). This is done because a component failure in the interface may have caused the handshaking procedure (see Section 4.2) to get out of step, which would have the same effect as the Host merely being tardy. "Flapping" the ready line insures that the interfaces are synchronized.

4. Place some NOP's and a type 10 message on the queue to the Host.

The Host will be declared up the next time that it sends a message to the IMP or accepts a message from the IMP. The Host must send at least three NOP messages to the IMP if it is aware that it has been declared tardy, since the error flip-flop will cause the first Host-to-IMP message to be discarded. (Alternatively, the Host could bring down its own ready line; the IMP would then proceed as though the Host were in a hard down, rather than continuing to treat the Host as though it were in a tardy down.)

If the Host has advance warning that it will be going down, it may use the Host Going Down message (see Section 3.3) to inform the IMP of its status (i.e., the reason for and duration of the down). Transmission of this message from the Host to the IMP will not cause the IMP to declare the Host down; the IMP will store the status information for use during the next Host down. When the Host comes up again, the status information stored in the IMP will be discarded.

The set of events described above is summarized in Table 3-1. Suggestions for Host use of the Ready indicators are contained in Appendix B.

EVENT: / STATE	READY LINE UP	READY LINE DOWN	SEND OR RECEIVE HOST MESSAGE	RECEIVE *HOST GOING DOWN*	HOST TARDY
UP:		HARD DOWN		SET STATUS	TARDY DOWN
HARD DOWN:	UP, CLEAR STATUS		UP, CLEAR STATUS*	UP, SET STATUS*	
TARDY DOWN:		HARD DOWN	UP, CLEAR STATUS	UP, SET STATUS	

*This can't, of course, happen without the Ready line being up, but the IMP might detect the input or output before detecting the change in Ready line status.

Table 3-1 Transitions Between Host States

3.3 Host-to-IMP Leader Format

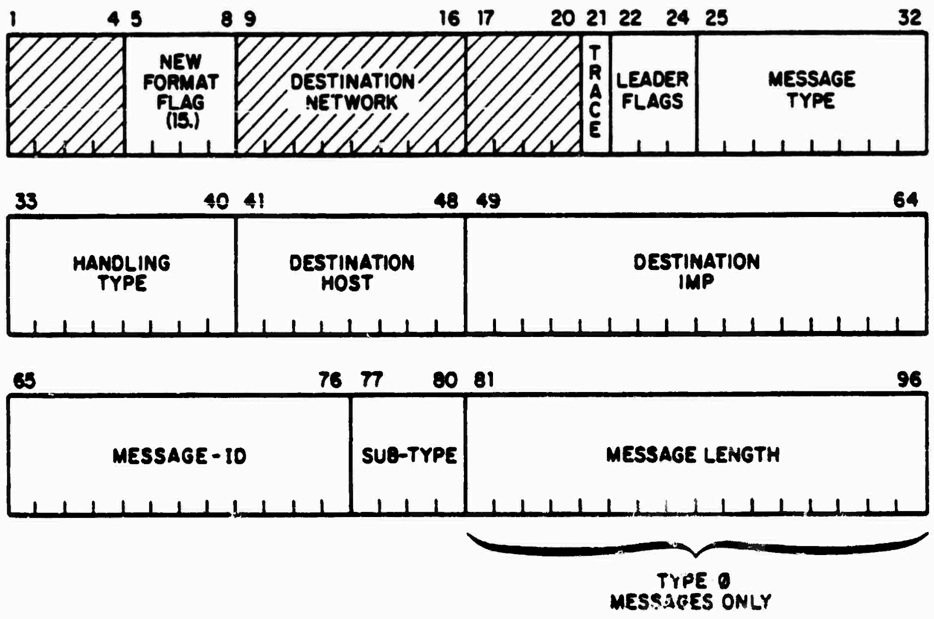


Figure 3-1 Host-to-IMP Leader Format

Bits 1 - 4 Unassigned -

Must be zero.

Bits 5 - 8 New Format Flag -

These bits are always set to the value 15. This permits the IMP to distinguish between new-style and old-style (Appendix A) leaders.

Bits 9 - 16 Destination Network -

For future use, these bits must always be zero.

Bits 17 - 20 Unassigned -

Must be zero.

Bit 21 Trace -

If equal to one, the message is designated for tracing as it proceeds through the network so that reports of this message's transit through the network may be sent to a trace destination (see Section 5.5).

Bits 22 - 24 Leader Flags -

Bits 23 and 24 are currently unassigned but are reserved for future network use and must be zero. Bit 22 is available as a destination Host flag, its meaning, if any, being assigned by that Host. The only Host with a preassigned meaning is the IMP Teletype Fake Host. If the bit is one, the message will be printed on the Teletype as a sequence of octal numbers, each representing one 16-bit IMP word. If equal to zero, then the message will be printed as a sequence of ASCII characters.*

*The IMP's internal ASCII character set is listed in Appendix E.

Bits 25 - 32 Message Type -

0. Regular Message - All Host-to-Host communication occurs via regular messages. Sub-types (bits 77-80):

   0. Standard, Non-Refusable. Interface blocking will occur if any resource needed to send the message is not immediately available.

   1. Refusable* Used to minimize the number of times the interface may be blocked. If any resource needed to send the message is not available, the message is discarded, and the Host is notified via a type 11, 12, or 13 Host-to-IMP control message. In the case of a type 12 (Refused, will notify) response, the IMP is committed to also sending a type 14 (Ready) when the resource does become available.

   2. Get Ready* (see Section 3.7). Similar to Refusable (above), except only the leader, rather than the full message, is sent in to the IMP. If all necessary resources are immediately available, the Host is notified via a Type 14 message.

   3. Uncontrolled - (see Section 3.6). The IMP will perform no message-control functions for this type of message.

   4 - 15. Unassigned.

*The non-blocking Host interface (see Section 3.7) is not yet implemented.

1. Error Without Message Identification - The Host program detected an error in a previous IMP-to-Host message and had to assume that the leader was garbled.

   Sub-types:

   0. Host's error flip-flop was set during transmission of the message.
   1. Host received a message less than 80 bits.
   2. Host received a message of an unassigned type (3, 15-255).

   3 - 15. Unassigned.

2. Host Going Down - It is assumed that as the time for the Host to (voluntarily) go down approaches, the Host itself will send warning messages to its network users. Just before going down, the Host should send the Host-Going-Down message to its IMP. The Host should then (if it can) continue to accept messages from the IMP for a period of 5 or 10 seconds, to allow messages already in the network to reach it. The IMP will store the Host-Going-Down message and return it to any source Host along with Destination (Host) Dead messages. The IMP will try to preserve this message over IMP reloads where appropriate. The NCC will be able to manually update the stored copy of this message in response to a phone call from the Host site in the event the Host is

going to be down longer than it said or if it did not have time to give warning before going down.

Bits 65-76 (the message-id field) of the Host-Going-Down message give the time of the Host's coming back up, bit-coded as follows:

Bits 65-67:	the day of the week the Host is coming back up. Monday is day 0 and Sunday is day 6.
Bits 68-72:	the hour of the day, from hour 0 to hour 23, that the Host is coming back up.
Bits 73-76:	the five minute interval, from 0 to 11, in the hour that the Host is coming back up.

All three of the above are to be specified in Universal Time (i.e., G.M.T.). The Host may indicate that it will be coming back up more than a week away by setting bits 65-76 all to ones. Setting all bits 65-75 to one and bit 76 to zero means it is unknown when the Host is coming back up.

Bits 77-80 (the sub-type field) of the Host-Going-Down message should be used by the Host to specify the reason it is going down. These bits are coded (in octal) as follows:

Value	Meaning
0-4	Reserved for IMP use
5	Scheduled P.M.
6	Scheduled Hardware Work
7	Scheduled Software Work
10	Emergency Restart
11	Power Outage
12	Software Breakpoint
13	Hardware Failure
14	Not scheduled up
15	Unspecified Reason
16-17	Currently Unused

3. Unassigned.

4. NOP - The IMP will discard this message, which is intended for use during initialization of IMP/Host communication. Bits 77-80 (the sub-type field) contain the number of 16-bit words of padding (9 max.) that the Host wishes to send and receive on type 0 messages. This padding occurs immediately after the leader (starting at bit 97) and is provided as a convenience for Hosts for which the combined Host/IMP (IMP/Host) and Host/Host leaders would otherwise not be an integral number of memory words. A simple rule for the Host to follow is to send three NOP messages whenever the Host or the IMP has been down either voluntarily or involuntarily.

5. Unassigned.

6. Unassigned.

7. Unassigned.

8. Error with Message Identification - The Host detected an error in a previous IMP-to-Host message after the leader was correctly received; e.g., the message was too long, or the IMP Error flip-flop was set after transmission of the first packet of a multiple packet message but before the end of the message. A message of this type will have a leader whose assigned bits are identical to the assigned bits in the leader of the message in error except that the message type bits will be changed to have value 8.

9-255. Unassigned.

Bits 33 - 40 Handling Type -

This field is bit-coded to indicate the transmission characteristics of the connection desired by the Host.

Bit 33: Priority - Most messages should have this bit set to zero; messages with this bit set to one will be treated as priority messages (see Section 3.1).

Bits 34-37: Currently unassigned, must be zero.

Bits 38-40: Maximum Message Size*

The maximum size (in packets) of any message the Host expects to send on the connection (#packets = (#bits in message - 96)/1008). This number is

*Until this is implemented by the IMP, the default value of 0 should be used by the Host.

expressed as (maximum # of packets - 1) and ranges from 1 (2 packets max) to 7 (8 packets max). A value of zero indicates the default maximum which is 8 packets. It is to the advantage of the Host to specify this quantity as accurately as possible, since it enables the destination IMP to make the most efficient allocation of reassembly space. On the other hand, messages that must remain in strict sequence must all have the same handling type. Multiple connections between two Hosts, each with a different maximum message size, should be used only when there are large differences in the maxima and strict sequencing is not required. A message whose length exceeds the specified maximum will be discarded and type 9, subtype 1 will be returned to the Host.

Bits 41 - 48 Destination Host -

Identify the particular Host at an IMP site. Host numbers 252-255 are reserved for use by the IMP's "fake" Hosts (see Section 5).

Bits 49 - 64 Destination IMP -

Identify the IMP site

Bits 65 - 76 Message-id -

Host-specified identification supplied in all type 0 and 8 messages. Also used in type 2 (Host-Going-Down) message.

Bits 77 - 80 Sub-type -

Used by message types 0, 2, 4, and 8.

Bits 81 - 96 Message Length -

This field is used for type 0 messages only and specifies the length (in bits) of the message, exclusive of leader, leader padding and hardware padding. The only use that the IMP makes of this field is the Get Ready (Sub-type 2) message where it is used to determine if the message is single or multi-packet. If a zero length is given in a Get Ready message, a multi-packet length is assumed.

The following table shows which non-constant fields are used by each valid message type.

	Message Type				
Fields	0	1	2	4	8
Trace	x				
Leader Flags	x				
Message Type	x	x	x	x	x
Handling Type	x			x	
Destination Host	x			x	
Destination IMP	x			x	
Message-id	x		x	x	
Sub-type	x		x	x	x
Message Length	x				

3.4 IMP-to-Host Leader Format

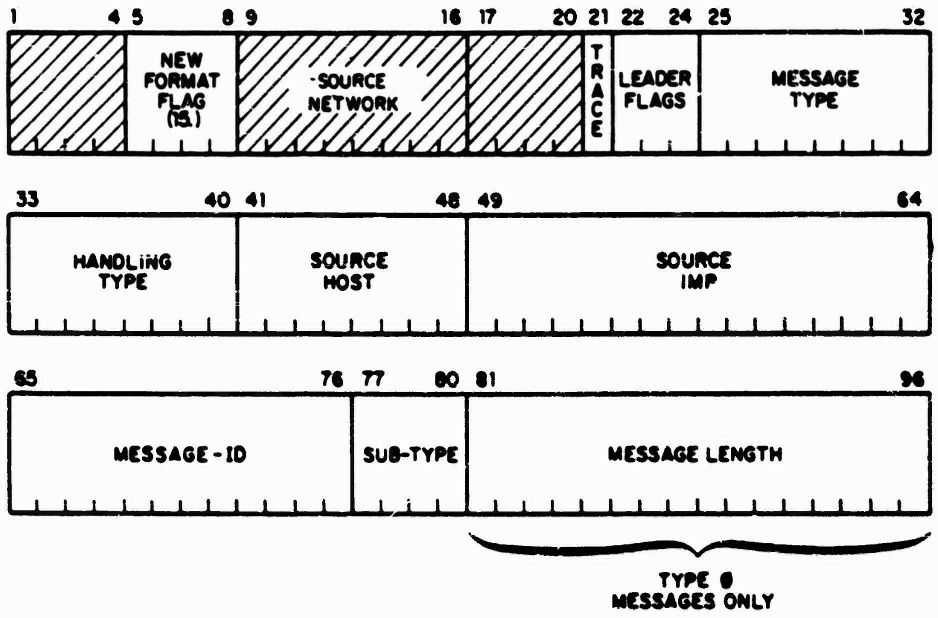


Figure 3-2 IMP-to-Host Leader Format

Bits 1 - 4 Unassigned -

Set to zero.

Bits 5 - 8 New Format Flag -

Set to 15.

Bits 9 - 16 Source Network -

Currently set to zero.

Bits 17 - 20 Unassigned -

Set to zero.

Bit 21 Trace -

If equal to one, source designated that message be traced (see Section 5.5). Used in type 0 messages only.

Bits 22 - 24 Leader Flags -

Bits 23 and 24 are currently unassigned and are set to zero. Bit 22 may be assigned a meaning by the destination Host, in which case it is used by the source Host to signal some special meaning, e.g. octal printing for the Teletype Fake Host. Used in type 0 messages only.

Bits 25 - 32 Message type -

0. Regular Message - All Host-to-Host communication occurs via regular messages. The subtype field is the same as sent in the Host-to-IMP message; in particular a sub-type of 3 indicates an uncontrolled message (see Section 3.6).

1. Error in Leader - the IMP detected an error in a previous Host-to-IMP message and had to assume that the leader was garbled.

   Sub-types:

   0. IMP's Error flip-flop set during the first 96 bits of a message (see Section 3.2).
   1. IMP received a message of less than 80 bits (32 if old format).
   2. IMP received a message of an illegal Type.

3. IMP received a message of the opposite leader style than it was expecting.

2. IMP Going Down - The IMP will transmit this message to its Host before it voluntarily goes down. The Host should forward the information in the message to its users from the network (and to its own users of the network).

Bits 65-80 of the message are coded as follows:

Bits 65-66: Why;

0. "last warning" or "panic restart": The IMP is going down in 30 seconds.
1. Scheduled hardware PM
2. Scheduled software reload
3. Emergency restart

Bits 67-70: How Soon; in 5 minute increments (zero implies immediately)

Bits 71-80: For How Long; in 5 minute increments (zero implies immediately)

3. Unused.

4. NOP - The Host should discard this message. It is used during initialization of IMP/Host communication. The Host and IMP fields will contain the local Host and IMP identification numbers, and the sub-type field will be zero. All other fields are unused.

5. RFNM - "Ready for Next Message". The named regular message was successfully delivered to the destination IMP, and the destination Host accepted it. In addition, if the named message is longer than one packet (about 1103 bits including leader) space may be reserved at the destination IMP for another transmission, but the space reservation will remain valid for only a short time (see Section 3.1). The subtype field will be 0 if the original message was non-refusable, and 1 if it was refusable.

6. Dead Host Status - Bits 65-76 (the message-id field) have the same meanings as bits 65-76 in the Host-to-IMP type 2 (Host-Going-Down) message described in Section 3.3. Bits 77-80 (the sub-type field) have the following meanings:

Value	Meaning
0	Currently Unused
1	The destination Host is not communicating with the network -- it took its ready-line down without saying why.
2	The destination Host is not communicating with the network -- the Host was tardy in taking traffic from the network without saying why.

3	The destination Host does not exist to the knowledge of the NCC.
4	The IMP software is preventing communication with this Host; this usually indicates IMP software re-initialization at the destination.
5	The destination Host is down for scheduled P.M.
6	The destination Host is down for scheduled hardware work.
7	The destination Host is down for scheduled software work.
8	The destination Host is down for emergency restart.
9	The destination Host is down because of power outage.
10	The destination Host is stopped at a software breakpoint.
11	The destination Host is down because of a hardware failure.
12	The destination Host is not scheduled to be up.
13-14	Currently Unused.
15	The destination Host is in the process of coming up.

When the value of the sub-type field is 1, 2, 3, 4, or 15, the message-id field will have the "unknown" indication.

Bit 33 in this message will always be set to zero and Hosts receiving this message should discard (without reporting an error) type 6 messages with bit 33 set to 1. This will allow the later addition of similar status information on dead destination IMPs.

The Dead Host status message will be returned to the source Host shortly (immediately, if possible) after each Destination Host Dead (type 7, subtype 1) message. The destination Host Dead message applies to a specific named message, although the information contained in the Destination Host Dead message should probably be reported to all users connected to the dead Host. The Dead Host Status message does not apply to a specific named message and all users connected to the dead Host should be notified of the information contained in the Dead Host Status message.

7. <u>Destination Host or IMP Dead (or unknown)</u> - This message is sent in response to a message for a destination which the IMP cannot reach. The message to the "dead" destination is discarded.

Sub-types:

0. The destination IMP cannot be reached.
1. The destination Host is not up.
2. Communication with the destination Host is not possible because it does not have the expanded (new) leader capability (see Appendix A).
3. Communication with the destination Host is administratively prohibited.

4-15. Currently unused.

8. <u>Error in Data</u> - The IMP's Error flip-flop was set after transmission of the leader of a message but before the end of the message.

9. <u>Incomplete Transmission</u> - The transmission of the named message was incomplete for some reason. An incomplete transmission message is similar to a RFNM, but is a failure indication rather than a success indication.

Sub-types:

0. Destination Host did not accept the message quickly enough.
1. Message was too long (in excess of maximum number of packets specified for connection).
2. The Host took more than 15 sec. to transmit the message to the IMP. This time is measured from the last bit of the leader through the last bit of the message.

3. Message lost in the network due to IMP or circuit failures.
4. The IMP could not accept the entire message within 15 sec. because of unavailable resources (see Section 3.1).
5. Source IMP I/O failure during receipt of this message.

6-15. Currently unused.

10. Interface Reset - The IMP's ready line has been dropped and pending output to the Host has been discarded (see Section 3.2). This probably indicates that the Host did not accept data from the IMP fast enough. Since dropping the ready line also sets the IMP's error flip-flop, the next message from the Host will be discarded and answered with a type 1 (sub-type 0) message. The sub-type field is unused.

11. Refused, Try Again* - A type 0, suotype 1 or 2 message was received from the Host but a certain "non-markable" resource needed for sending the message was not available. The message was discarded, and the Host should try to send it again when best able to do so.

Sub-type:

*The non-blocking Host interface (see Section 3.7) is not yet implemented.

0. IMP buffer was not available.

1. Transmit block for connection was not available.

2-15. Currently unused.

12. Refused, Will Notify* - A type 0, subtype 1 or 2 message was received from the Host but a certain "markable" resource needed for sending the message was not available. The message was discarded, and the Host will be notified via a type 14 (Ready) message when the resource becomes available.

Sub-types:

0-1. Currently unused.

2. Connection not available.

3. Reassembly space (for multi-packet message only) not available at destination.

4. Message number not available.

5. Transaction block for message not available.

6-15. Currently unused.

13. Refused, Still Trying* - A type 12 response is indicated, but a type 14 message has already been queued for some previous type 12 response. The message was discarded and no other response will be given. The subtype field is unused.

*The non-blocking Host interface (see Section 3.7) is not yet implemented.

14. Ready* The needed resource has become available for some previous type 0, subtype 1 or 2 message. The actual message is "named" by the message-id field.

15-255. Unassigned. Messages of other than type 0 are sent to the Host prior to messages of type 0.

Bits 33 - 40 Handling Type -

The value assigned by the source Host, this field is used only in message types 0, 5, 7-9, and 11-14.

Bits 41 - 48 Source Host -

See Source IMP, below.

Bits 49 - 64 Source IMP -

For type 0 messages, these fields identify the particular Host and IMP site that originated the message. For type 4 messages, these fields identify the local Host and IMP, and for message types 5-9 and 11-14, these fields identify the particular Host and IMP site to which a type 0 message was sent or will be sent. The fields are unused in all other message types.

*The non-blocking Host interface (see Section 3.7) is not yet implemented.

*The non-blocking Host interface (see Section 3.7) is not yet implemented.

Bits 65 - 76 Message-id -

For message types 0, 5, 7-9, and 11-14, this is the value assigned by the source Host to "name" the message. The field is also used by message types 2 and 6, and unused by all other message types.

Bits 77 - 80 Sub-type -

This field is used by message types 0-2, 4-7, 9, and 11-12.

Bits 81 - 96 Message Length -

This field is contained in type 0 messages only, and is the actual length in bits of the message (exclusive of leader, leader padding, and hardware padding) as computed by the destination IMP using the end of message padding conventions. It should be noted that the IMP will not verify the length of the message if it is specified by the Host.

The following table shows which non-constant fields are used by each valid message type.

	Message Type													
Fields	0	1	2	4	5	6	7	8	9	10	11	12	13	14
Trace	x													
Leader Flags	x													
Message Type	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Handling Type	x				x		x	x	x		x	x	x	x
Source Host	x			x	x	x	x	x	x		x	x	x	x
Message-id	x		x		x	x	x	x	x		x	x	x	x
Sub-type	x	x	x	x	x	x	x	x	x		x	x		
Message Length	x													

3.5 Word Length Mismatch and Message Boundaries

There are two related aspects of word length mismatch: first, the obvious need for message formatting in order for Host computers having different word lengths to communicate; and, second, the need for locating the end of a message, since mismatched word lengths may lead to messages that end in the middle of words. The IMP design guarantees that between Hosts of identical word length, the natural word boundaries are preserved. Generally, however, reformatting is left to the Hosts. The problem of recognizing the end of a message at the receiving Host is solved in the following manner. As a message passes from the transmitting Host to its IMP, the standard Host/IMP interface appends a one to the bit string when it receives the end-of-message signal. This bit may fall in any position of an IMP word. The hardware then fills any remaining bits of this IMP word with trailing zeros. This process is called IMP padding. The transmitting Host may also specify the message length (in bits), which need not be the same as the physical length of the message.

As the message is serially shifted to the receiving Host, the last bit from the IMP will generally fall somewhere in the middle of the receiving Host's word. The remaining bits in this word are to be filled in with additional trailing zeroes from the Host's special interface hardware. (Note that a one is purposely omitted here.) Thus, the message appears in the receiving Host with a one immediately following the last data bit in the message and a string of zero or more trailing zeroes, that terminates at a Host word boundary, following the one. The last Host word in the received bit stream does not necessarily contain the last data bit in the message; it may contain nothing but padding.

The maximum message that is shipped across the interface from the IMP to the destination Host contains 8160 bits (i.e., it includes the source IMP's padding). The destination Host's special interface unit will generally add padding of its own to round out the total number of bits going into the Host's memory to a multiple of the destination Host's word length. The destination Host should, therefore, be prepared to accept messages of at least 8160 bits. Not counting the destination Host's padding, messages of greater than 8160 bits in length should be discarded by the receiving Host.

It should be noted that Hosts may specify <u>leader padding</u> (see Section 3.3, NOP message). This padding is some integral number of 16-bit words which are transmitted and received immediately following the 96-bit leader of type 0 messages. This facility is designed to assist the Host in aligning some portion of the transmitted or received data with its own word boundaries. In particular, the Host may wish to make the sum of leader, leader padding, and other elements of Host-to-Host Leader equal to an integral number of Host words. This leader padding is <u>not</u> counted in the message length and exists only across the Host/IMP interface (i.e., not in the network).

3.6 Uncontrolled Packets

For certain limited experiments which are being carried on using the network, it may be desirable for specified Hosts to be able to communicate without using the normal ordering and error-control mechanisms in the IMP. Communication of this type is possible using the Host-to-IMP and IMP-to-Host message type 0, sub-type 3. The rules governing IMP handling of these messages are:

1. Messages of type 0, subtype 3 are limited to the Host-to-IMP leader (96 bits) and not more than 991 additional data bits. Messages which exceed this length will be discarded without error notification.

2. At the destination IMP, these messages are put on the output queue for the destination Host in the order in which they are received; the messages are likely to be delivered in a different order from the order in which they were sent. Duplicate copies of some messages may be delivered.

3. There is no source-to-destination control of these messages. Lost messages will not be retransmitted. No RFNM, Incomplete Transmission, Destination Dead, etc., will be returned to the source.

4. The same bit-level error control applied to Regular messages will be applied to these messages passing between IMPs; i.e., type 0 subtype 3 messages are delivered with a very low probability of bit error.

5. If at any time there are insufficient resources in the network to handle one of these messages, it will be immediately discarded.

6. Use of these messages between two Hosts will not affect use of regular messages between these Hosts. Regular messages and subtype 3 messages may be intermixed over the Host/IMP interface.

7. Uncontrolled use of these messages will degrade the performance of the network for <u>all</u> users. Therefore, ability to use these messages will be regulated by the Network Control Center and will require prior arrangement for each experiment.

3.7 Non-Blocking Host Interface*

As mentioned in Section 3.1, it is sometimes necessary for the source IMP to block the transmission of a message from the source Host. When this blocking occurs, all messages from that Host are held back, even though some of them might well be transmitted unimpeded if allowed into the IMP. Such might be the case, for example, if Host A is sending to Hosts B, C, and D, and the connection to Host B has eight messages in transit, the first (oldest) of which has become lost in the net. If a ninth message is sent to B, the interface will be blocked for the duration of the "incomplete" timeout (30-45 seconds), waiting for a message slot to become available on that connection. During this time, however, it would have been possible for A to send messages to C and D, had the interface not been blocked.

The non-blocking Host interface is a software mechanism which provides the source Host with the capability of keeping the interface unblocked for the vast majority of situations under which it might otherwise have become blocked. There will still be a few circumstances, associated with bandwidth and storage limitations of the source IMP, under which the interface may be blocked regardless of the mechanism used by the Host.

*This section is a preliminary specification and is still subject to modification. The extensions required to the Host/IMP protocol have not yet been implemented.

The non-blocking mechanism works by allowing the Host to flag some or all of its type 0 messages as "refusable", thus allowing the IMP to discard them if they would otherwise block the interface. In such a case, not only is the Host notified that the message was discarded, but it is also given guidance as to when the message should be retransmitted. In most cases, the particular resource that was missing is "markable", and the Host can be notified when the resource becomes available. In some cases, the resource is not "markable", and the Host must simply retransmit in accordance with its own requirements. The specific protocol for this mechanism is now described.

Host-to-IMP type 0 messages have four subtypes: Non-refusable, Refusable, Get Ready, and Uncontrolled. The uncontrolled subtype, described in Section 3.6, is never refused, and because it does not require most of the resources of "controlled" messages, is seldom blocked. The Non-refusable subtype is the standard mode of operation, which can cause interface blocking under the various circumstances described in Section 3.1. The Refusable subtype is treated identically to the Non-refusable subtype if blocking is not necessary. Under most circumstances where blocking would have been necessary, however, this message subtype is discarded, and one of three types of IMP-to-Host messages sent back to the Host. A Refused, Try Again (type 11) message indicates a "non-markable" resource was required, and the Host should merely retransmit at its convenience. A Refused, Will Notify (type 12) message indicates a "markable" resource was required. The Host should wait for a fourth IMP-to-Host message type, Ready (type 14), before retransmitting. The IMP will send the Ready when the resource becomes available. A Refused, Still Trying (type 13) message indicates that the IMP has already given a Refused, Will Notify on that connection, but has not yet sent the Ready (it will only queue one such response at a time for any connection). There is no additional response after the Refused, Still Trying, and the Host should queue the message to be retransmitted after the one for which the Ready is expected.

The Get Ready subtype of the type 0 Host-to-IMP message is not a real message in the sense that it contains only the leader of an intended (future) message. It is provided so that the Host can determine whether or not a message could get through without blocking, without actually sending the data in the message through the interface. The possible responses to this subtype are identical to those of the Refusable subtype, except that in the normal case, when the Refusable message would have been transmitted to the destination without any interface blocking followed eventually by a RFNM, the IMP's response to the Get Ready is to send a Ready back to the Host.

Finally, it should be noted that a Ready does not guarantee that a retransmission will not be blocked, since no resources are actually reserved for some particular message-id, and in fact many are shared by all connections. The best strategy for the Host willing to use the non-blocking feature is to make all messages Refusable, even when responding to a Ready.

4. HARDWARE REQUIREMENTS AND DESCRIPTION

A local Host is connected to the IMP through a Host cable (provided with the IMP), which joins a standard Host/IMP interface unit in the IMP to a special Host/IMP interface unit in the Host. A distant Host is connected to an augmented standard Host/IMP interface through a cable provided by the Host. The structure of the standard Host/IMP interface, the IMP/Host handshaking procedure, the end-of-message indication, the Master Ready lines, and the signals on the Host cable are all described in detail below. A very distant Host is connected via communications circuits to a modem interface unit as described in Appendix F.

The special interface should be designed by the Host personnel to operate in conjunction with the standard Host/IMP interface or the augmented interface as the case may be. We have not, however, attempted to specify the special Host/IMP interface in any detail. We recommend that the special interface be modeled after the standard interface, and, in the remainder of this section, we assume that it will be. It should be noted that the special interface must be operated in a full duplex mode.* A simplified schematic drawing of a special Host/IMP interface is

*Those few Hosts which originally implemented half duplex interfaces have had inordinate difficulties of various kinds. See, for example, Section 3.1.

included in Appendix B to assist Host personnel in the design of the special interface. The distant Host modification to the standard interface affects only the cable and the method of cable driving; it does not change the basic operation of the interface.

4.1 Structure of the Standard Host/IMP Interface

The standard Host/IMP interface is a full duplex bit-serial unit that is logically divided into a Host-to-IMP section and an IMP-to-Host section. Each section contains a 16-bit shift register (and control logic), one of which is for shifting bits to the Host and the other for receiving bits from the Host. A simplified picture of the Host/IMP interface is shown in Figure 4-1.

The technique of transferring information between the Host and the IMP is nearly identical in each direction; we will, therefore, refer to the sender and the receiver without specifying the Host or IMP explicitly.*

In general, words are taken one by one from the sender's memory; and transferred bit serially across the interface to the receiver, where they are reassembled into words of the appropriate (i.e., receiver's) length and stored into the

*Although this report specifies a highly symmetrical interface, there are some aspects peculiar to the IMP side and some to the host side. For a discussion of these asymmetries (padding, diskewing, etc.) see "JANUS Interface Specification" NIC #43649.

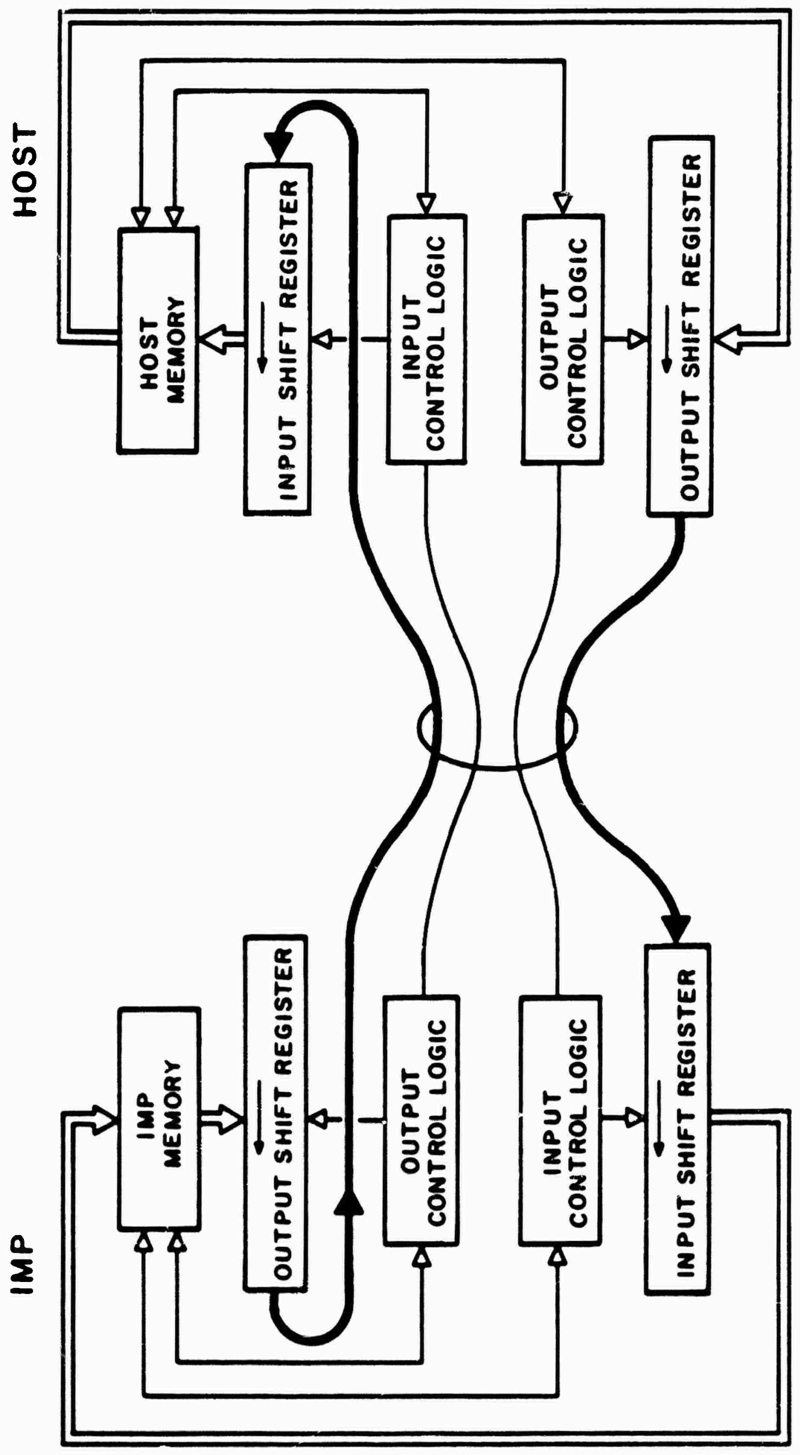


Figure 4-1 Simplified Illustration of Host/IMP Interface

receiver's memory. The transmission thus consists of a bit stream containing no special indications of word boundaries but delayed occasionally while the sender fetches, or the receiver stores, a word. The high-order bit of each word is transmitted first.

Bit transfer is asynchronous, the transmission of each bit being controlled by a Ready-For-Next-Bit, There's-Your-Bit handshaking procedure. Each bit is transferred only when both sender and receiver indicate preparedness. This permits either the sender or the receiver to hold up the transmission between any two bits in order to take as much time as necessary to get a new word from memory, to tuck an assembled word into memory, or to activate an interrupt routine that sets up new input or output buffers. Neither the sender nor the receiver should expect transmission to take place at a pre-determined bit rate and each must be able to accept arbitrary delays introduced by the other at any point in the bit stream.

The design of an asynchronous interface was selected for two reasons: first, because of the inherently asynchronous nature of the process by which words of one length are fetched from one machine and reformed into words of another length and stored in another machine; and, secondly, because such a design allows a variety of special Host/IMP interfaces to be designed independent of stringent timing specifications that may be difficult or impossible for certain Hosts to meet.

4.2 IMP/Host Handshaking

Figure 4-2 shows a much simplified version of the control logic for the bit-by-bit handshaking procedure. When PG #1 (Pulse Generator) fires, it turns off the Bit Available flip-flop and a new data bit is shifted into position by the sender. The Bit Available flip-flop is then turned back on, and, if (or when) the receiver is ready to receive a bit, a There's-Your-Bit signal is sent to him. This triggers PG #2* which shifts in the new bit and shuts off the Ready-For-Next-Bit flip-flop. When this indicator goes off, the sender knows that the bit has been taken by the receiver. PG #1 then fires and shuts off the Bit Available flip-flop in preparation for getting the next bit ready for transmission. After the receiver has taken in the bit and is ready to accept a new one, it turns the Ready-For-Next-Bit flip-flop back on. The cycle then repeats.

Each time the sender is notified that a bit has been accepted (by the off transition of Ready-For-Next-Bit), a word length counter is checked to see whether a new word must be fetched from memory. Similarly when a bit is accepted at the receiver, it may be necessary to tuck an assembled word into the memory before registering readiness to receive anther bit. In addition to these obvious requirements, the simplified picture

*The on (↑) transition of There's-Your-Bit triggers PG #2. The off (↓) transition of Ready-For-Next-Bit triggers PG #1.

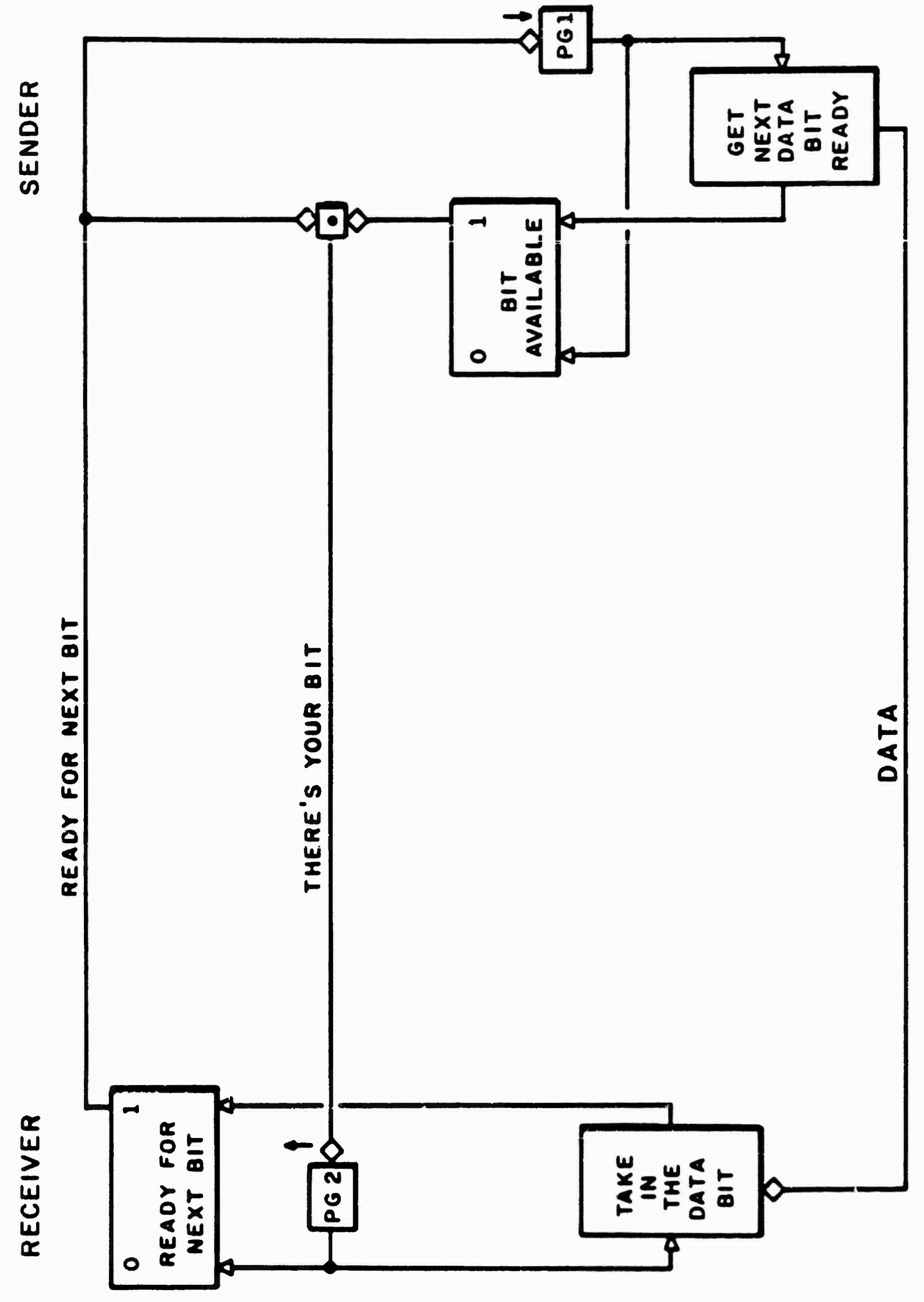


Figure 4-2 Simplified Control Logic for Host/IMP Handshaking

contains critical race problems*, which have been carefully resolved in the IMP's interface and must be similarly resolved in the Host's special interface.

The receiver may choose either of two methods of handshaking, a two-way or a four-way handshake. In the four-way handshake, the receiver awaits the dropping of There's-Your-Bit before raising Ready-For-Next-Bit. A full cycle of the four-way handshake works as follows: The sender readies the next data bit and the There's-Your-Bit signal is sent to the receiver (1st cable transit). The receiver takes in the bit and notifies the sender by dropping Ready-For-Next-Bit (2nd cable transit). The sender responds by dropping the There's-Your-Bit signal (3rd cable transit) and after the receiver has noted this, the Ready-For-Next-Bit signal can be turned back on (4th cable transit), registering preparedness for a new bit.

The two-way handshake works as follows: The sender readies the next data bit and the There's-Your-Bit signal is sent to the receiver (1st cable transit). The receiver takes in the bit and notifies the sender by dropping Ready-For-Next-Bit (2nd cable transit). Instead of waiting for this signal to propagate to the sender and the resultant dropping of There's-Your-Bit to return, the receiver holds Ready-For-Next-Bit off for a brief period and then turns it back on.

*For example, the race in shutting off the Ready-For-Next-Bit flip-flop.

This method has two dangers that must be considered, both arising from the situation where Ready-For-Next-Bit is off for too short a time.

1. If Ready-For-Next-Bit is off for too short a period, the sender may never note that it went off and he will continue to wait for the bit to be taken. The IMP itself requires that the signal be off at the IMP end of the cable for at least 50 nanoseconds for local Hosts and for at least 1 usec for distant Hosts.*

2. If the receiver turns Ready-For-Next-Bit back on before the There's-Your-Bit signal has been observed to go off at the receiver's end of the cable, then the receiver may mistakenly believe the new bit is ready to be taken in. This problem is avoided if the receiver maintains a There's-Your-Next-Bit flip-flop which is turned off when Ready-For-Next-Bit is turned off and is turned on only by the leading edge (on transition) of the There's-Your-Bit signal from the sender.

*The 316 and 516 IMPs, in fact, always use the two-way procedure. They do not wait for the There's-Your-Host-Bit signal to go off but instead guarantee to hold the Ready-For-Next-Host-Bit signal off for at least 1 usec. The Pluribus IMP uses the four-way handshake.

For local Hosts, where the cable delays are not significant, either handshake procedure may be used. For distant Hosts, where cable delays may be significant, the two-way handshake procedure is recommended, in order to avoid placing an unnecessary restriction on the maximum bit rate.

The IMP introduces some deliberate delays into this control loop, both as a sender and as a receiver. Specifically, as a sender, the IMP introduces approximately 10 usec of delay* between the time that the Host indicates that it has taken one bit and the time that the next bit is made available. As a receiver, the IMP shifts in the data bit and turns off the Ready-For-Next-Bit signal shortly after the There's-Your-Bit signal comes on. However, Ready-For-Next-Bit will not be turned on again until about 10 usec* after There's-Your-Bit comes on. By introducing these deliberate delays, the IMP slows down the rate of information flow on the Host channels, thereby controlling the maximum amount of IMP memory bandwidth that the channels can consume. This control is essential to avoid usurping bandwidth required for the store-and-forward functioning of the IMP.

*These are minimum times assuming no IMP memory reference is required. Where a memory fetch or store is required, the times will be increased by at least 4 usec.

Because of the loop nature of the handshake procedure, the Host can also introduce delays. However, knowing that the IMP will limit the data rate, the Host should, in general, not introduce further deliberate delays of its own. The delays we have mentioned are adjustable and can be tuned so that the interface operates at much higher speeds. At the time of installation of a new IMP, the standard interface will be set to run the Host channels at a 10-usec-per-bit maximum rate. Once the IMP is connected to the Host, both the input and the output channels will normally be tuned to operate at a maximum rate of 100 kilobits/second, thereby lumping together the delays in the IMP interface and the Host special interface.*

4.3 End-of-Message Indication

A Host indicates the end of its message to the IMP by presenting a Last-Host-Bit signal to the IMP during transmission of the last data bit. This signal will generally occur somewhere in the middle of an IMP word, i.e., with the input shift register in the standard interface only partially loaded. Additional

*Since the IMP as a receiver holds the Ready-For-Next-Bit signal off for 10 usec, there does not appear to be any real need for the Host to have a Bit-Available flip-flop go on and off on a per-bit basis. The There's-Your-Bit line will go off when Ready-For-Next-Bit goes off. However, the 10 usec delay is subject to shrinkage; therefore, the Host should not rely on this delay to provide time for the next bit to arrive -- even if getting the bit amounts only to moving a shift register over one place.

padding bits will then be shifted into the register, namely a single one followed by enough zeroes (perhaps none) to fill up the register. These additional bits are appended at the end of a Host message by the hardware in the input section of the standard interface. If the last data bit happens to just fill the shift register, an additional IMP word consisting of a single one followed by fifteen zeroes will be appended to the message. Alternatively, if the single one happens to just fill the shift register, the IMP padding will contain only this single one. At the destination, the IMP will indicate the end of the message to its Host by presenting a <u>Last-IMP-Bit</u> signal to the Host together with the last bit of the IMP padding. In general, this signal will occur somewhere in the middle of a Host word, i.e., with the input shift register in the special interface only partially loaded. The Host must shift enough additional zeroes (perhaps none) into this register to fill up the register.

4.4 Master Ready Lines

Whenever the IMP is ready, it holds closed a relay contact that connects two wires (the <u>IMP Master Ready</u> and the <u>IMP Ready Test</u> lines) in the Host cable. Figure 4-3 illustrates how the Host can employ this contact closure to ground a clamped logic

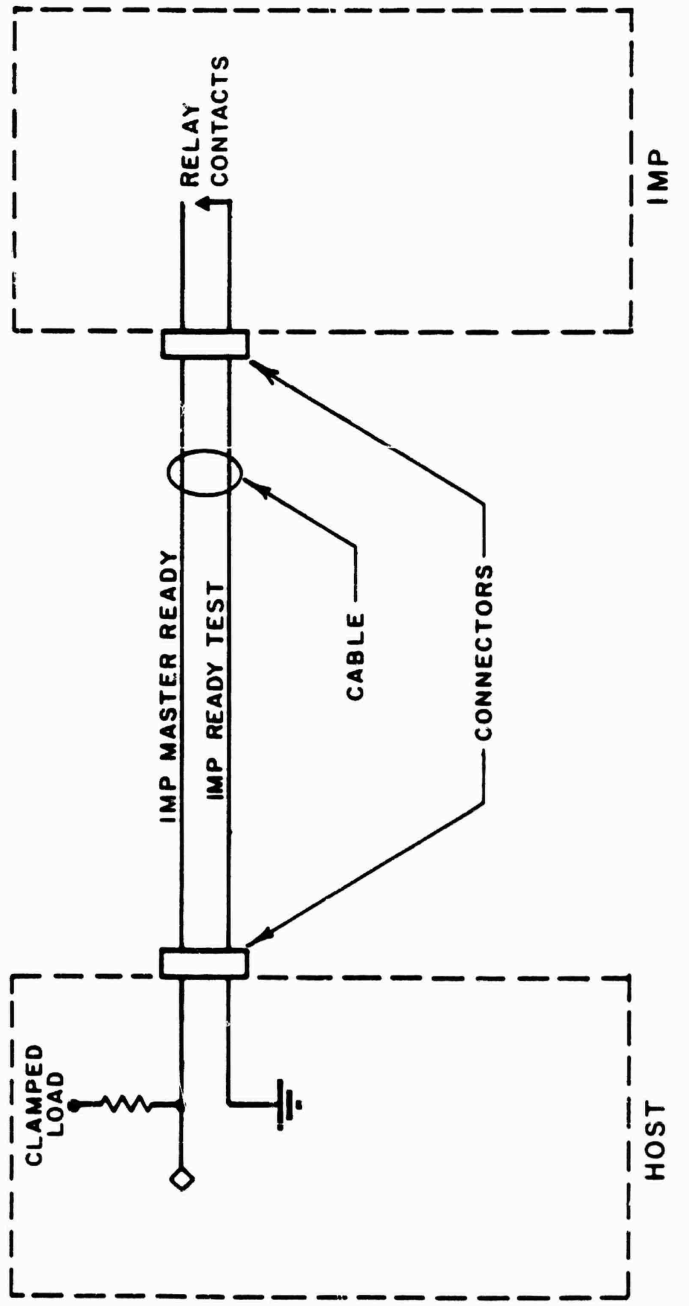


Figure 4-3 IMP Ready Test and IMP Master Ready Lines

Master Ready signal passes through a relay, it will, in general, show contact bounce. When the IMP becomes ready (i.e., closes its relay), it executes a programmed delay before the There's-Your-IMP-Bit line becomes true. This delay covers the contact bounce period and thus the Host need not worry about bounce on the gated versions of this signal. (The IMP also executes a programmed delay before beginning a new input operation. Since there may however be errors in the current transmission to the IMP, the Host should always send at least one NOP message after seeing the IMP in a not-Ready state.)

The Host should provide similar protection by not permitting the There's-Your-Host-Bit signal to become true until after its relay contacts have solidly finished closing.

4.5 Host Cable Connections

Following is a summary of the signals on the Host cable:

1. IMP Master Ready - The return for the IMP Ready Test signal through the IMP's relay contact.

2. IMP Ready Test - The test signal sent to the IMP to interrogate its ready status through the IMP's relay contacts. No more than 100 ma. should flow in this wire and the IMP Master Ready wire.

line whose polarity indicates readiness of the IMP*. Note that, if the cable is removed at either end, the IMP appears to the Host not to be ready. The relay contacts are a Normally-Open pair and thus, if the IMP's power goes off, the line indicates "not ready".

The relay closure is also controlled by the IMP program. If the IMP detects a serious program failure, it initiates an automatic recovery procedure. This same procedure is also initiated by the Network Control Center under certain conditions. Execution of the recovery procedure causes the relay to open; successful recovery will eventually cause the relay to close again.

Similarly, each Host must provide for its IMP a set of contacts, which open when Host power goes off or whenever the Host does not wish to communicate with the rest of the network for an extended period.** The IMP will use this contact, in the specific manner suggested above, to pass a signal ground around to itself for testing Host readiness.

The special Host interface should gate all incoming signals with the signal (or its inverse) on the IMP Master Ready line in order to avoid responding to meaningless transitions. Since the

*The choice of ground as the interrogation level is obviously arbitrary, and the Host may use any reasonable arrangement.

**See Section 3.2 for a more complete discussion of the alternatives available to the Host for voluntarily stopping communication with the rest of the network.

itself.*

7. Ready-For-Next-Host-Bit - This signal will be presented to the Host whenever the IMP is waiting for a transmission by the Host. Each time that the Host gives the IMP a bit (via There's-Your-Host-Bit), the Ready-For-Next-Host-Bit will go off after the bit has been taken in. It will go back on again within 10 usec unless a memory access is required (once every 16 bits). A much longer off period will result when an IMP memory buffer region fills, and an interrupt service routine must operate before the IMP is ready for another bit.

8. Last-Host-Bit - When the Host transmits the last bit of a message, the Last-Host-Bit signal should be sent to the IMP in conjunction with the There's-Your-Host-Bit signal. Specifically, the Last-Host-Bit signal must come on no later than the There's-Your-Host-Bit signal comes on, and should remain on at least until Ready-For-Next-Host-Bit goes off. The IMP will pad the message with a one followed by enough zeroes (perhaps none) to fill the current IMP word.

*At first glance this seems like duplication. However, when the next bit becomes available, the Bit Available flip-flop will be turned back on and yet the There's-Your-Bit signal should not be sent unless the Ready-For-Next-Bit signal is on. Thus, the need for the AND gate. Shutting off Bit Available is required to avoid confusing the receiver with an old bit when a new Ready-For-Next-Bit signal comes on.

3. Host Master Ready - The return for the Host-Ready-Test signal through the Host's relay contact.

4. Host Ready Test - The ground signal sent to the Host to interrogate its ready status through the Host's relay contacts. No more than 100 ma. should flow in this wire and the Host Master Ready wire.

5. Host-to-IMP Data Lines - The data from the Host should be changed for successive bits only after the IMP's Ready-For-Next-Host-Bit signal goes off indicating that the previous bit has been selected.

6. There's-Your-Host-Bit - This signal should be presented to the IMP by the Host as soon as the Host has a bit available to transmit and the IMP is indicating that it is Ready For Next Host Bit. When the Ready-For-Next-Host-Bit signal goes off, the There's-Your-Host-Bit signal should be removed. This must be done in two ways, as shown in Figure 4-2 -- first by the AND gate between the Bit Available flip-flop and the Ready-For-Next-Bit signal, and second by immediately turning off the Bit Available flip-flop

9. IMP-to-Host Data Line - The data for the Host will be changed for successive bits only after the Host's Ready-For-Next-IMP-Bit signal goes off, indicating that the previous bit has been accepted.

10. There's-Your-IMP-Bit - This signal will be presented to the Host by the IMP as soon as the IMP has a bit available to transmit and the Host presents the Ready-For-Next-IMP-Bit signal. When the Ready-For-Next-IMP-Bit goes off, the There's-Your-IMP-Bit signal will be removed. It will not be renewed until a new Ready-For-Next-IMP-Bit signal arrives.

11. Ready-For-Next-IMP-Bit - This signal should be presented to the IMP whenever the Host is ready to receive information. Each time that the IMP gives the Host a bit (via the There's-Your-IMP-Bit line), the Ready-For-Next-IMP-Bit signal should go off after the bit has been taken in. This notifies the IMP that the bit has been taken and that a new bit can be moved into position and made available. Ready-For-Next-IMP-Bit should be off for at least 50 nanoseconds (1 usec for distant Hosts) as seen at the IMP before it goes back on again. It may, of course, be off for as long as it takes the Host to ready itself to receive the next bit.

12. Last-IMP-Bit - When the IMP transmits the last bit of the source IMP's padding, the Last-IMP-Bit signal will be sent to the Host in conjunction with the There's-Your-IMP-Bit signal. Specifically, the Last-IMP-Bit signal will come on no later than the There's-Your-IMP-Bit signal. Last-IMP-Bit will stay on for some *arbitrary* short time after There's-Your-IMP-Bit goes off. The Host's interface must not interrogate this line after the Ready-For-Next-IMP-Bit signal has been turned off. The Host's special interface should round out the last memory word with zeroes, as required.

The asynchronous (i.e., sequential) nature of the interface causes stress to be laid on the *order* in which operations occur rather than on their *timing*. Minimum on or off times for the circuits in the IMP are 50 nanoseconds for a local Host and 1 usec for a distant Host. Thus, for example, the Host's Ready-For-Next-IMP-Bit line must be visibly down at the IMP for at least this length of time before it is brought back up even if the Host takes the bit more quickly than that. Similarly, the Host's Bit Available flip-flop must appear off to the IMP (via the There's-Your-Host-Bit line) for at least 50 nanoseconds for local Hosts and 1 usec for distant Hosts. The IMP delays only a very short time from the arrival of There's-Your-Host-Bit before taking the Host's data bit or checking the Last-Host-Bit line.

However, for IMP-to-Host transmission, the IMP will guarantee that the IMP data bit is on the line and the Last-IMP-Bit level is correct at least 500 nanoseconds before turning on the There's-Your-IMP-Bit signal. Thus, skews of under 500 nanoseconds in the signals for IMP-to-Host transmission at the IMP end of the cable will be removed by the IMP, but skews for Host-to-IMP transmission must be removed by the Host.

4.5.1 Connection to a Local Host

The nominal asserted level for all logic lines (Data, Ready-For-Next-Bit, There's-Your-Bit, Last Bit) is +5 volts and the unasserted level is ground (these are with respect to the IMP signal ground). The driving and receiving circuits and specifications are shown in Appendix C.

The Host cable supplied with the 516 IMP and the Pluribus IMP is 30 feet long and contains 12 RG 174/U coaxial conductors with grounded shields. Host personnel must provide an appropriate connector for the Host end of the cable. The shield of each conductor is connected to signal ground at the IMP connector. Each cable is labelled with the IMP connector pin number corresponding to the center lead of the coaxial conductor. These wires are assigned as indicated in Figure 4-4 for the 516 IMP; that is, the number in the figure corresponds to the number of the label attached to each coaxial conductor. Pluribus IMP

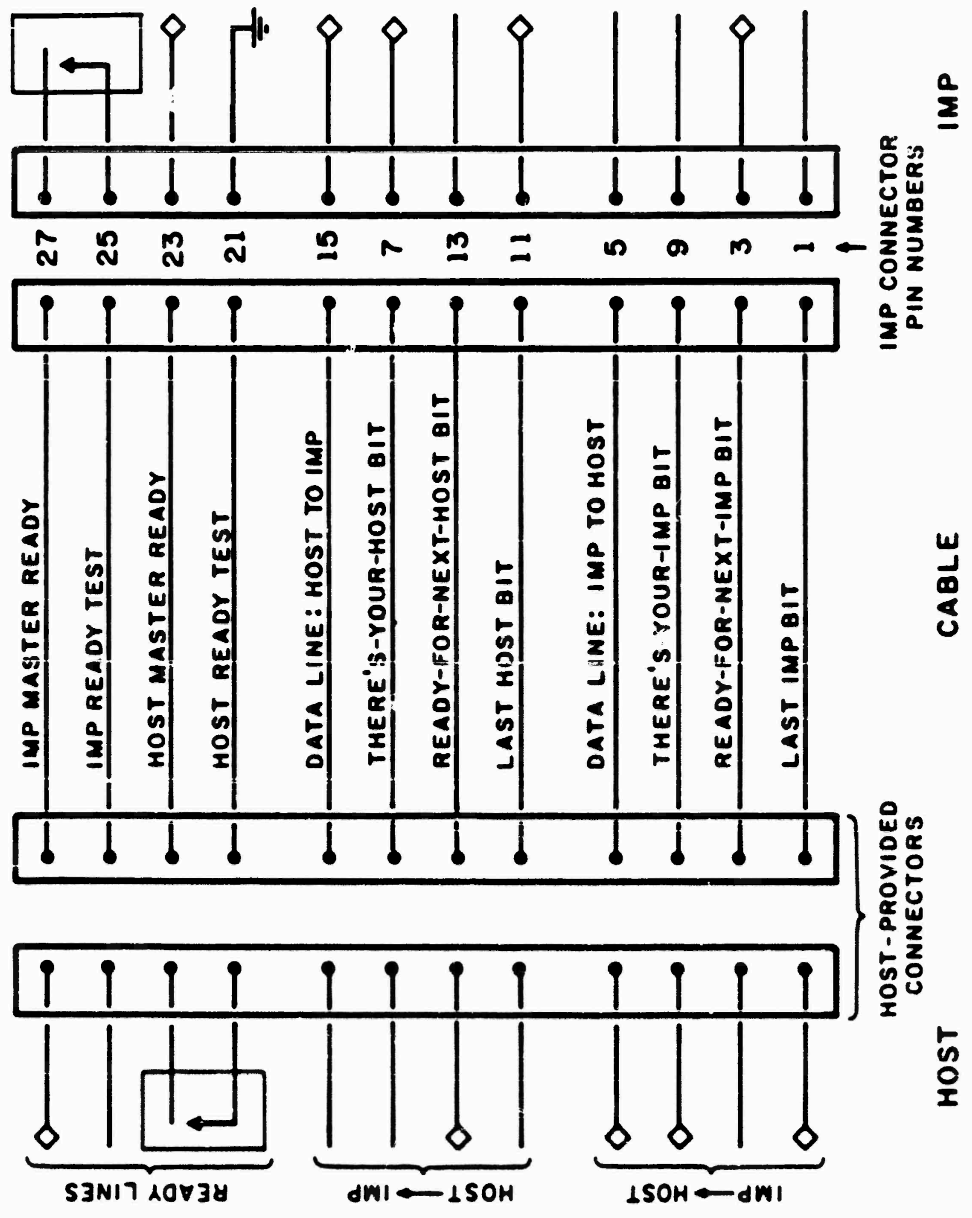
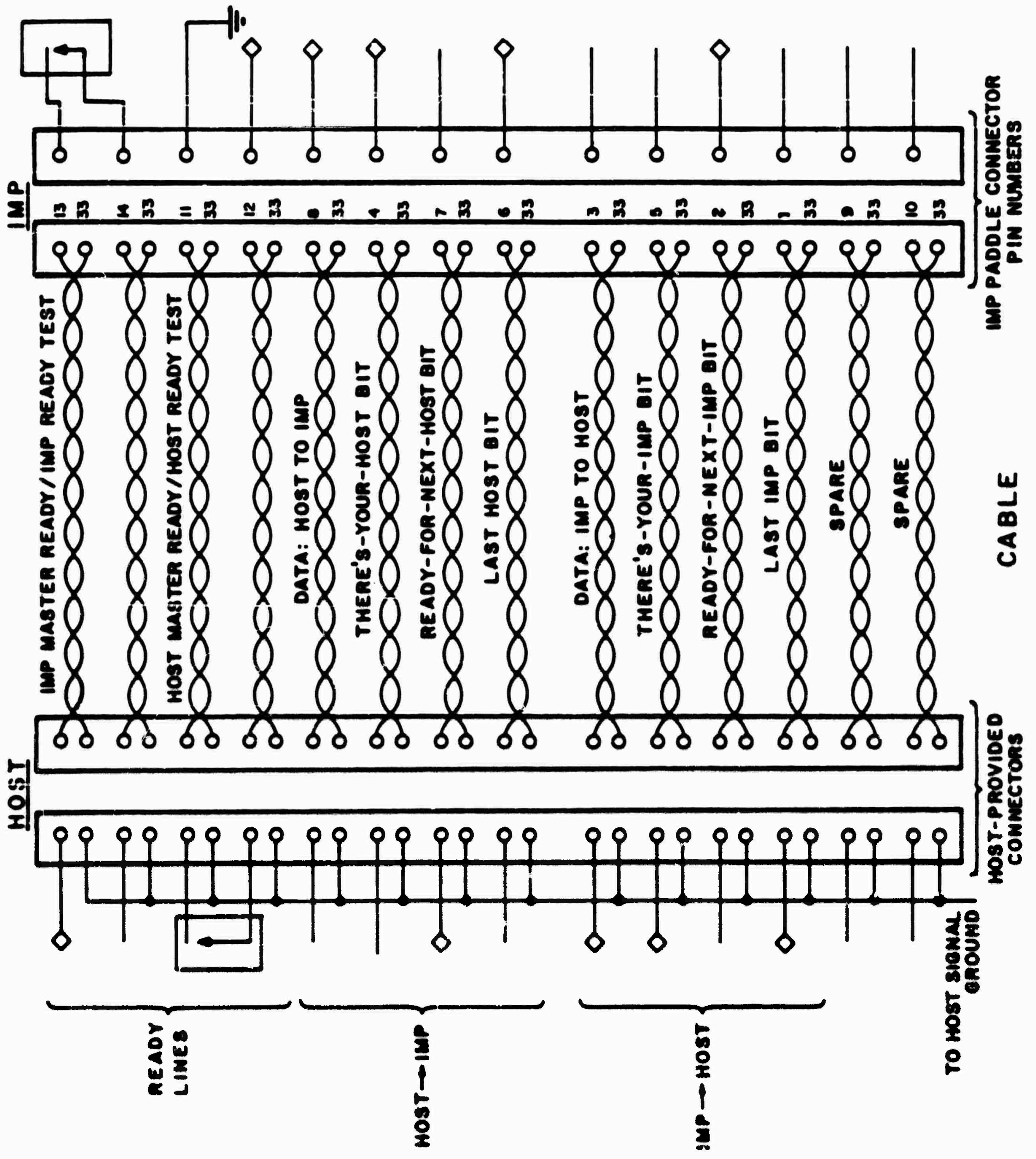
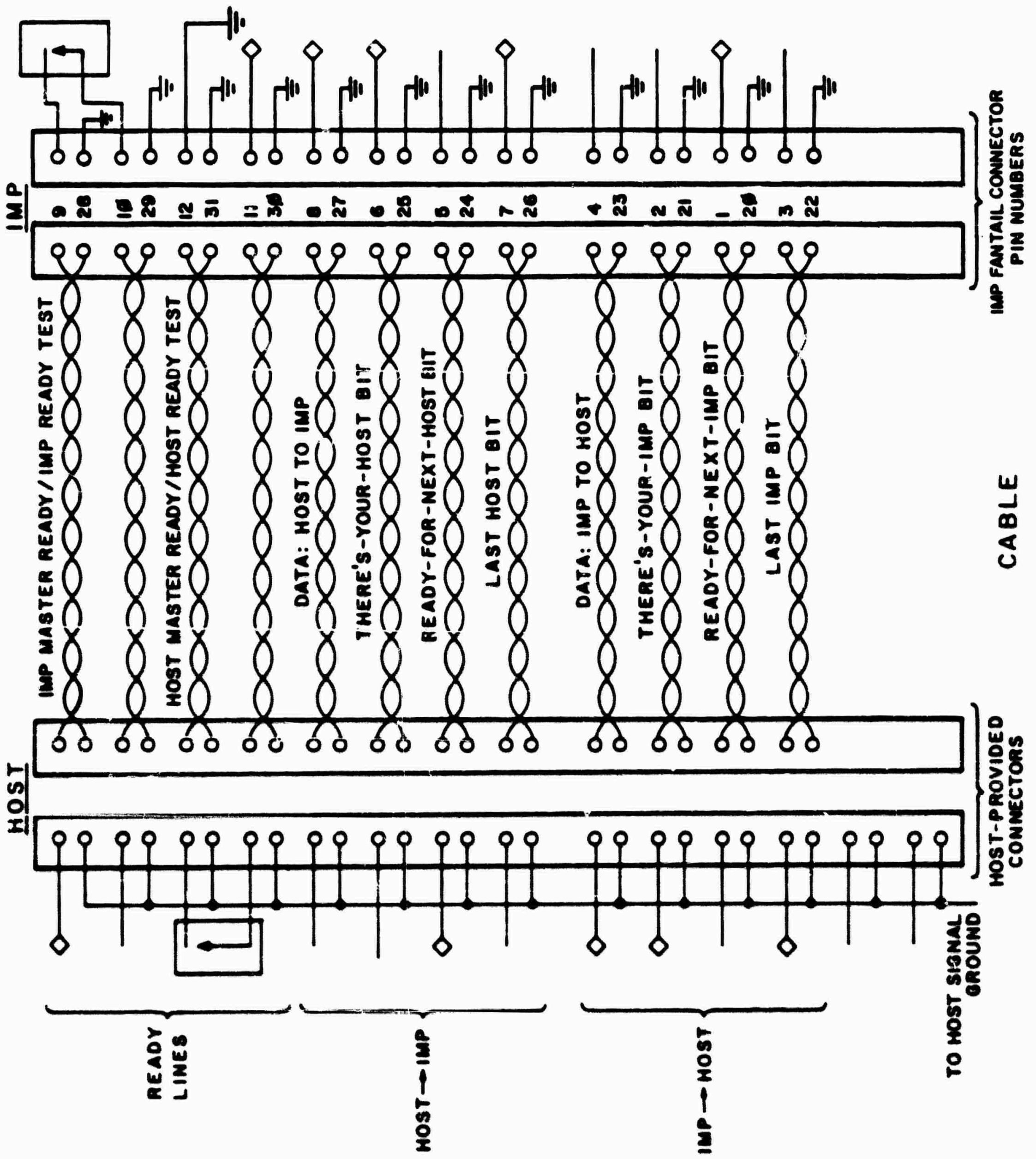


Figure 4-4 516 Host Cable Signals (Local Host)


IMP
HOST
13 33 14 33 11 33 12 33 8 33 4 33 7 33 6 33 3 33 5 33 2 33 1 33 9 33 10 33
IMP PADDLE CONNECTOR PIN NUMBERS
IMP MASTER READY/IMP READY TEST
HOST MASTER READY/HOST READY TEST
DATA: HOST TO IMP
THERE'S-YOUR-HOST BIT
READY-FOR-NEXT-HOST BIT
LAST HOST BIT
DATA: IMP TO HOST
THERE'S-YOUR-IMP BIT
READY-FOR-NEXT-IMP BIT
LAST IMP BIT
SPARE
SPARE
CABLE
HOST-PROVIDED CONNECTORS
TO HOST SIGNAL GROUND
READY LINES
HOST→IMP
IMP→HOST


IMP
HOST
9 28 10 29 12 31 11 30 8 27 6 25 5 24 7 26 4 23 2 21 1 20 3 22
IMP MASTER READY/IMP READY TEST
HOST MASTER READY/HOST READY TEST
DATA: HOST TO IMP
THERE'S-YOUR-HOST BIT
READY-FOR-NEXT-HOST BIT
LAST HOST BIT
DATA: IMP TO HOST
THERE'S-YOUR-IMP BIT
READY-FOR-NEXT-IMP BIT
LAST IMP BIT
CABLE
IMP FANTAIL CONNECTOR PIN NUMBERS
HOST-PROVIDED CONNECTORS
TO HOST SIGNAL GROUND
READY LINES
HOST → IMP
IMP → HOST

connections are pictured in Figure 4-6. All shields should be connected to signal ground in the Host. DC amplifiers are used for line driving and, by this means, we expect to couple the signal ground systems as tightly as possible.

The Host cable supplied with the 316 IMP is 30 feet long and contains 32 twisted pairs. The cable is terminated at the IMP end with a paddle card which plugs directly into the 316 Host interface. Each pair of the cable consists of a colored wire and a black wire numbered with the pin number of the paddle card to which the colored wire is connected. All black wires connect to the paddle card signal ground. Host personnel must provide an appropriate connector for the Host end of the cable. The wires are assigned as in Figure 4-5. All twisted pair grounds should be connected to signal ground in the Host. DC amplifiers are used for line drivers and the signal ground systems of the Host and IMP should be as highly coupled as possible.

4.5.2 Connection to a Distant Host

Connection to a distant Host necessitates the use of balanced lines and requires that a Host pay careful attention to differentials in ground potential. The distant Host's special interface must provide balanced drivers and receivers. Ground isolation is provided by the IMP.

The Host must supply a shielded cable containing multiple twisted pairs of #20 (or heavier) gauge wire. The characteristic impedance (Z0) must be approximately 120 ohms. The wires may be either individually shielded or may have a single shield covering all pairs. The shield is used to carry the Host's ground reference and should have very low resistance. There must be at last 10 pairs in the cable, and we strongly recommend that at least two spare pairs be carried (see Figure 4-7). A suitable cable is Direct Burial Cable, REA Specification PE-23, 19AWG conductors, 12 pairs.

At the IMP side the cable must be terminated in an MS24266R18B31PN (Amphenol 48-16R18-31P) plug with an MS27291-5 clamp and MS24254-20P contacts for 316 and 516 IMPs. For distant host connections to Pluribus IMPs, the cable must be terminated in a Cinch DC-37P or equivalent, with Cannon D-110278 disc latches and (recommended) Amphenol 17-1373 shell and strain relief. The outer sheath of the cable should be stripped back at least two feet to allow flexibility at the connector when the DC-37P is used. Pair and pin assignments are shown in figures 4-7A (316 and 516) and 4-7B (Pluribus). Note that the cable shield(s) should be connected to connector pins as specified, and NOT to the connector shell. For Pluribus installations where the site already has a cable terminated in the Amphenol MS style connector, BBN can supply a short adapter cable having appropriate connectors on both ends.

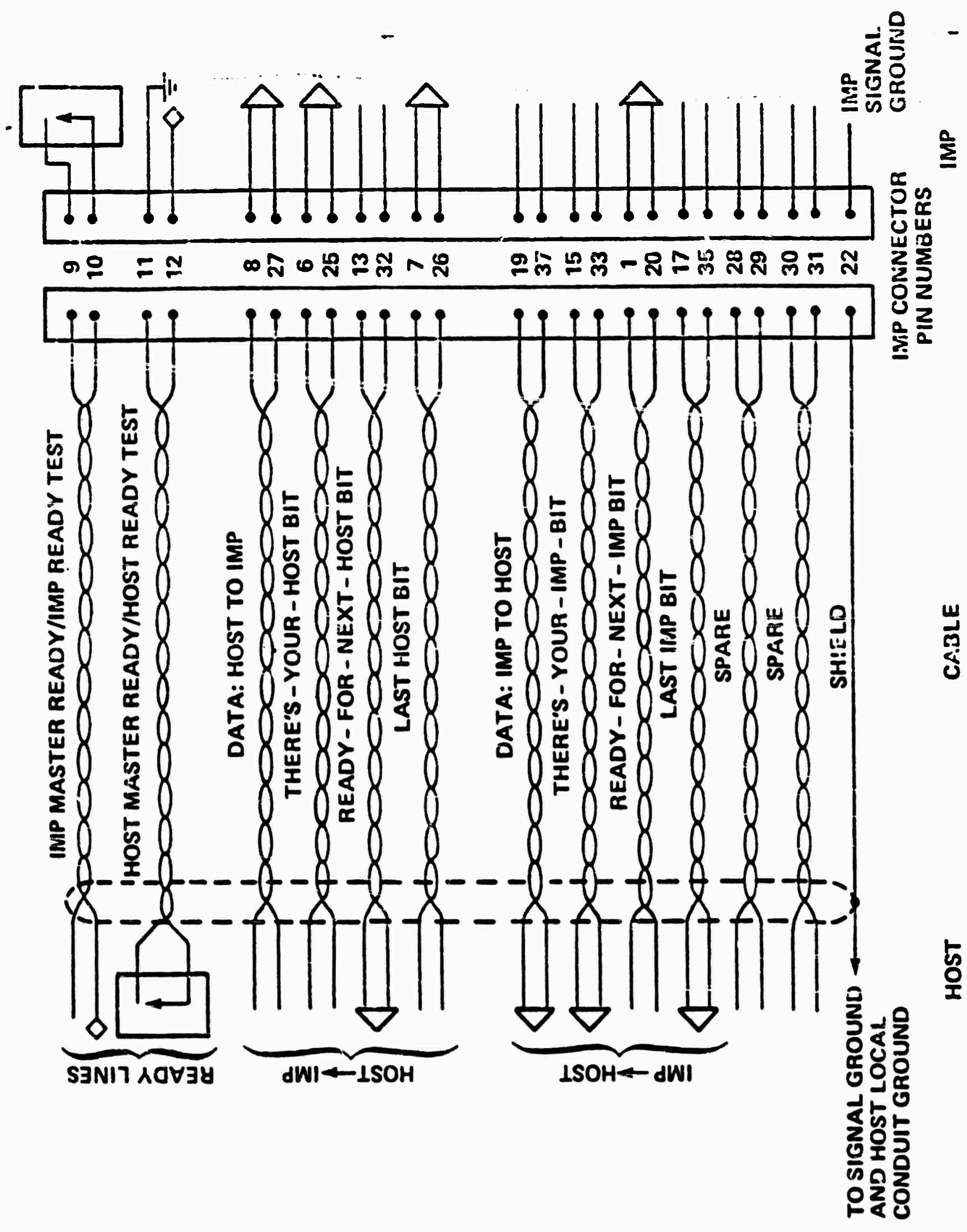


Figure 4-7 B Pluribus IMP Cable Signals (Distant Host)

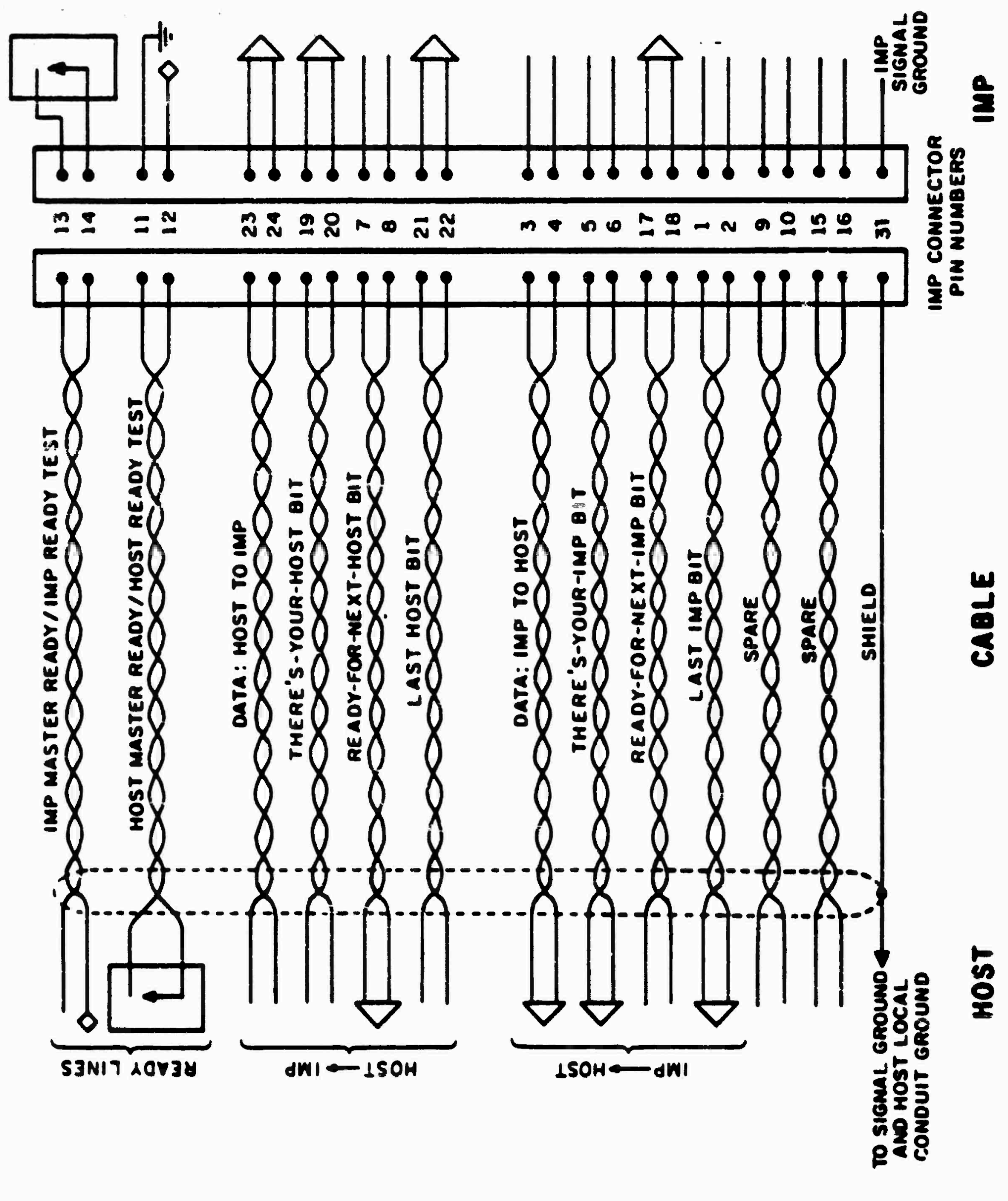


Figure 4-7 A Host Cable Signals (Distant Host)

the drivers and receivers used in the IMP are shown in Appendix D.

The Host should provide drivers and receivers similar to those used in the IMP. Use of these exact circuits is acceptable as is use of any other circuit capable of driving and receiving a differential signal of 1.0 volts centered around ground. Care should be taken to preserve proper signal polarity in the cable.

The cable shields should be very solidly connected to the Host's signal ground, which should be connected to the third-wire power ground at the Host computer. DC isolation is done at the IMP end of the cable to prevent significant currents from flowing through the shields. This isolation is accomplished by optically isolating the signals. All signals from the distant Host must have transition times of less than 100 nanoseconds, and must remain in each state for at least 1 usec between transitions.

The logic signals on the pairs of the cable (Data, Ready-For-Next-Bit There's-Your-Bit Last-Bit) are balanced voltage signals with each side terminated at the driver in 62 ohms to ground. Thus, the terminating impedance is 124 ohms and matches the cable impedance. The asserted logic signal drives the odd-numbered connector pin of each pair to +0.5 volts, and the other pin to -0.5 volts, producing a differential signal of 1.0 volt. The unasserted signal switches the polarity of this pair. There is no voltage drop across the cable since the receiver is unterminated. This produces a step reflection at the receiver which is absorbed at the transmitter. At the Host end the transmitters should terminate the cable in 120 ohms across each signal pair.

Standard 6-volt IMP logic signals are converted to differential signals by the line drivers and from differential signals to 6 volt logical signals by the receivers. Drawings for

APPENDIX A

Appendix A has been removed.
The information contained in
this appendix is obsolete.

APPENDIX B

RECOMMENDATIONS FOR HOST IMPLEMENTATION OF THE HOST/IMP INTERFACE

This Appendix recommends a plan for Host implementation of the Host/IMP interface, both the hardware and the lowest-level Host software which drives the hardware. In particular, the software discussed here has the tasks of input and output, detection of errors, and management of the Ready lines. Figures B-1 and B-2 provide simplified schematic drawings of the Host interface hardware.

B.1 Ready Line Philosophy

The actions which should be taken when transitions occur on the Ready line have the objective of reliably resynchronizing transmission after a temporary lapse of service, and possible loss of state information by either the IMP or the Host.

First, consider the IMP Ready line. When it drops, the IMP has suffered a possible loss of state, so the message in transit from the IMP to the Host (if any) is likely to be incomplete. Similarly, the message in transit from the Host to the IMP (if any) is likely to be incomplete. Both the Host and the IMP must recognize this explicitly by sending a message intended to be discarded (for example, a NOP) and discarding the message currently being received. (Note that the discardable message may be appended to some other message already partly received.)

The simplest arrangement for the Host's interface driver is a pair of processes, one sending messages and the other receiving

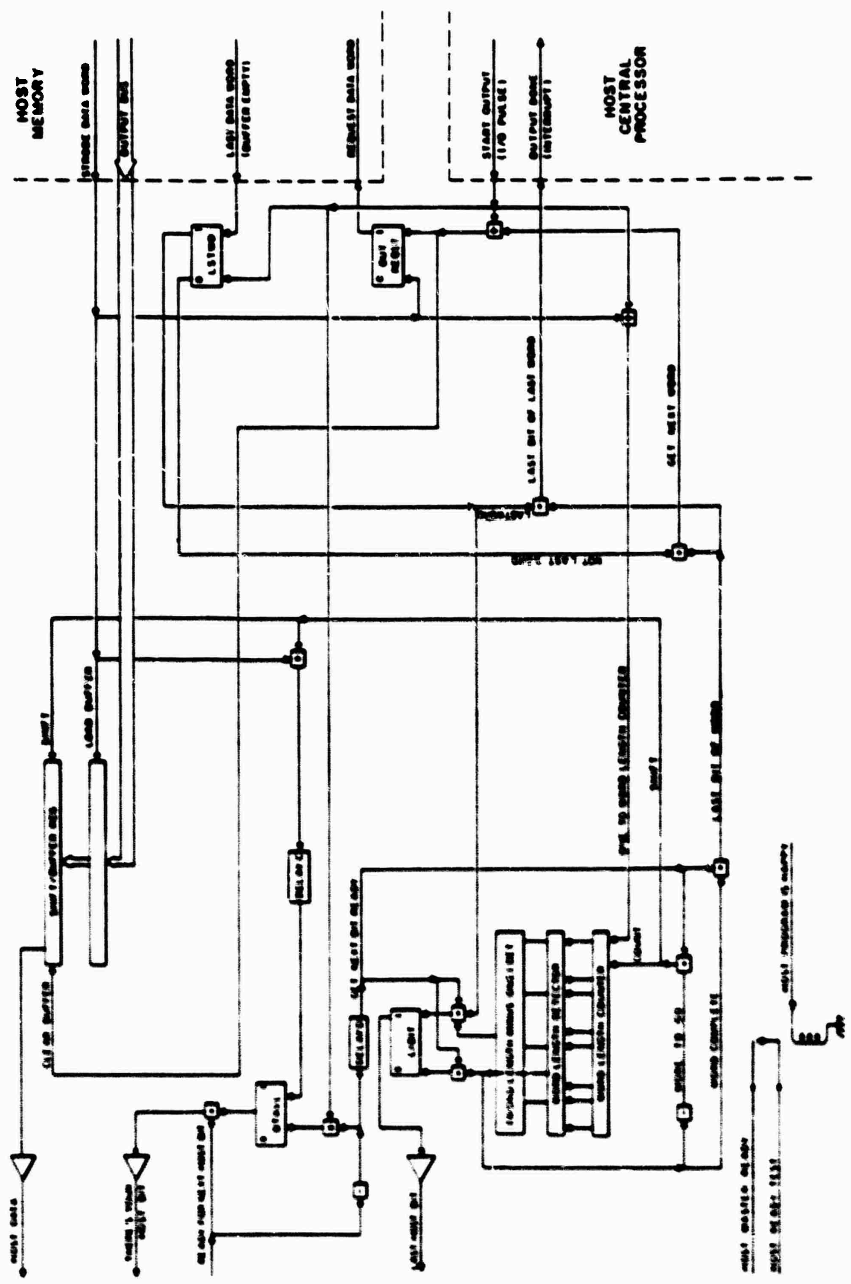


Figure B-1 Host-to-IMP (Host's Special Interface)

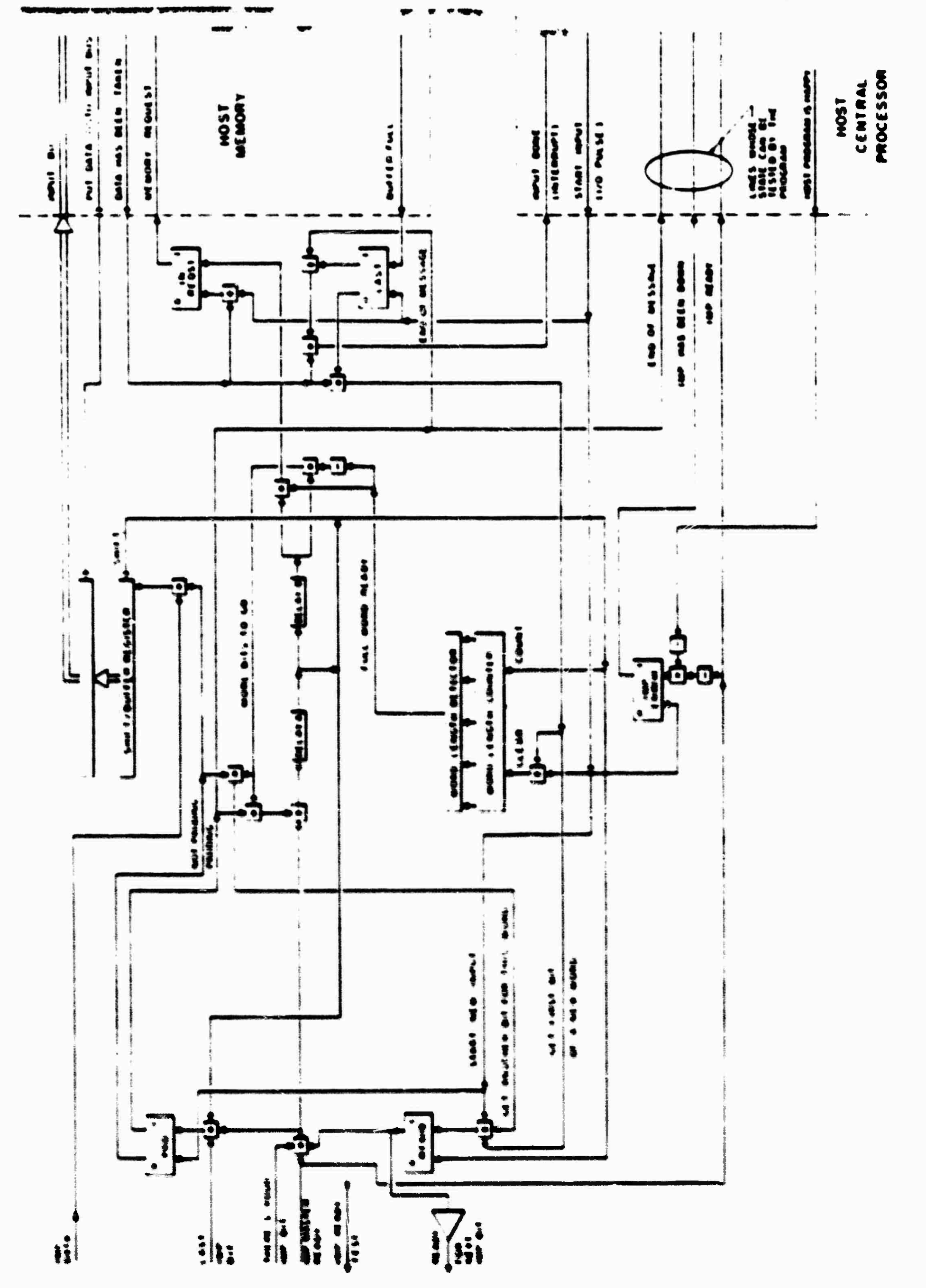


Figure B-2 IMP-to-Host (Host's Special Interface) messages. A drop of the IMP's ready line must be provided as an error status bit to each process. However, the two processes will need to clear this condition independently: the simplest implementation is an Input Error flop and an Output Error flop. Both flops are set by a drop of the IMP's ready line, and they are cleared independently under program control.

When the IMP raises its ready line, each contact bounce will again set the Error flops in the Host's interface. To insure that messages are not flowing across the interface at this time, assertions of the signals There's-Your-IMP-Bit and Ready-For-Next-Host-Bit have been delayed sufficiently in the IMP to guarantee that the IMP ready line has stabilized.

B.2 Programming the I/O Routines

System startup or restart requires the initialization step of clearing the Host Ready flop (driving the relay controlling the Host Ready line), waiting at least 1/2 second, and setting the Host Ready flop. Restarting the following two (asynchronous) interface driver processes will then properly resynchronize Host-IMP communication.

```
INPUT:  Wait until an input buffer is available
        Wait until IMP ready
        Start input
        Wait until input is complete
        IF Input Error
        THEN clear Input Error
                Comment: Discard erroneous message; reuse
                         buffer
        ELSE queue message on input queue
        GOTO INPUT
```

```
OUTPUT:  Wait until a message is present on output queue
         Wait until IMP ready
         Start output
         Wait until output is complete
         IF Output Error
         THEN clear Output Error
              Comment: Save erroneous message for
                       retransmission
         ELSE remove message from output queue
         GOTO OUTPUT
```

The <u>IMP</u> <u>Ready</u> line and error flops should only affect the processes above; this interface resynchronization should be invisible to both the process which interprets IMP-to-Host messages (it will be resynchronized by the IMP-to-Host type 10 message) and to any user software.

Actually, it is possible to share a single Error flop between the input and output processes by implementing Input Error and Output Error as software flags. A process testing for error is required (e.g., a mutual exclusion semaphore) to guarantee that only one process at a time is testing and modifying the flags. If the Error flop is set, the process must copy it into the other process' flag before clearing the flop and its own flag.

B.3 Host Ready Line Implementation

When the Host drops and raises its ready line, the IMP behaves in a fashion symmetric to that outlined above. Of course, this drop indicates that the state of the Host's interface driver, as well as the current incoming and outgoing messages, are likely to be lost. The appropriate action is triggered by setting the Error flop or flops in the Host interface, and the processes specified above will correctly resynchronize message flow in both directions. Of course, to guarantee that messages are not flowing across the interface while the Host ready line is undergoing contact bounce, the Host must delay transmission until its ready line has stabilized. This may be done in two ways:

Hardware: an integrating one-shot driven by the Host ready line flop is ANDed with There's-Your-Host-Bit and Ready-For-Next-IMP-Bit to guarantee that message transfer will not start until the ready flop has been on for 1/2 second.

Software: the initialization program executes a 1/2 second wait after setting the Host ready flop before permitting input or output to begin.

B.4 Summary of Ready Line Controls

The following summarizes the specification of the Host's Ready Line control:

1. The special Host interface contains a Host ready flop which drives a relay closure in the Host Ready line. This flop is set and cleared under program control.

2. The special Host interface detects the IMP's ready signal and sets a program-readable status condition (not an "interrupt" condition).

3. The special Host interface contains one or two error flops set when either the Host Ready flop is off or the IMP ready signal is off. The flop (flops) is a program-readable and program-clearable status condition (but not an interrupt condition). These status flops must not be cleared by system initialization.

4. If hardware stabilization of the Host's Ready line is provided, it is a 1/2 second integrating one-shot driven by the Host Ready flop. This signal is ANDed with There's-Your-Host-Bit and Ready-For-Next-IMP-Bit.

APPENDIX C

LOCAL HOST CONNECTION

ELECTRICAL CHARACTERISTICS

All Host-IMP logic signals (Data, Ready-For-Next-Bit, There's-Your-Bit, Last Bit) are unbalanced, source-terminated lines with a nominal characteristic impedance of 68 ohms. The line is terminated at the driving end with the characteristic impedance. The receiver is ideally an open circuit; in practice, it is a single TTL gate. In this scheme a voltage step of half the nominal level is propagated from source to receiver. At the receiver, it is reflected by the high impedance termination, resulting in a full level step at the receiver and another half level step propagating back to the source, where it is absorbed by the termination.

Voltage levels for drivers and receivers used by the IMP are given below:

		Min	Typ	Max
<u>Pluribus</u>				
	V_{OH}	4.1	5.0	
	V_{OL}		0.07	0.1
	V_{IH}		1.7	2.0
	V_{IL}	0.6	0.9	

316/516

V_{OH}	3.5	4.5	
V_{OL}		0.2	0.35
V_{IH}		1.55	2.5
V_{IL}	1.1	1.35	

The IMP will properly receive 5-volt logic signals; however, signals from the IMP may go to 6 volts. Therefore, the Host must provide a voltage divider, if these signals are to be received by normal 5-volt logic, to prevent destruction of the receiving circuit.

APPENDIX D

DRIVER RECEIVER FOR DISTANT HOST

Printed with permission of Honeywell, Inc.
Computer Control Division, Framingham, Massachusetts
Descriptions and schematic diagrams reflect use in the IMP.

D.1 Differential Receiver PAC Model CC-124

The Differential Receiver PAC, Model CC-124, contains three identical and independent circuits. Each circuit takes a bipolar differential signal and converts it to standard single-ended u-PAC logic levels. The schematic diagram (Figure D-1) reflects the use of this PAC in the IMP.

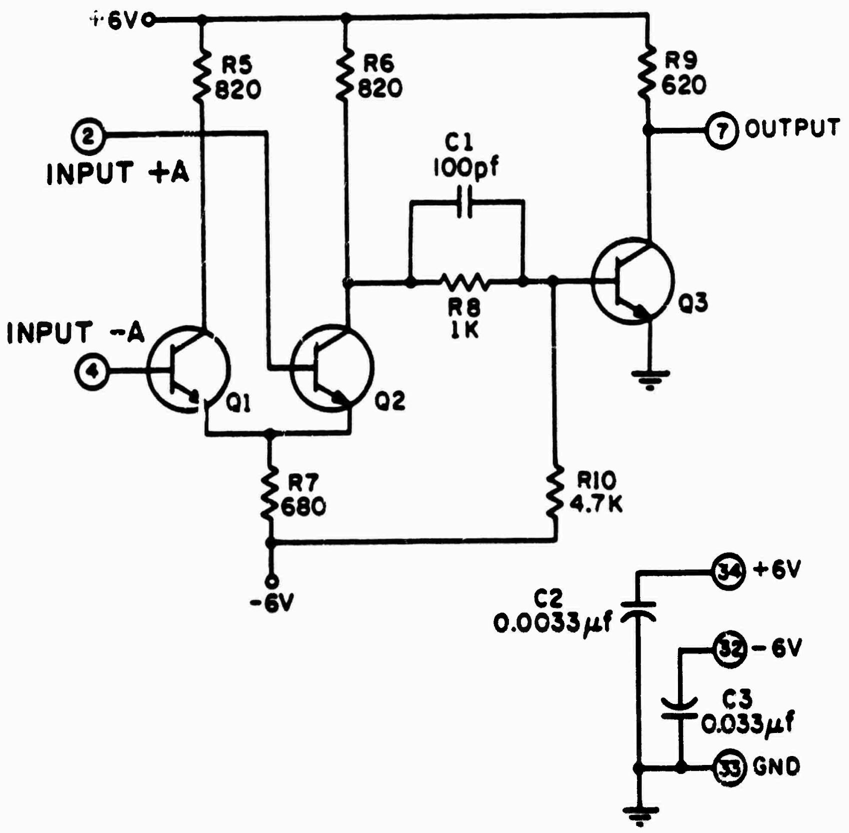
D.1.1 Circuit Description

The circuit functions as both a Differential Amplifier and Discriminator.

When the "+A" input is more positive than the "-A", Q3 is cut off and the output is a logic "1". When the polarity of the input signal is reversed, or "-A" is made more positive than "+A", then Q3 is turned on and the output goes to logic "0".

D.1.2 Terminating Network

The input to the CC-124 is unterminated. The terminating network in the PAC is not used.



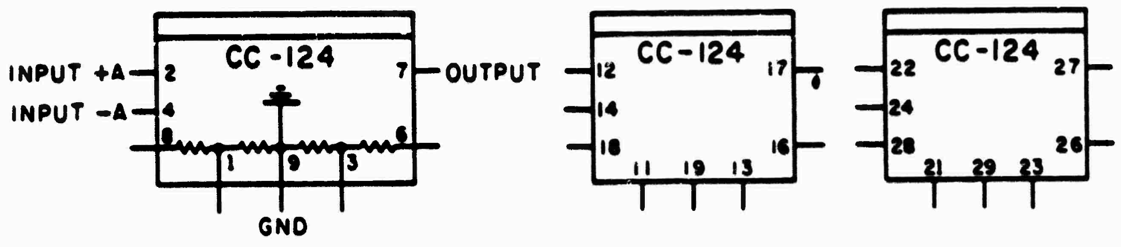


Figure D-1 Differential Receiver PAC Model CC-124, Schematic Diagram and Logic Symbol. (Shown as connected in IMP)

D.1.3 Specifications

Frequency of Operation: DC to 5 MC.

Input: Differential signal, 0.1V to 4.0V.

Input Impedance: 2.5K (min.)

Common Mode Rejection: Greater than ±2.5V

Output Drive Capability: 8 unit loads and 70 pf stray capacitance each.

Circuit Delay: 30 nsec (max.).

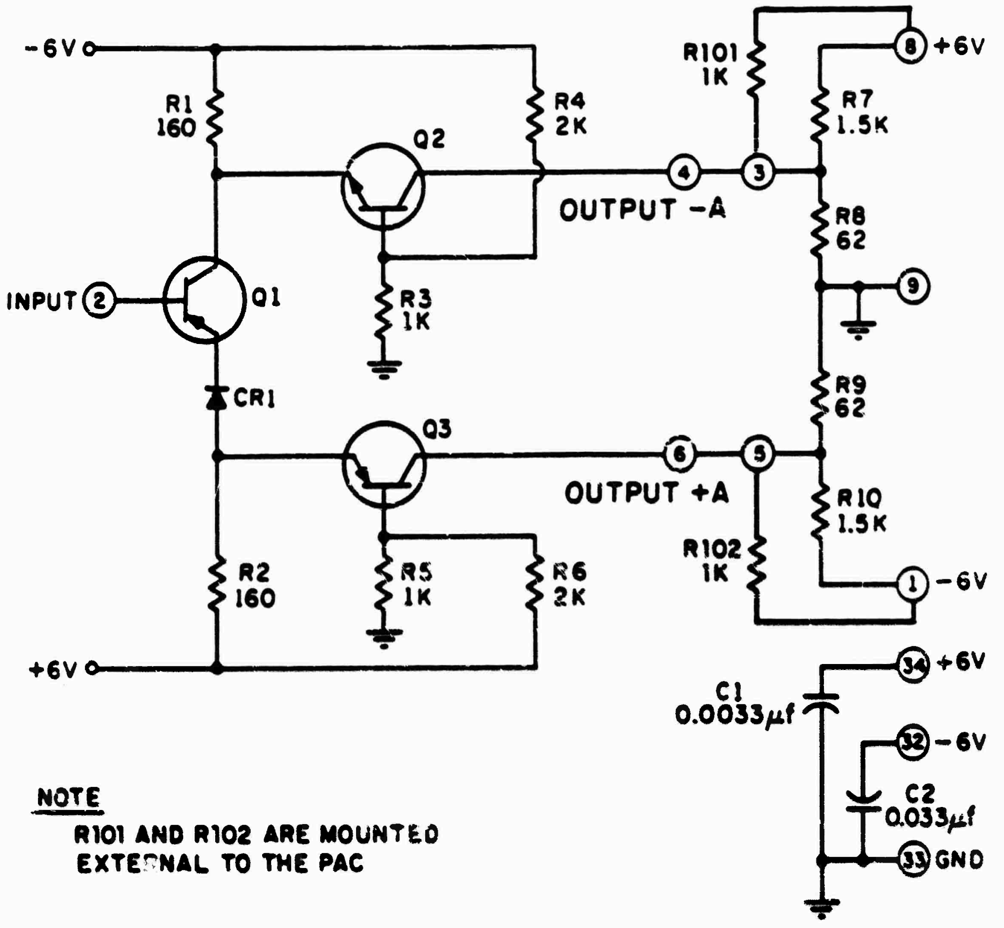
Current Requirements (exclusive of terminators):

+6V: 60 ma (max.).

-6V: 30 ma (max.).

D.2 Differential Line Driver PAC Model CC-125

The Differential Line Driver PAC, model CC-125, contains three identical and independent circuits. Each circuit will switch approximately 18 ma into a balanced load when a standard u-PAC logic level of "1" is present at the input. When the input is at logic "0", the output is open circuited. The schematic diagram (Figure D-2) reflects use of this PAC in the IMP. Note that the circuit has been modified by the addition of externally (i.e., back panel) mounted resistors.



NOTE
R101 AND R102 ARE MOUNTED
EXTERNAL TO THE PAC

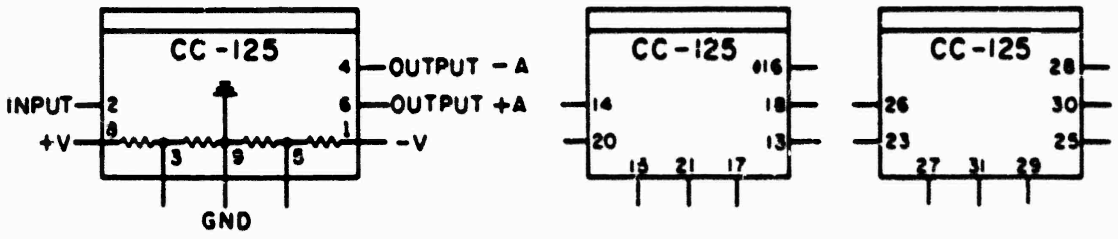


Figure D-2 Differential Line Driver PAC Model CC-125. Schematic Diagram and Logic Symbol. (Shown as connected in IMP)

D.2.1 Circuit Function

When the input is at ground or logic "0", Q_1 is biased "on". With Q_1 "on", the emitters of Q_2 and Q_3 are biased "off", and the output is effectively open-circuited.

When the input is open or at logic "1", Q_1 is turned "off", which causes Q_2 and Q_3 to turn on, switching approximately 18 ma into the output.

D.2.2 Terminating Network

The terminating network consists of resistors R7-R10 as well as the externally mounted resistors R101 and R102, and is designed for use with 100 to 140 ohm, balanced, twisted-pair transmission lines. With a logic "0" applied to the input of the transmitter, the terminating network establishes a 1.0V differential signal on the transmission line pair. When a logic "1" is applied to the input of the transmitter, the polarity of the 1-volt differential signal on the transmission line pair will be reversed.

D.2.3 Specifications

Frequency of Operation: DC to 5 MC.

Input Loading: 1 unit load each.

Output Drive Capability: Approximately 18 ma into a balanced load.

Circuit Delay: 15 nsec (max.).

Current Requirements: Exclusive of terminators

+6V: 90 ma (max.).

-6V: 90 ma (max.).

The combination of the internal terminator network and the externally connected resistors will draw about 9 ma each when connected to +6V and -6V.

APPENDIX E

ASCII CODES

ASCII CODES

ASCII		MNEMONIC	SYMBOL
OCT	HEX		
200	80	NUL	↑@
201	81	SOH	↑A
202	82	STX	↑B
203	83	ETX	↑C
204	84	EOT	↑D
205	85	ENQ	↑E
206	86	ACK	↑F
207	87	BEL	↑G
210	88	BS	↑H
211	89	HT	↑I
212	8A	LF	↑J
213	8B	VT	↑K
214	8C	FF	↑L
215	8D	CR	↑M
216	8E	SO	↑N
217	8F	SI	↑O
220	90	DLE	↑P
221	91	DC1	↑Q
222	92	DC2	↑R

ASCII			
OCT	HEX	MNEMONIC	SYMBOL
223	93	DC3	↑S
224	94	DC4	↑T
225	95	NAK	↑U
226	96	SYN	↑V
227	97	ETB	↑W
230	98	CAN	↑X
231	99	EM	↑Y
232	9A	SUB	↑Z
233	9B	ESC	↑[
234	9C	FS	↑\
235	9D	GS	↑]
236	9E	RS	↑↑
237	9F	US	↑_
240	A0	SP	Space
241	A1		!
242	A2		"
243	A3		#
244	A4		$
245	A5		%
246	A6		&

ASCII			
OCT	HEX	MNEMONIC	SYMBOL
247	A7		'
25Ø	A8		(
251	A9		)
252	AA		*
253	AB		+
254	AC		,
255	AD		-
256	AE		.
257	AF		/
26Ø	BØ		Ø
261	B1		1
262	B2		2
263	B3		3
264	B4		4
265	B5		5
266	B6		6
267	B7		7
27Ø	B8		8
271	B9		9
272	BA		:

ASCII			
OCT	HEX	MNEMONIC	SYMBOL
273	BB		;
274	BC		<
275	BD		=
276	BE		>
277	BF		?
300	C0		@
301	C1		A
302	C2		B
303	C3		C
304	C4		D
305	C5		E
306	C6		F
307	C7		G
310	C8		H
311	C9		I
312	CA		J
313	CB		K
314	CC		L
315	CD		M
316	CE		N

ASCII		MNEMONIC	SYMBOL
OCT	HEX		
317	CF		O
320	D0		P
321	D1		Q
322	D2		R
323	D3		S
324	D4		T
325	D5		U
326	D6		V
327	D7		W
330	D8		X
331	D9		Y
332	DA		Z
333	DB		[
334	DC		\
335	DD		]
336	DE		^
337	DF		_
340	E0		`
341	E1		a
342	E2		b

ASCII OCT	ASCII HEX	MNEMONIC	SYMBOL
343	E3		c
344	E4		d
345	E5		e
346	E6		f
347	E7		g
350	E8		h
351	E9		i
352	EA		j
353	EB		k
354	EC		l
355	ED		m
356	EE		n
357	EF		o
36Ø	FØ		p
361	F1		q
362	F2		r
363	F3		s
364	F4		t
365	F5		u

ASCII			
OCT	HEX	MNEMONIC	SYMBOL
366	F6		v
367	F7		w
370	F8		x
371	F9		y
372	FA		z
373	FB		{
374	FC		¦
375	FD		}
376	FE		‾
377	FF	DEL	RUBOUT

The IMP uses 8-bit ASCII with the left-most bit set to one.

↑ = control

↑@ - shift control - P

APPENDIX F

VERY DISTANT HOST INTERFACE

[OBSOLETE]

APPENDIX G

IMP POWER WIRING CONVENTION

[OBSOLETE]

APPENDIX H

INTERFACING A HOST TO A PRIVATE LINE INTERFACE

[OBSOLETE]

APPENDIX J

C/30 HDH INTERFACE SPECIFICATION

12/83

Table of Contents

1 Overview

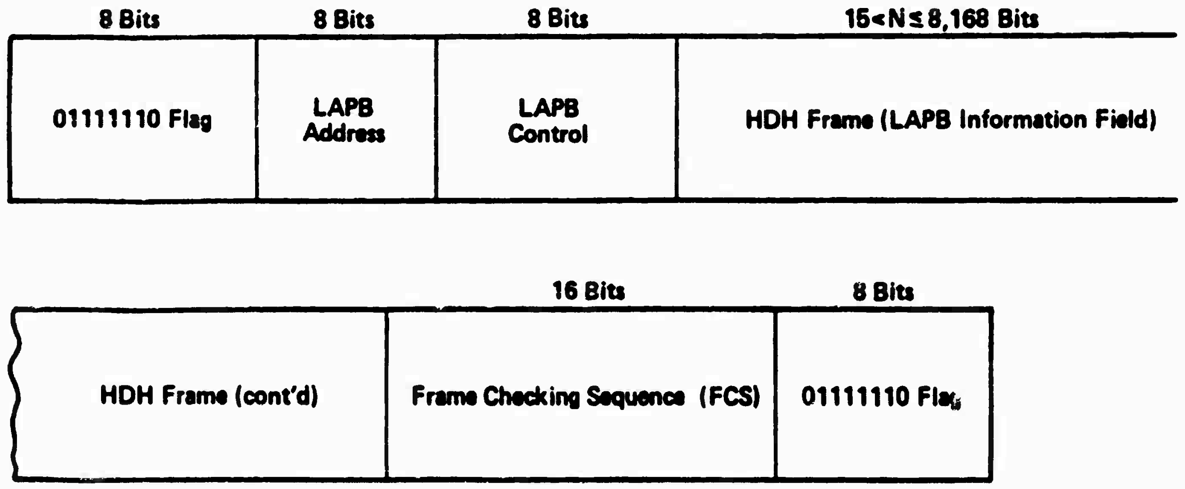
This report specifies the attachment of an HDH host to a BBN Communications Corporation (BBNCC) C/30 Packet Switching Node (IMP). This report assumes that the reader is familiar with CCITT Recommendation X.25 and FIPS 100/Fed. Std. 1041. For the purposes of the ensuing discussion, references to the DCE refer to that part of the IMP which interfaces with the host. References to the DTE refer to that part of the host entity which interfaces with the IMP.

The HDH interface protocol is a combination of protocol layers including LAPB, HDH itself. and 1822. HDH is defined as four independent architectural levels that fit into the Open Systems Interconnection (OSI) Reference Model in the following way:

Level 1:	The PHYSICAL level of the connection. The physical, electrical, functional, and procedural characteristics to activate, maintain, and deactivate the physical link between the DTE and the DCE. (See J.2)
Level 2:	The LINK level of the connection. The link access procedure for data interchange across the link between the DTE and the DCE. The HDH IMP uses the LAPB subset of HDLC.
Level 3:	The PACKET level of the connection. The packet format and control procedures for the exchange of packets containing control information and user data between the DTE and the DCE is provided by the HDH protocol. The 1822 protocol provides procedures for the exchange of these packets between the DTE and a remote DTE.

HDH has a packet mode option and a message mode option. In the packet mode option,data from several LAPB information (I) frames are combined to form one 1822 message. In the message mode option the entire 1822 message is held in one LAPB I frame. A diagram of an HDH frame within a LAPB I frame is shown in Figure J-1. The LAPB subset of the HDLC protocol is specified in the CCITT Recommendation X.25 1980, which is provided in Appendix K of this document. Section 3 of this appendix provides general information about the BBN C/30 LAPB implementation and Appendix J.1 provides notes, organized by reference to the CCITT X.25 document, which help to clarify the functional link level characteristics. The HDH protocol is described in Section 4 and an HDH host finite state machine is provided in Section 5. Diagrams of the HDH protocol formats are shown in Figures J-2 and J-3.

Each 1822 message is fragmented into one or more LAPB frames. Each LAPB frame includes 2 octets of LAPB header and 2 octets of HDH header. The HDH protocol is inserted between LAPB and 1822 to act as an access protocol. HDH's functions are to segment 1822 messages into LAPB frames, rebuild messages from the frames, perform line quality monitoring, simulate the Host-IMP ready line up/down signalling which is handled by hardware mechanisms in the 1822 hardware interface, and operate the loopback and reflect modes. Only the first LAPB frame of an HDH message contains an 1822 leader. The rest of the frames, if



Note: Only One Flag Is Required Between LAPB Frames

Figure J-1 HDH Frame Within a LAPB Information Frame

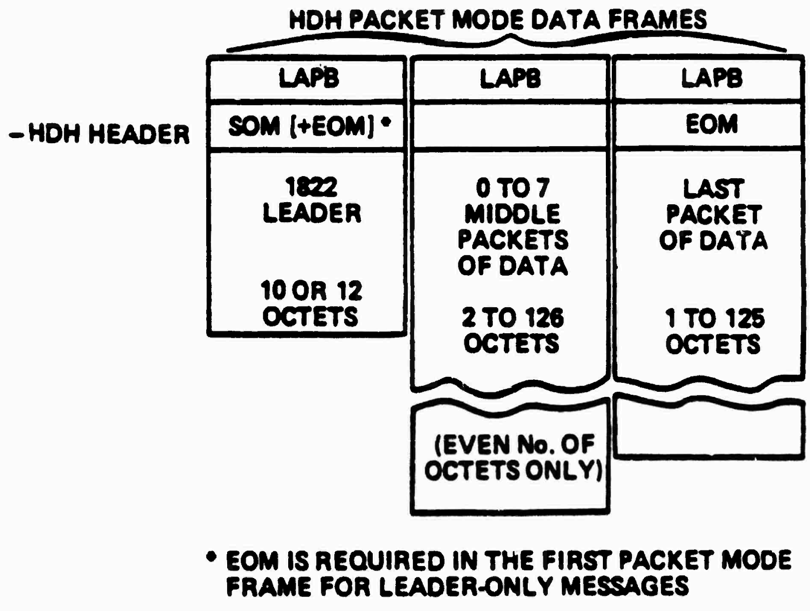

HDH PACKET MODE DATA FRAMES
LAPB
LAPB
LAPB
-HDH HEADER
SOM [+EOM]*
EOM
1822 LEADER
10 OR 12 OCTETS
0 TO 7 MIDDLE PACKETS OF DATA
2 TO 126 OCTETS
(EVEN No. OF OCTETS ONLY)
LAST PACKET OF DATA
1 TO 125 OCTETS
* EOM IS REQUIRED IN THE FIRST PACKET MODE FRAME FOR LEADER-ONLY MESSAGES

Figure J-2 HDH Packet Mode Format

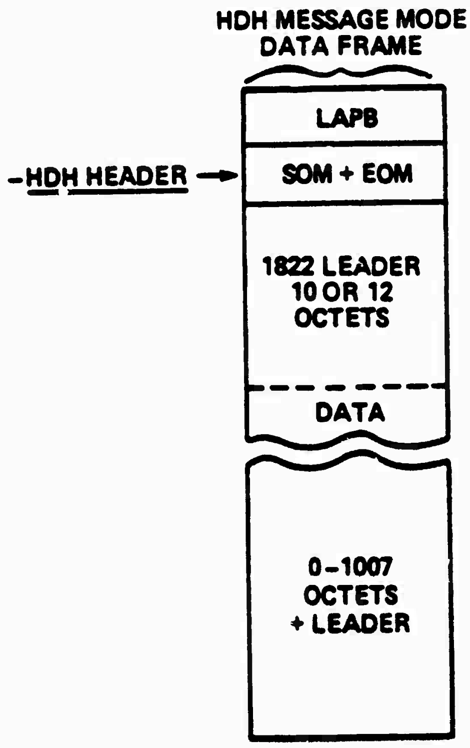


Figure J-3 HDH Message Mode Format

any, contain LAPB and HDH headers in addition to 1822 message data.

1.1 Choosing the Correct HDH Interface

The choice between HDH packet mode or HDH message mode is entirely site dependent; therefore, the choice is left up to the IMP site manager. The following is a short discussion of the factors which may affect the decision.

Message mode allows for rather large single frame messages, while packet mode allows large messages to be broken up into smaller message frames. Hosts using message mode must therefore provide enough ready buffer space for the largest possible messages (1021 octets excluding the LAPB header). This can be done with large buffers or, if the host has a scatter/gather DMA channel, through buffer chains. Packet mode allows the host to divide its memory resources into smaller buffers because messages are broken into a number of LAPB frames whose maximum size is limited to 128 octets (excluding the LAPB header). The Level 2 (LAPB) window of seven might limit throughput more with shorter Packet Mode packets than with long Message Mode packets.

1.2 Hardware Requirements

This report specifies the interface presented by a BBN Communications Corporation C/30 Processor, together with specific commercial software products. DTE attachment requires the C/30 MSYNC I/O interface board. To support the HDH interface described in this report. the C/30 must be configured with at least 64 thousand (64K) words of main memory.

1.3 Summary of Features

The C/30 IMP supports three physical interface standards, RS-232C, RS-449, and V.35, at clock rates from 1.2 to 64 kilobits per second (Kb/s). Physical line status is sampled using incoming status interchange circuits.

The C/30 IMP link level interface supports both LAP and LAPB procedures. Link level status monitoring during LAPB operation is accomplished using a transparent idle-link polling mechanism. DCE link level parameters are configurable.

The C/30 HDH packet level interface supports either Packet Mode or Message Mode transfer formats.

2 Physical Level Specifications

The C/30 HDH IMP physical level specification is in conformance with FIPS 100/Fed. Std. 1041 and CCITT Recommendation X.25. This section presents additional information.

An HDH DTE attached to a C/30 HDH IMP may be collocated with the IMP or may be connected to it via an access line. In all cases the DTE presents a physical DTE interface; the network administration must supply the matching DCE interface. By appropriate choice of modem or modem-like hardware, an administration could offer up to four physical level interfaces: RS-232-C (CCITT V.28), RS-449, both balanced and unbalanced (CCITT V.11 and V.10, respectively; also MIL-188-114 balanced and unbalanced), and CCITT V.35. Appendix J.2 of this document describes in detail the choices of physical interface that might be made available to a host DTE attached to a C/30 HDH IMP and the specifications for each type of interface. Table J.1 summarizes the physical interfaces available at each data rate supported by the C/30 HDH DCE.

An HDH DTE may implement more than one signalling rate. At each signalling rate implemented, the DTE must offer at least one of the physical interface options listed as "A" (available) for that rate in Table J.1. DTE implementors are encouraged to offer the widest variety of signalling rates and physical interfaces practical to maximize the flexibility of use of their equipment with the C/30 HDH IMP.

Physical Interface	Signalling Rate in Kb/s 1.2 to 9.6	14.4, 19.2	48 and above
RS-232-C (and equiv.)	A	A	-
RS-449/422 balanced (and equiv.)	A	A	A
RS-449/423 unbalanced (and equiv.)	A	-	-
CCITT V.35	-	-	A

Legend

A = Available
- = Not available

(Taken from Appendix J.2, Table 3)

Table J.1 C/30 HDH Physical Signalling Rates and Interfaces

3 Link Level Procedures

C/30 HDH link level procedures are as specified by CCITT Recommendation X.25 (see Appendix K) and FIPS 100/Fed. Std. 1041. This section presents additional information.

3.1 Link Level Parameters and Options

1. The default value of K, the maximum number of sequentially numbered I frames that the DCE will have outstanding (unacknowledged) at any given time, is seven. The Administration may configure a C/30 HDH DCE to have an optional value of K from one to six.

2. The default value of N2, the maximum number of transmissions and retransmissions of a frame by the DCE following the expiration of the T1 timer, is twenty. The Administration may configure a C/30 HDH DCE to have an optional value of N2 from one to 200.

3. The optional 32-bit Frame Checking Sequence (FCS) is not supported.

3.2 Timer T1 and Parameter T2

The period of the timer T1 used by the C/30 HDH DCE reflects assumptions about the processing speed of the DTE. The DCE assumes that parameter T2, the response latency of the DTE to a frame from the DCE, is no greater than 1/2 second. Likewise, the DCE guarantees that its parameter T2, the latency in responding to frames from the DTE, is 1/2 second for signalling rates of 19.2 Kb/s or slower, and 1/4 second for faster links.

T1 may be computed to be 4X + T2, based on the assumptions that the link propagation time is negligible, the worst-case frame transmission time is X, and that timer T1 is started when a frame is scheduled for output, that each frame is scheduled just as transmission of the previous frame starts, that frames are not aborted, and that each frame and its predecessor are of maximum length N1 = 8184 bits for Message Mode or N1 = 1040 for Packet Mode.

As an example, for a signalling rate of 9.6 Kb/s, this yields X = .82 sec (for Message Mode). If T2 is .5 sec., the total time for the DTE to respond in the worst case should be 3.8 seconds. In fact, the DCE uses a T1 timer value of 4 seconds for a link speed of 9.6 Kb/s.

In no case does the DCE use a value for T1 smaller than 3 seconds. This means that, for faster links, the DTE's T2 parameter may be lengthened because the X term in the above formula is smaller. For links of 19.2 Kb/s or faster, DTEs are expected to satisfy latency requirements that allow the DCE to use the formula 4X + T2 < 3 seconds.

The DTE may choose any value for T1 that is compatible with the DCE's T2 parameter values. The value of T1 used by the DTE may always be set longer than the formula indicates, with the result that recovery from certain types of link errors will be slower. However, the DCE's parameter T2 cannot be reduced, so the formula should be viewed as yielding a lower bound on the DTE's T1 timer.

4 Packet Level Procedures

The packet level of the HDH interface is implemented in two protocol layers: HDH which is described below, and 1822 host/IMP protocol which is described in BBN Report 1822.

4.1 The HDH Protocol

The HDH protocol is implemented on IMPs to support the connection of hosts to the IMP using LAPB at the link level, and the Host IMP Protocol (1822) at the network level.

The interface inserts the HDH transmission protocol underneath the standard 1822 protocol, while using LAPB as the link level protocol. Therefore, each LAPB frame carries a two-octet HDH header plus an 1822 protocol message. In order to accommodate both present and future requirements, the HDH protocol has two modes as shown in Figures J-2 and J-3. One mode is called packet mode; it requires that the information part of the LAPB frame not exceed 128 octets (126 data + 2 for HDH header), and that the 1822 leader be sent in a separate frame. In this mode, the host and IMP explicitly break up messages into packets. The other mode, called message mode, permits the entire message (up to 1007 data octets, plus leader) to be sent in a single LAPB frame. The mode used by a particular IMP interface is set at subscription time, but may be changed by requesting a new configuration from the network operations center. The mode cannot be set dynamically via negotiations between the host and the IMP.

The HDH protocol is required at the local host-IMP connection in order to segment messages into packets, rebuild messages from packets, perform line quality monitoring, simulate the Host-IMP ready line up/down signalling, and operate the IMP loopback and reflect modes. The remainder of this section describes the HDH protocol. Whereas the IMP will implement the entire protocol, there are a few elements need not be implemented by the host, and those will be so noted.

4.1.1 HDH Data Messages

The host and IMP send each other LAPB frames using the formats illustrated in Figures J-1, J-2 and J-3. Each LAPB frame, in addition to the LAPB address and control octets, contains a 16-bit HDH header which is shown in Figure J-4. The header fields are described below. HDH Control Frames are shown in Figure J-5, and are described in section 4.1.2 .

	15	14	13	12	11	10	9 — 0
For Control Frame	1	SEQ	H/I	LIN	LQP	REF	Line Down Counter
For Data Frame	0	SEQ	H/I	SOM	EOM	REF	Octets in Rest of Frame

SEQ:	1 Signals Sequence Break
H/I:	1 If IMP Originated, 0 If Host Originated
LIN:	1 If Line Is Up, 0 If Down
SOM:	1 If Frame Is Start of Message
EOM:	1 If Frame Is End of Message
REF:	1 Signals Reflect Mode
LQP:	1 If Link Quality is Poor

Figure J-4 HDH Header Format

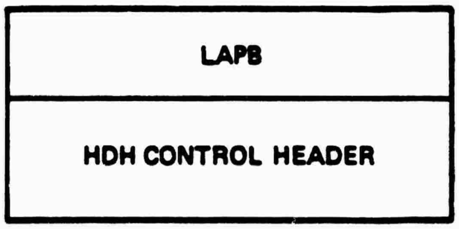


Figure J-5 HDH Control Frame - Packet or Message Mode

Bit 15 - Control bit

The high order bit of the HDH header distinguishes HDH control frames (to be discussed later) from data frames.

Bit 14 - Sequence Break (SEQ) bit

This bit, when set to one, indicates a sequence break in the frame stream between the previous frame and the current one (see description below).

Bit 13 - Host/IMP bit

This bit indicates if the frame was host or IMP originated. A zero indicates a host originated frame and a one indicates an IMP originated frame.

Bit 12 - Start of Message (SOM)

This bit, when set, indicates that this frame is the first frame in a message.

Bit 11 - End of Message (EOM)

This bit, when set, indicates that this frame is the last frame in a message.

Bit 10 - Reflect Mode (REF) bit

This bit indicates that the current frame sent by the IMP to the host must be looped back to the IMP unchanged. Only the IMP is allowed to set this bit (see below).

Bits 9-0 Octet Count

This field contains a count of octets in the rest of the frame (exclusive of the LAPB and HDH headers).

The sequence break bit (SEQ) is used by the host or IMP to signal to the other side that a discontinuity has occurred in the message stream. When either side receives an HDH frame with the SEQ bit set, it should discard any accumulated frames of messages up to the last EOM (i.e. the last complete message) and continue to discard new frames until the next start of message (SOM) is received. This has two practical uses in the HDH environment:

1. The IMP will send an HDH sequence break when it first brings the HDH level up to cause the host to discard any old message frames which might remain on the host's input queues. The host may send an HDH sequence break at this time, but it is not necessary since the IMP flushes its queues when the HDH level is taken down. Line just up is the only case where an HDH sequence break has meaning in message mode HDH.

2. In packet mode, the HDH sequence break is the mechanism by which one end can inform the other end that it detected a potential discontinuity in the message it was sending. For example, an internal reset in the host might cause such a data loss. Ideally, the host should send an HDH sequence break followed by a retransmission of the entire packet mode message starting with the SOM packet. If the host cannot keep a copy of the complete packet mode message due to buffer space limitations, it should just send the HDH sequence break followed by the next complete packet mode message. Higher layers must detect the data loss and cause retransmission in this case. It is important to send HDH sequence breaks only

when the desired recovery can be achieved.

The host/IMP bit is used by the IMP to confirm the loopback condition from the IMP to itself. The host should always send its frames with this bit zero, and the IMP with the bit set to one. The host need not implement a loopback mode, but it is recommended that in any case this bit be examined in order to detect possible inadvertent loopback conditions. The CCITT X.25 specification (Appendix K) does not address the issue of a functional loopback condition, and implies looped frames will always be discarded. In practice, however, it is possible to have a functional loopback by exchanging the LAPB addresses A and B on output, and this practice may be observed by various LAPB implementations (as it is in the IMP).

In packet mode, the SOM and EOM bits are used to delimit messages. In an 1822 type 0 message of zero length, or in a non-type 0 (1822 control, such as RFNM), both bits are turned on and the leader frame constitutes the entire message. Otherwise, only the SOM bit is turned on in the leader, neither bit is turned on in middle packets, and EOM is turned on in the last packet. The IMP discards frames which violate these rules. If violations occur.an 1822 error in data message will be returned to the host.

In message mode, both SOM and EOM are turned on in all frames, and the leader and data portions of the message are included continuously in the same frame. The IMP discards frames which violate this rule. If violations occur, an 1822 error in data message will be returned to the host.

The reflect mode (REF) bit indicates that the IMP expects the host to return everything it receives UNCHANGED. This is an important diagnostic tool which aids the network operations center in debugging problems on HDH links. Only the IMP is allowed to set this bit. If the host implementor requires a loopback to the host, this can be implemented by sending HDH packets with 1822 leaders addressed to the the sending host. The IMP will simply route these packets right back to the host.

The octet count field indicates the number of valid octets in the remainder of the frame. This field must always be filled in, and may indicate fewer or the same number of octets as the physical frame contains. This is to provide for host word sizes that may not always align with the desired packet size.

In packet mode, the first LAPB frame contains only the 1822 leader. Therefore, the octet count of the first frame must be either 10, for non-type 0 messages, or 12, for type 0 messages. The octet count for middle packets can be any even number from 2 to 126, and there may be up to seven middle packets, none of which needs to be maximum length. The octet count for last packets can be any number from 1 to 125. In message mode, the octet count must be at least 10 (for non-type 0 messages), and may be any number up to and including 1019. An octet count larger than the amount in the physical frame or a count which violates the above maximum HDH frame length rules is considered a protocol violation and the IMP will discard the illegal frame and an 1822 error in data message will be returned to the host.

4.1.2 HDH Control Messages

HDH control frames, shown in Figures J-4 and J-5, are distinguished by having the high-order bit of the HDH header set to 1, and are used for measuring the quality of the packet level and determining its up/down status. HDH control frames contain no additional octets beyond the HDH header.In the following discussion. control frames sent by the IMP are called "Hello"s and control frames sent by the host are called "I-heard-you"s. The following is a description of the fields in an HDH control frame.

Bit 15 - Control bit

The high order bit of the HDH header distinguishes HDH control frames (bit set to 1) from data frames.

Bit 14 - Sequence Break (SEQ) bit

This bit, when set, indicates a sequence break in the frame stream between the previous frame and the current one.

Bit 13 - Host/IMP bit

This bit indicates if the frame was host or IMP originated. A zero indicates a host originated frame and a one indicates an IMP originated frame.

Bit 12 - Line Status (LIN) bit

This bit, when set, indicates the HDH level between the host and IMP is up (see description below).

Bit 11 - Link Quality Poor (LQP) bit

When this bit is set to one, it indicates that the IMP has determined that the quality of the LAPB link level is below the "acceptable quality" threshold configured for this host link. This bit may be set by the IMP before the link is so bad that the link cannot transfer data at all (see description below).

Bit 10 - Reflect Mode (REF) bit

This bit indicates that the current frame sent by the IMP to the host must be looped back to the IMP. Only the IMP is allowed to set this bit under the current HDH protocol specification.

Bits 9-0 - Line Down Counter

This field contains the control message timer value for the IMP/Host line. The host should declare the HDH level down if it has not received a control message ("hello") from the IMP before the indicated time period lapses. The timer is expressed in units of one second and can have reset values ranging from 3 to 1023 (see description below).

In addition to the previously described sequence break (SEQ), reflect mode (REF), and host/IMP bits, the HDH header contains a link quality poor (LQP) bit, a HDH level up/down bit, and a line down timer parameter value, all of which are described below. The control frames are used by the algorithm described in 4.1.2.1 to decide when the connection between the host and the IMP is dead or alive.

The link quality poor (LQP) bit,when set to one, indicates that the IMP has determined that the quality of the LAPB link level is below the "acceptable quality" threshold configured for this host link. This bit may be set by the IMP before the link is so bad that the HDH level is declared down by the IMP. Only the IMP sets LQP. For singly homed hosts, LQP being set should cause a message to be sent to the operator so that the line situation is identified as a possible reason for poor throughput. For dually homed hosts, LQP being set should cause the host to try its other HDH line. The host should avoid switching back and forth if the alternate HDH line has the LQP bit set as well.

4.1.2.1 HDH Level Up/Down Algorithm

HDH requires that that a Master/Slave relationship exist between the IMP and the host so that the HDH level up/down protocol can work correctly. Periodically, the IMP sends a control frame, called a "hello" message, to the host. Since the frame is sent by the IMP, the host/IMP bit is one. The host should respond immediately to this frame by changing the host/IMP bit to zero, updating the LIN bit and returning the frame to the IMP. The host's response is called an "I-heard-you". The host cannot originate control frames; it can only respond to "hello"s with "I-heard-you"s. Only the IMP can set the LIN bit and declare the HDH level up. The LIN bit is the "virtual IMP ready line" that HDH provides to replace the Host/IMP hardware ready line. The LIN bit the IMP sends is the logical AND of the state of the IMP's ready line and the state of the HDH link to the host. The LIN bit the host sends is NOT the equivilent of a "virtual host ready line" because a host cannot set the LIN bit in an "I-heard-you" to one if the IMP has not set it to one in the last "Hello" received. The host may take the HDH level down at any time by setting the LIN bit in a "I-heard-you" to zero. Control frames may be freely intermixed with data packet frames, including between packets of the same message.

The IMP expects to receive the I-heard-you for its hello frame before it goes to send the next hello frame or within 2 seconds, whichever is sooner, and if it does not, it records a miss. If more than KH misses occur within NH tries, the IMP declares the HDH connection down. This condition is called HDH level down. No data is sent on the HDH connection while the HDH level is down.

The IMP also performs a link quality test for LAPB hosts. A count of the number of level 2 frames that were retransmitted is divided by the total number of level 2 frames sent. If this number is lower than a configurable parameter called the "link useable threshold", the HDH level is declared down. Thus HDH provides tests of both the packet level and link level quality on a periodic basis to provide the maximum amount of information about the host connection.

While the HDH level is down, the IMP continues to send hello control frames and the host responds to those it receives. The IMP will declare the HDH level up if it receives MH responses in a row with no intervening misses AND the level 2 quality is above the "link useable threshold". The host must obey the declarations of the IMP, taking the HDH level down or up as instructed by the setting of the HDH level up/down (LIN) bit in the hello frame. In particular, the host should not send data on the line while hello frames indicate that the HDH level is down.

While the HDH level is up the host should also maintain a "control message timer", which is set to the value conveyed in the most recent "line down count" every time the host receives a hello control frame from the IMP. If the timer ever expires, the host should consider the HDH level down. In this way the host should detect that the HDH level is down if the line has broken so completely that control frames are not received at all from the IMP. Since the value of the control message timer depends on the line speed, and since the host may not be aware of the line speed, the reset value for the timer is communicated to the host in the control frames sent by the IMP. The timer is expressed in units of one second and can have reset values ranging from 3 to 1023.

The host may take the HDH level down (turn off the LIN bit in the I-heard-you) for any reason.

Typical values for the IMP's parameters in the packet level protocol are KH=4, NH=20, and MH=10. The interval for sending hello control frames will vary depending on the speed of the line. Typically, it is between 3.2 and 6.4 seconds. Hence, it may take as long as a minute for an HDH host to come up.

5 HDH Host Finite State Machine

The HDH protocol has two states in message mode and three states in packet mode. The message mode states are the HDH Level Up State and the HDH Level Down State. The packet mode state machine has two HDH level up states, the HDH Level Up Start of Message State and the HDH Level Up Middle of Message State, in addition to the HDH Level Down State. The following is a summary of possible actions taken by the host in response to HDH inputs from the IMP in the various states.

5.1 HDH Message Mode

5.1.1 HDH Level Down State

Input from IMP: Control message. The host responds with a control message. Additional action taken is based on the incoming control message header bits as follows:

Sequence break bit - Flush queues if LIN bit is set to one.

Host/IMP bit off - Error. message is discarded.
Host/IMP bit on - Correct, no action needed.

LIN bit off - Stay in the HDH Level Down State.
LIN bit on - Go to HDH Level Up State.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP. No other control frame is sent by the host.

Line Down Count Field - Set the control message timer to the value in this field.

Input from IMP: A data message. The host's response is based on the incoming data message HDH header bits as follows:

Sequence Break bit - Ignored.

Host/IMP bit off - Error. message is discarded.
Host/IMP bit on - Correct, no action needed.

SOM and EOM on - Ignored in the HDH Level Down State.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged. No action is needed.

5.1.2 HDH Level Up State

Input from IMP: Control message. The host responds with a control message. Additional action is based on the incoming control message HDH header bits as follows:

Sequence Break bit - Ignored.

Host/IMP bit off - Error. message is discarded.
Host/IMP bit on - Correct, no action needed.

LIN bit off - Go to the HDH Level Down State and stop processing incoming data messages.

LIN bit on - No action needed.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged. No other control frame is sent by the host.

Line Down Count Field - Set the control message timer to the value in this field.

Input from IMP: Data message. Action taken is based on the incoming data message HDH header bits as follows:

Sequence Break bit - Ignored.

Host/IMP bit on - Error. message is discarded.
Host/IMP bit on - Correct, no action needed.

SOM or EOM off - Error. message is discarded.

SOM and EOM on - Correct, process data.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged. No further action needed.

5.2 HDH Packet Mode

5.2.1 HDH Level Down State

Input from IMP: Control message. The host responds with a control message. Additional action is based on the control bits in the incoming control message as follows:

Sequence Break bit - Flush queues if LIN is set in this control packet. If LIN not set, ignore.

Host/IMP bit off - Error, message is discarded.
Host/IMP bit on - Correct, no action needed.

LIN bit off - Stay in the HDH Level Down State.
LIN bit on - Go to HDH Level Up Start of Message State.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged in addition to any other processing which may be needed. No other control frame is sent by the host.

Line Down Count Field - Set the control message timer to the value in this field.

Input from IMP: A data message. The host response is based on the HDH header bits of the incoming data message as follows:

Sequence Break bit - Ignored in the HDH Level Down State.

Host/IMP bit off - Error, message is discarded.
Host/IMP bit on - Correct, no action needed.

SOM and EOM - Ignored in the HDH Level Down State.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged. No further action needed.

5.2.2 HDH Level Up Start of Message State

Input from IMP: Control message. The host responds with a control message. Additional action taken is based on the incoming control message HDH Header bits as follows:

Sequence Break - Ignored since we are in the HDH Level up Start of Message State and the next data frame must be a Start of Message frame.

Host/IMP bit on - Error. message is discarded.

LIN bit off - Go to HDH Level Down State and stop processing incoming data messages.
LIN bit on - No action is taken.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged. No other control frame is sent by the host.

Line Down Count Field - Set the control message timer the value in this field.

Input from IMP: A data message. The host response is based on the HDH header bits of the incoming data message as follows:

Sequence Break bit - Ignored in HDH Level Up Start of Message State the next data frame should be the start of a new message.

Host/IMP bit off - Error. message is discarded.
Host/IMP bit on - Correct, no action needed.

SOM bit on, EOM bit off - Accept frame for processing and go to HDH Level Up Middle of Message State.

SOM bit off - Error. discard incoming frame.

SOM and EOM bit on - This is a single frame message. Process message and stay in HDH Level Up Start of Message State.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged in addition to any further processing.

5.2.3 HDH Level Up Middle of Message State

Input from IMP: Control message. The host responds with a control message. Additional action taken is based on the incoming control message HDH header bits as follows:

Sequence Break (SEQ) bit on - Discard all previous frames in this message. go to HDH Level Up Start of Message State.

Host/IMP bit on - Correct, no action needed.
Host/IMP bit off - Error. message is discarded.

LIN bit off - Go to HDH Level Down state. stop processing incoming data messages, and discard all previous frames.

LIN bit on - No action is taken.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged. No other control frame is sent by host.

Line Down Count Field - Set the control message timer to the value in this field.

Input from IMP: A data message. The host response is based on the HDH header bits of the incoming data message as follows:

Sequence Break (SEQ) bit on - Discard all previous frames in this message, go to HDH Level Up Start of Message State. If this frame has the SOM bit set. treat it as the first frame and go to HDH Level Up Middle of Message State.

Host/IMP bit off - Error. message is discarded.
Host/IMP bit on - Correct, no action needed.

SOM on - Error. discard all previous frames in this message, treat the new frame as a start of message, and stay in the HDH Level Up Middle of Message State.

EOM on - The message is now complete, process it and go to the HDH Level Up Start of Message State.

SOM and EOM bit off - This is a valid packet mode middle packet. process it and wait for next data packet.

Reflect bit off - No action needed.
Reflect bit on - This frame must be looped back to the IMP unchanged. No further action needed.

APPENDIX J.1: Alignment to Recommendation X.25

This appendix contains notes which help to clarify the functional link level characteristics of the C/30 HDH DCE. It is organized by reference to the relevant section of the CCITT Recommendation X.25, "Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks," International Telegraph and Telephone Consultative Committee YELLOW BOOK, Vol. VIII.2, Geneva, 1981. Numbers in square brackets indicate sections of that recommendation. For the purposes of the ensuing discussion, references to the DCE refer to that part of the IMP which interfaces with the host. References to the DTE refer to that part of the host entity which interfaces with the IMP.

1 The Link Level DTE/DCE Interface

1.1 Scope and Field of Application [2.1]

Although the C/30 IMP supports both LAP and LAPB classes of procedures, DTE manufacturers and implementors are strongly urged to implement the LAPB procedure, as this is the only service available in all networks supporting the X.25 Recommendation. In C/30 networks, use of LAPB provides for superior error recovery and link monitoring procedures by the DCE.

1.2 Frame Structure [2.2]

The address and control fields consist of one octet each. C/30 IMPs do not support the extension of either the control or the address fields. The frame is required by the implementation to consist of an integral number of octets. The C/30 IMP maintains interframe time fill between frames; detection of idle channel state by the DTE would imply that the DCE is not servicing the link.

1.3 Elements of Procedure [2.3]

1.3.1 Control Field Parameters - Modulus [2.3.2.3]

I frames are numbered sequentially modulo 8. Extended numbering (modulo 128) is not supported.

1.3.2 Rejection Condition [2.3.5.6]

After entering rejection condition by the transmission of an FRMR or CMDR frame, the DCE will retransmit the frame every T1 seconds. After N2 retries without a reset by the DTE, the DCE will reset the link by sending a SABM if it is operating according to LAPB procedures. After N2 tries without a link reset by the DTE, the DCE operating according to LAP procedures will send a DM response and enter the disconnected phase.

1.4 Procedures to Set the Mode Variable B [2.4.1]

The C/30 maintains the internal mode variable B, as it implements both LAP and LAPB classes of procedures.

When in the disconnected phase and in automatic transmit mode, the link level software will, after timer T1 expires, transmit a DM frame to the DTE to solicit a mode-setting command and restart timer T1. If the DTE transmits a SABM frame, the DCE will proceed according to LAPB procedures. If the DTE transmits a SARM frame, the DCE will proceed according to LAP procedures.

1.5 Procedure for the Use of the POLL/FINAL Bit [2.4.3]

The POLL/FINAL bit is used by the DCE in conjunction with the timer recovery condition.

The DCE starts timer T1 whenever it schedules for transmission a frame which requires a response from the DTE, and it is not already running. When timer T1 runs out before the needed response is received from the DTE, the DCE (re)enters the timer recovery condition and transmits an appropriate command with the POLL bit set to 1. The timer recovery condition is cleared when the DCE receives from the DTE a valid unnumbered or supervisory response with the FINAL bit set to 1.

The DCE also uses the T1 timer in connection with automatic transmit mode in the disconnected phase. In this mode it will keep the T1 timer running and transmit a DM response to request a mode-setting command from the DTE each time it expires.

When the link is in the information transfer phase, the DCE operating according to LAPB procedures will ensure in the absence of I frames that the DTE link level procedures are operable by periodic transmission of an S frame with the POLL bit set to solicit a response from the DTE with the FINAL bit set. Failure of the DTE to respond to this command will initiate timer recovery procedures by the DCE. The DTE need make no special allowance for this procedure, since its procedures in responding to the S frame command are simply those it must employ according to Recommendation X.25.

1.6 Procedures for Link Set-up and Disconnection (LAP) [2.4.4]

1.6.1 Link Set-up [2.4.4.1]

If the DCE attempting to set-up or reset the link transmits a SARM command N2 times, its recovery action is to attempt to disconnect the link.

1.6.2 Link Disconnection [2.4.4.3]

If the DCE transmits a DISC command N2 times, its recovery action is to send a DM response and enter the disconnected phase.

1.7 Procedures for Link Setup and Disconnection (LAPB) [2.4.5]

1.7.1 Link Set-up [2.4.5.1]

If the DCE attempting to set-up or reset the link transmits a SABM command N2 times, its recovery action is to attempt to disconnect the link.

1.7.2 Link Disconnection [2.4.5.3]

If the DCE transmits a DISC command N2 times, its recovery action is to send a DM response and enter the disconnected phase.

1.7.3 Disconnected Phase [2.4.5.4]

The C/30 will not initiate link set-up while in the disconnected phase. However, if it is in automatic transmit mode, it will indicate a request for a link set-up command by transmitting a DM response every T1 seconds.

The C/30 in disconnected phase will react only to the frames listed below:

- o SABM will initiate a link set-up, and selects LAPB procedures.
- o SARM will initiate a link set-up, and selects LAP procedures.
- o FRMR received when the B variable is 1 (i.e., the last link set-up was by a SABM command) will be answered with a DM.
- o CMDR received when the B variable is 0 will cause the DCE to initiate a link disconnection by sending a DISC command.
- o DISC received when the B variable is 1 will be answered with a DM response.
- o DISC received when the B variable is 0 will be answered with a UA response.
- o DM will cause the DCE to initiate a link set-up by sending a SABM if the B variable is 1, or a SARM if the B variable is 0.
- o Any other command received with the poll bit set will be answered with a DM response with the final bit set.

All other frames will be ignored.

1.8 Receiving an I Frame [2.4.6.2]

DCE behavior in regard to acknowledgment of I frames can be summarized as follows:

When an I frame is correctly received from the DTE, the DCE starts a timer associated with parameter T2, if it is not already running.

If K2 I frames have been received and not acknowledged, or if no I or S frames are available for transmission before the timer expires, the DCE will transmit an S frame response to carry the value of V(R).

The DCE will stop the timer whenever it sends out a frame containing an N(R) (i.e., an S or I frame).

DCE busy condition may cause the I field of incoming I frames to be ignored; please refer to the following section for details. I frames containing zero length information fields are indicated to the packet level in the received frame. The DCE ignores frames containing a non-integral number of octets.

1.9 DCE Busy Condition [2.4.6.7]

The DCE has two levels of the busy condition. In the first level, called slow mode, any received I frames are accepted by transmission of an RNR response. In the second level, called stop mode, the DCE will discard any I frames it receives from the DTE until the DCE leaves the busy condition, and will continue to reply with RNR responses. The decisions to enter and leave the two modes of DCE busy condition are controlled by three parameters which describe the length of the queue of packets to level 3 in the DCE. Values for these parameters are given in Section J-1.15 below.

To clear the busy condition, the DCE will send a REJ or RR supervisory frame, depending upon whether or not it knowingly discarded any I frames.

1.10 Waiting Acknowledgment [2.4.6.8]

When the T1 timer runs out before a transmitted I frame is acknowledged by the DTE, the DCE will restart timer T1, and do one of the following: if the DCE is operating according to the LAPB procedures, the DCE will transmit an appropriate supervisory command (RR or RNR) with the POLL bit set to 1; if the DCE is operating according to the LAP procedures, it will set its send state variable to the last N(R) received from the DTE, and retransmit the corresponding I frame with the POLL bit set to 1.

Upon N2 retransmissions of the S or I frame command without an appropriate response, the DCE will reset the link by transmission of a SABM (in LAPB procedures) or SARM (LAP) command.

1.11 Procedures for Resetting (Applicable to LAP) [2.4.7]

After transmission of a SARM command N2 times, the C/30 DCE will attempt to disconnect the link.

The DCE does start the T1 timer whenever it sends a CMDR response, and will retransmit the CMDR response each time it expires. After N2 retransmissions of the CMDR response, the DCE will send a DM response and enter disconnected phase.

1.12 Rejection Conditions (Applicable to LAP) [2.4.8]

The DCE will initiate a resetting procedure upon reception of any spurious response frame or any response frame with a spurious final bit. Upon receipt of a CMDR frame. the DCE will reset the link with a SARM command.

1.13 Procedures for Resetting (Applicable to LAPB) [2.4.9]

The DCE does start the T1 timer whenever it sends a FRMR response, and will retransmit the FRMR response each time it expires. After N2 retransmissions of the FRMR response, it will attempt to reset the link using a SABM command.

1.14 Rejection Conditions (Applicable to LAPB) [2.4.10]

1.14.1 Receiving DM or FRMR in Information Transfer Phase [2.4.10.2]

If the DCE receives a DM response during the information transfer phase, it will transmit a DM and enter the disconnected phase. If the DCE receives a FRMR response during the information transfer phase, it will initiate a resetting procedure. If it receives a spurious UA response during the information transfer phase, the DCE will reset the link by sending a SABM command. Any unexpected frame, or frame with an unexpected final bit, will cause the DCE to enter frame rejection condition and transmit a FRMR response.

1.15 Level 2 Implementation-Specific System Parameters

The DCE has three parameters associated with the Busy Condition. The first, whose default value is four, indicates the number of I frames that may be queued awaiting packet level processing before the DCE enters the busy condition slow mode, in which received frames are acknowledged with an RNR response. When the number of items queued for Level 3 reaches the second parameter, whose default value is six, the DCE enters the busy condition stop mode, in which received I frames are discarded. When the number of items queued for Level 3 while in the busy condition falls to the third parameter, whose default value is one, the DCE leaves the busy condition. The Administration may configure each of these parameters for each C/30 DCE to any value between one and seven.

When receiving I frames, the DCE never allows more than K2 frames to be received without sending a frame that carries N(R) (that is, an I or S frame). This is an attempt to keep the DTE's transmit window from closing. K2 must always be equal to or less than K, and should be reduced for lines that have long delay, such as satellite lines. The Administration may configure K2 independently for each C/30 DCE to any value between one and seven; the default value is four.

APPENDIX J.2: C/30 HDH Synchronous Physical Level Specification

1 Introduction

This section describes the functional, electrical, and mechanical connection (the level 1 connection) that is required when an HDH host is connected to a BBNCC C/30 HDH IMP. Such hosts require a synchronous modem connection or the equivalent, which typically will be supplied by the Administration. The host will present the DTE interface while the supplied modem equipment will present the DCE interface. For the purposes of the ensuing discussion, references to the DCE refer to that part of the IMP which interfaces with the host. References to the DTE refer to that part of the host entity which interfaces with the IMP.

2 Supported Interfaces

C/30 HDH IMPs presently support four synchronous level 1 interfaces as shown below:

1. EIA RS-232-C, CCITT V.28 & V.24, X.21 bis;
2. EIA RS-449 & 422, CCITT V.11, MIL-188-114 balanced, Fed. Std. 1031/1020;
3. EIA RS-449 & 423. CCITT V.10, MIL-188-114 unbalanced, Fed. Std. 1031/1030; and
4. CCITT V.35.

MIL-188-114 balanced is deemed equivalent to RS-449 with RS-422, the difference being that MIL-188-114 is more tolerant of noise on signal common and more tolerant of common mode noise. MIL-188-114 unbalanced is deemed equivalent to RS-449 with RS-423. In most cases where MIL-188-114 balanced is specified, MIL-188-114 unbalanced is also available, but it is not recommended.

Table J.2-1 is a dictionary of terms that relates the CCITT signal ID to the EIA signal ID and to the more common abbreviations. Table J.2-2 identifies signals as either required, optional, or not used.

Figure J.2-1 and Table J.2-2 identify typical DTE connections to a C/30 HDH DCE. The required host services will dictate which scheme is selected for a particular DTE.

Table J.2-3 relates required speed of service to interface type. Tables J.2-4 through J.2-6 list signal and pin names for the four supported physical interfaces.

Together, these tables and figures serve as a guide to level 1 interface selection. From these, most systems will be able to identify the most appropriate interface. However, this information is not all-inclusive, and other interface arrangements may be possible. Various considerations may lead an Administration to limit the choice of physical interfaces available to host DTEs.

```
                  Demarcation Point
                 (mating connectors)

                     DTE   DCE

 |-----------|
 |           |----] [------(1) Modem     RS-232-C, RS-449, V.35
 |           |
 |           |----] [------(2) LDM       RS-232-C, RS-449
 |   HOST    |
 |           |----] [------(3) Null Modem Cable
 |           |
 |           |----] [------(4) SME       Cable plus clock source;
 |-----------|                           RS-232-C, RS-449
```

Figure J.2-1 Typical Level 1 Connection Schemes

EIA ID	CCITT ID	ABBREV NAME	NAME
AA	101	FG	Frame (Chassis/Protective) Ground
AB	102	SG	Signal/Supply Common
SC	102a	--	RS-449 DTE Common
RC	102b	--	RS-449 DCE Common
BA	103	TD	Transmit Data
BB	104	RD	Receive Data
CA	105	RTS	Request to Send
CB	106	CTS	Clear to Send
CC	107	DSR	Data Set Ready
CD	108.2	DTR	Data Terminal Ready
CF	109	DCD	Data Carrier Detect
CG	110	SQ	Signal Quality
CH	111	--	Signal Rate Selector to DCE
CI	112	--	Signal Rate Selector to DTE
DA	113	ETC	External Transmit Clock
DB	114	TC	Transmit Clock
DD	115	RC	Receive Clock
--	116	--	Select Standby
--	117	--	Standby Indicator
SBA	118	STD	Secondary Transmit Data
SBB	119	SRD	Secondary Receive Data
SCA	120	SRS	Secondary Request to Send
SCB	121	SCS	Secondary Clear to Send
SCF	122	SCD	Secondary Carrier Detect
SCG	123	SSQ	Secondary Signal Quality
--	124	--	Select Frequency Group
CE	125	RI	Ringing Indicator
--	126	--	Select Transmit Frequency
--	127	--	Select Receive Frequency
--	128	--	External Receive Clock
--	129	RR	Request to Receive
--	130	--	Secondary Transmit Tone
--	131	--	Receive Character Timing
--	132	--	Return to Non-Data Mode
--	133	RTR	Ready to Receive
--	134	--	Received Data Present
--	136	--	New Signal
--	140	RL	Remote Loopback
--	141	LL	Local Loopback
--	142	TM	Test Status Monitor
--	191	--	Transmit Voice Answer
--	192	--	Receive Voice Answer

Table J.2-1 EIA and CCITT Interchange Circuits

Scheme (From Fig. B-1)	Explanation
(1) Modem	Any of the four supported physical interfaces providing service over long haul leased voice grade or group grade telephone facilities.
(2) Limited Distance Modem	LDM generally available at 9600 b/s and below in an RS-232-C version. Other types are available for all speeds.
(3) Null Modem	A Null Modem is a length of cable with the signal leads wired so as to present a DCE interface. To be used in local connection schemes where either the host DTE or the C/30 IMP has a clocking source capability. All four supported level 1 interfaces are available. If host DTE clock and C/30 IMP clock are both available, DTE clock will be preferred.
(4) Synchronous Modem Eliminator	SME is a length of cable with a hardware device interjected. The device allows rewiring of signals so as to present a DCE interface. The device also provides clocking when neither the host DTE nor the C/30 IMP has such capability. All four supported level 1 interfaces are available.

Table J.2-2 Typical Level 1 Connection Schemes

Physical Interface	Signalling Rate in Kb/s 1.2 to 9.6	14.4, 19.2	48 and above
RS-232-C (and equiv.)	A	A	-
RS-449/422 balanced (and equiv.)	A	A	A
RS-449/423 unbalanced (and equiv.)	A	-	-
CCITT V.35	-	-	A

Legend

A = Available
- = Not available

Table J-2.3 Interface Type by Service Speed

Signal Name	Abbrev	Pin No.	EIA ID	Signal Source
Frame Ground	FG	1	AA	DTE/DCE
Transmitted Data	TD	2	BA	DTE
Received Data	RD	3	BB	DCE
Request to Send	RTS	4	CA	DTE
Clear to Send	CTS	5	CB	DCE
Data Set Ready	DSR	6	CC	DCE
Signal Ground	DTE			
Ext. Transmit Clock	ETC	24	DA	DTE
Wired Spare	--	18	--	---
Wired Spare	--	22	--	---
Wired Spare	--	25	--	---

Required pins: 1, 2, 3, 4, 5, 6, 7, 8, 15, 17, 20, 24
Optional pins: 9, 10, 18, 22, 25

Notes:

1. The DTE will present a CANNON DB-25P male connector with pinouts as above or equivalent hardware with identical pinouts.

2. The DCE will present a CANNON DB-25S female connector or equivalent.

Table J.2-4 RS-232-C Interface

Signal Name	Abbrev	Pin Nos.	EIA ID	Signal Source
Send Data	SD	4,22	BA	DTE
Send Timing	ST	5,23	DB	DCE
Receive Data	RD	6,24	BB	DCE
Request to Send	RTS	7,25	CA	DTE
Receive Timing	RT	8,26	DD	DCE
Clear to Send	CTS	9,27	CB	DCE
Local Loopback	LL	10	--	DTE
Data Mode	DM	11,29	CC	DCE
Terminal Ready	TR	12,30	CD	DTE
Receiver Ready	RR	13,31	CF	DCE
Remote Loopback	RL	14	--	DTE
Terminal Timing	TT	17,35	DA	DTE
Test Mode	TM	18	--	DCE
Signal Ground	SG	19	AB	DTE/DCE
Receive Common	RC	20	RC	DCE
Send Common	SC	37	SC	DTE
Wired Spare	--	1	--	---
Wired Spare	--	3,21	--	---

Required pins: 4,22; 5,23; 6,24; 7,25; 8,26; 9,27;
11,29; 12,30; 13,31; 17,35; 19; 20; 37
Optional pins: 10; 14; 18; 1; 3,21

Notes:

1. The DTE will present a CANNON DC-37P male connector with pinouts as above or equivalent hardware with identical pinout.

2. The DCE will present a CANNON DC-37S female connector or equivalent.

Table J.2-5 RS-449 Interface (and equivalents)

Signal Name	Abbrev	Pin Nos.	EIA ID	Signal Source
Frame Ground	FG	A	AA	DTE/DCE
Signal Ground	SG	B	AB	DTE/DCE
Transmit Data	TD	P/S	BA	DTE
Receive Data	RD	R/T	BB	DCE
Request to Send	RTS	C	CA	DTE
Clear to Send	CTS	D	CB	DCE
Data Set Ready	DSR	E	CC	DCE
Data Carrier Detect	DCD	F	CF	DCE
Local Loopback	LL	K	--	DTE
Ext. Transmit Clock	ETC	U/W	DA	DTE
Transmit Clock	TC	Y/aa	DB	DCE
Receive Clock	RC	V/X	DD	DCE

Required Pins:	A; B; P/S; R/T; C; D; E; F; U/W; Y/aa; V/X
Optional Pins:	K

<u>Notes</u>:

1. The DTE will present a Winchester MRA(C)-34D-JTCH-H8 male connector with pinout as above or equivalent hardware with the identical pinout.

2. The DCE will present a mating female connector.

Table J.2-6 V.35 Interface

Report No. 1822 BBN Communications Corporation

Appendix K: CCITT Recommendation X.25 1980 Link Access Procedure (LAPB)

12/83

12/83

2 Link access procedure across the DTE/DCE interface

2.1 *Scope and field of application*

2.1.1 The Link Access Procedures (LAP and LAPB) are described as the Link Level Element and are used for data interchange between a DCE and a DTE operating in user classes of service 8 to 11 as indicated in Recommendation X.1 [1].

2.1.2 The procedures use the principles and terminology of the High Level Data Link Control (HDLC) procedures specified by the International Organization for Standardization (ISO).

2.1.3 The transmission facility is duplex.

2.1.4 DCE compatibility of operation with the ISO balanced class of procedure (Class BA, options 2, 8) is achieved using the provisions found under the headings annotated as "applicable to LAPB" in this Recommendation.

DTE manufacturers and implementors must be aware that the procedure hereunder described as LAPB will be the only one available in all networks.

Likewise, a DTE may continue to use the provisions found under the headings annotated as "applicable to LAP" in this Recommendation (in those networks supporting such a procedure), but for new DTE implementations, LAPB should be preferred.

Note – Other possible applications for further study are, for example:

- two-way alternate, asynchronous response mode;
- two-way simultaneous, normal response mode;
- two-way alternate, normal response mode.

2.2 *Frame structure*

2.2.1 All transmissions are in frames conforming to one of the formats of Table 1/X.25. The flag preceding the address field is defined as the opening flag.

2.2.2 *Flag sequence*

All frames shall start and end with the flag sequence consisting of one 0 followed by six contiguous 1s and one 0. A single flag may be used as both the closing flag for one frame and the opening flag for the next frame.

2.2.3 *Address field*

The address field shall consist of one octet. The coding of the address field is described in § 2.4.2 below.

TABLE 1/X.25

Frame formats

Bit order of transmission	12345678	12345678	12345678	16 to 1	12345678
	Flag	Address	Control	FCS	Flag
	F 01111110	A 8-bits	C 8-bits	FCS 16-bits	F 01111110

FCS Frame Checking Sequence

Bit order of transmission	12345678	12345678	12345678		16 to 1	12345678
	Flag	Address	Control	Information	FCS	Flag
	F 01111110	A 8-bits	C 8-bits	I N-bits	FCS 16-bits	F 01111110

FCS Frame Checking Sequence

2.2.4 *Control field*

The control field shall consist of one octet. The content of this field is described in § 2.3.2 below.

Note – The use of the extended control field is a subject for further study.

2.2.5 *Information field*

The information field of a frame is unrestricted with respect to code or grouping of bits except for the packet formats specified in § 6 below.

See §§ 2.3.4.10 and 2.4.11.3 below with regard to the maximum information field length.

2.2.6 *Transparency*

The DTE or DCE, when transmitting, shall examine the frame content between the two flag sequences including the address, control, information and FCS sequences and shall insert a 0 bit after all sequences of 5 contiguous 1 bits (including the last 5 bits of the FCS) to ensure that a flag sequence is not simulated. The DTE or DCE, when receiving, shall examine the frame content and shall discard any 0 bit which directly follows 5 contiguous 1 bits.

2.2.7 *Frame checking sequence (FCS)*

The FCS shall be a 16-bit sequence. It shall be the ones complement of the sum (modulo 2) of:

1) The remainder of $x^k(x^{15} + x^{14} + x^{13} + \ldots + x^2 + x + 1)$ divided (modulo 2) by the generator polynomial $x^{16} + x^{12} + x^5 + 1$, where k is the number of bits in the frame existing between, but not including, the final bit of the opening flag and the first bit of the FCS, excluding bits inserted for transparency, and

2) the remainder after multiplication by x^{16} and then division (modulo 2) by the generator polynominal $x^{16} + x^{12} + x^5 + 1$, of the content of the frame, existing between but not including, the final bit of the opening flag and the first bit of the FCS, excluding bits inserted for transparency.

As a typical implementation, at the transmitter, the initial remainder of the division is preset to all 1s and is then modified by division by the generator polynominal (as described above) on the address, control and information fields; the 1s complement of the resulting remainder is transmitted as the 16-bit FCS sequence.

At the receiver, the initial remainder is preset to all 1s, and the serial incoming protected bits and the FCS, when divided by the generator polynominal, will result in a remainder of 0001110100001111 (x^{15} through x^0, respectively) in the absence of transmission errors.

2.2.8 *Order of bit transmission*

Addresses, commands, responses and sequence numbers shall be transmitted with the low order bit first (for example, the first bit of the sequence number that is transmitted shall have the weight 2^0).

The order of transmitting bits within the information field is not specified under § 2. The FCS shall be transmitted to the line commencing with the coefficient of the highest term.

Note – The low order bit is defined as bit 1, as depicted in Tables 1/X.25 to 4/X.25.

2.2.9 *Invalid frames*

A frame not properly bounded by two flags, or having fewer than 32 bits between flags, is an invalid frame.

2.2.10 *Frame abortion*

Aborting a frame is performed by transmitting at least seven contiguous 1s (with no inserted 0s).

2.2.11 *Interframe time fill*

Interframe time fill is accomplished by transmitting contiguous flags between frames.

2.2.12 *Link channel states*

2.2.12.1 *Active channel state*

A channel is in an active condition when the DTE or DCE is actively transmitting a frame, an abortion sequence or interframe time fill.

2.2.12.2 *Idle channel state*

A channel is defined to be in an idle condition when a contiguous 1s state is detected that persists for at least 15 bit times.

Note 1 – The action to be taken upon detection of the idle channel state is a subject for further study.

Note 2 – A link channel as defined here is the means of transmission for one direction.

2.3 *Elements of procedure*

2.3.1 The elements of procedure are defined in terms of actions that occur on receipt of commands at a DTE or DCE.

The elements of procedure specified below contain a selection of commands and responses relevant to the link and system configuration described in § 2.1 above.

A procedure is derived from these elements of procedure and is described in § 2.4 below. Together §§ 2.2 and 2.3 form the general requirements for the proper management of the access link.

2.3.2 *Control field formats and state variables*

2.3.2.1 *Control field formats*

The control field contains a command or a response, and sequence numbers where applicable.

Three types of control field formats (see Table 2/X.25) are used to perform numbered information transfer (I frames), numbered supervisory functions (S frames) and unnumbered control functions (U frames).

TABLE 2/X.25

Control field formats

Control field bits	1	2	3	4	5	6	7	8
I frame	0	N(S)			P/F	N(R)		
S frame	1	0	S	S	P/F	N(R)		
U frame	1	1	M	M	P/F	M	M	M

N(S)	Transmitter send sequence number (bit 2 = low order bit)
N(R)	Transmitter receive sequence number (bit 6 = low order bit)
S	Supervisory function bit
M	Modifier function bit
P/F	Poll bit when issued as a command, final bit when issued as a response (1 = Poll/Final)

2.3.2.1.1 *Information transfer format – I*

The I format is used to perform an information transfer. The functions of N(S), N(R) and P/F are independent; i.e., each I frame has an N(S), an N(R) which may or may not acknowledge additional I frames received by the DTE or DCE, and a P/F bit.

2.3.2.1.2 *Supervisory format – S*

The S format is used to perform link supervisory control functions such as acknowledge I frames, request retransmission of I frames, and to request a temporary suspension of transmission of I frames.

2.3.2.1.3 *Unnumbered format – U*

The U format is used to provide additional link control functions. This format contains no sequence numbers.

2.3.2.2 *Control field parameters*

The various parameters associated with the control field formats are described below.

2.3.2.3 *Modulus*

Each I frame is sequentially numbered and may have the value 0 through modulus minus 1 (where "modulus" is the modulus of the sequence numbers). The modulus equals 8 and the sequence numbers cycle through the entire range.

2.3.2.4 *Frame variables and sequence numbers*

2.3.2.4.1 *Send state variable V(S)*

The send state variable denotes the sequence number of the next in-sequence I frame to be transmitted. The send state variable can take on the value 0 through modulus minus 1. The value of the send state variable is incremented by 1 with each successive I frame transmission, but at the DCE cannot exceed N(R) of the last received I or S frame by more than the maximum number of outstanding I frames (k). The value of k is defined in § 2.4.11.4 below.

2.3.2.4.2 *Send sequence number N(S)*

Only I frames contain N(S), the send sequence number of transmitted frames. Prior to transmission of an in-sequence I frame, the value of N(S) is set equal to the value of the send state variable.

2.3.2.4.3 *Receive state variable V(R)*

The receive state variable denotes the sequence number of the next in-sequence I frame to be received. The receive state variable can take on the value 0 through modulus minus 1. The value of the receive state variable is incremented by one with the receipt of an error free, in-sequence I frame whose send sequence number N(S) equals the receive state variable.

2.3.2.4.4 *Receive sequence number N(R)*

All I frames and S frames contain N(R), the expected sequence number of the next received I frame. Prior to transmission of a frame of the above types, the value of N(R) is set equal to the current value of the receive state variable. N(R) indicates that the DTE or DCE transmitting the N(R) has received correctly all I frames numbered up to and including N(R) − 1.

2.3.3 *Functions of the poll/final Bit*

The Poll/Final (P/F) bit serves a function in both command frames and reponse frames. In command frames the P/F bit is referred to as the P bit. In response frames it is referred to as the F bit.

The use of the P/F bit is described in § 2.4.3 below.

2.3.4 *Commands and responses*

The following commands and responses will be used by either the DTE or DCE and are represented in Table 3/X.25.

The commands and responses are as follows:

2.3.4.1 *Information (I) command*

The function of the information (I) command is to transfer across a data link sequentially numbered frames containing an information field.

2.3.4.2 *Receive ready (RR) command and response*

The receive ready (RR) supervisory frame is used by the DTE or DCE to:

1) indicate it is ready to receive an I frame;

2) acknowledge previously received I frames numbered up to and including N(R) − 1.

RR may be used to clear a busy condition that was initiated by the transmission of RNR. The RR command with the P bit set to 1 may be used by the DTE or DCE to ask for the status of the DCE or DTE, respectively.

2.3.4.3 *Reject (REJ) command and response*

The reject (REJ) supervisory frame is used by the DTE or DCE to request retransmission of I frames starting with the frame numbered N(R). I frames numbered N(R) − 1 and below are acknowledged. Additional I frames pending initial transmission may be transmitted following the retransmitted I frame(s).

TABLE 3/X.25

Commands and responses

Format	Commands	Responses	Encoding							
			1	2	3	4	5	6	7	8
Information transfer	I (information)		0	N(S)			P	N(R)		
Supervisory	RR (receive ready)	RR (receive ready)	1	0	0	0	P/F	N(R)		
	RNR (receive not ready)	RNR (receive not ready)	1	0	1	0	P/F	N(R)		
	REJ (reject)	REJ (reject)	1	0	0	1	P/F	N(R)		
Unnumbered	SARM (set asynchronous response mode)	DM (disconnected mode)	1	1	1	1	P/F	0	0	0
	SABM (set asynchronous balanced mode)		1	1	1	1	P	1	0	0
	DISC (disconnect)		1	1	0	0	P	0	1	0
		UA (unnumbered acknowledgement)	1	1	0	0	F	1	1	0
		CMDR (command reject) FRMR (frame reject)	1	1	1	0	F	0	0	1

Note 1 – The need for, and use of, additional commands and responses are for further study.

Note 2 – DTEs do not have to implement both SARM and SABM; furthermore DM and SABM need not be used if SARM only is used.

Note 3 – RR, RNR and REJ supervisory command frames are not used by the DCE when SARM is used (LAP).

Only one REJ exception condition for a given direction of information transfer may be established at any time. The REJ exception condition is cleared (reset) upon the receipt of an I frame with an N(S) equal to the N(R) of the REJ.

The REJ command with the P bit set to 1 may be used by the DTE or DCE to ask for the status of the DCE or DTE, respectively.

2.3.4.4 *Receive not ready (RNR) command and response*

The receive not ready (RNR) supervisory frame is used by the DTE or DCE to indicate a busy condition, i.e., temporary inability to accept additional incoming I frames. I frames numbered up to and including N(R) – 1 are acknowledged. I frame N(R) and subsequent I frames received, if any, are not acknowledged; the acceptance status of these I frames will be indicated in subsequent exchanges.

An indication that the busy condition has cleared is communicated by the transmission of a UA, RR, REJ or SABM.

The RNR command with the P bit set to 1 may be used by the DTE or DCE to ask for the status of the DCE or DTE, respectively.

2.3.4.5 *Set asynchronous response mode (SARM) command*

The SARM unnumbered command is used to place the addressed DTE or DCE in the asynchronous response mode (ARM) information transfer phase.

No information field is permitted with the SARM command. A DTE or DCE confirms acceptance of SARM by the transmission at the first opportunity of UA response. Upon acceptance of this command, the DTE or DCE receive state variable V(R) is set to 0.

Previously transmitted I frames that are unacknowledged when this command is actioned remain unacknowledged.

2.3.4.6 *Set asynchronous balanced mode (SABM) command*

The SABM unnumbered command is used to place the addressed DTE or DCE in the asynchronous balanced mode (ABM) information transfer phase.

No information field is permitted with the SABM command. A DTE or DCE confirms acceptance of SABM by the transmission at the first opportunity of a UA response. Upon acceptance of this command the DTE or DCE send state variable V(S) and receive state variable V(R) are set to 0.

Previously transmitted I frames that are unacknowledged when this command is actioned remain unacknowledged.

2.3.4.7 *Disconnect (DISC) command*

The DISC unnumbered command is used to terminate the mode previously set. It is used to inform the DTE or DCE receiving the DISC that the DCE or DTE sending the DISC is suspending operation.

No information field is permitted with the DISC command. Prior to actioning the command, the DTE or DCE receiving the DISC confirms the acceptance of DISC by the transmission of a UA response. The DCE or DTE sending the DISC enters the disconnected phase when it receives the acknowledging UA response.

Previously transmitted I frames that are unacknowledged when this command is actioned remain unacknowledged.

2.3.4.8 *Unnumbered acknowledgement (UA) response*

The UA unnumbered response is used by the DTE or DCE to acknowledge the receipt and acceptance of the U format commands. Received U format commands are not actioned until the UA response is transmitted. The UA response is transmitted as directed by the received U format command. No information field is permitted with the UA response.

2.3.4.9 *Disconnected mode (DM) response*

The DM unnumbered response is used to report a status where the DTE or DCE is logically disconnected from the link, and is in the disconnected phase. The DM response is sent in this phase to request a set mode command, or, if sent in response to the reception of a set mode command, to inform the DTE or DCE that the DCE or DTE, respectively, is still in the disconnected phase and cannot action the set mode command. No information field is permitted with the DM response.

A DTE or DCE in a disconnected phase will monitor received commands, and will react to SABM as outlined in § 2.4.5 below and will respond DM with the F bit set to 1 to any other command received with the P bit set to 1.

2.3.4.10 *Command reject (CMDR) response; Frame reject (FRMR) response*

The CMDR (FRMR) response is used by the DTE or DCE to report an error condition not recoverable by retransmission of the identical frame; i.e., one of the following conditions, which results from the receipt of a frame without FCS error:

1) the receipt of a command or response that is invalid or not implemented;

2) the receipt of an I frame with an information field which exceeds the maximum established length;

3) the receipt of an invalid N(R) (in the case of LAP, see § 2.4.8.1);

4) the receipt of a frame with an information field which is not permitted or the receipt of an S or U frame with incorrect length.

An invalid N(R) is defined as one which points to an I frame which has previously been transmitted and acknowledged or to an I frame which has not been transmitted and is not the next sequential I frame pending transmission.

An information field which immediately follows the control field, and consists of 3 octets, is returned with this response and provides the reason for the CMDR (FRMR) response. This format is given in Table 4/X.25.

TABLE 4/X.25

CMDR (FRMR) information field format

Information field bits

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Rejected frame control field								0	V(S)			(see Note)	V(R)			W	X	Y	Z	0	0	0	0

- Rejected frame control field is the control field of the received frame which caused the command (frame) reject.
- V(S) is the current send state variable value at the DTE or DCE reporting the rejection condition (bit 10 = low order bit).
- V(R) is the current receive state variable value at the DTE or DCE reporting the rejection condition (bit 14 = low order bit).
- W set to 1 indicates that the control field received and returned in bits 1 through 8 was invalid or not implemented.
- X set to 1 indicates that the control field received and returned in bits 1 through 8 was considered invalid because the frame contained an information field which is not permitted or is an S or U frame with incorrect length. Bit W must be set to 1 in conjunction with this bit.
- Y set to 1 indicates that the information field received exceeded the maximum established capacity of the DTE or DCE reporting the rejection condition.
- Z set to 1 indicates that the control field received and returned in bits 1 through 8 contained an invalid N(R).

Note – Bits 9, 13 and 21 to 24 shall be set to 0 for CMDR. For FRMR, bits 9 and 21 to 24 shall be set to 0. Bit 13 shall be set to 1 if the frame rejected was a response, and set to 0 if the frame rejected was a command.

2.3.5 *Exception condition reporting and recovery*

The error recovery procedures which are available to effect recovery following the detection/occurrence of an exception condition at the link level are described below. Exception conditions described are those situations which may occur as the result of transmission errors, DTE or DCE malfunction or operational situations.

2.3.5.1 *Busy condition*

The busy condition results when a DTE or DCE is temporarily unable to continue to receive I frames due to internal constraints, e.g., receive buffering limitations. In this case an RNR frame is transmitted from the busy DTE or DCE. I frames pending transmission may be transmitted from the busy DTE or DCE prior to or following the RNR. Clearing of the busy condition is indicated as described in § 2.3.4.4 above.

2.3.5.2 *N(S) sequence error*

The information field of all I frames whose N(S) does not equal the receive state variable V(R) will be discarded.

An N(S) sequence exception condition occurs in the receiver when an I frame received error-free (no FCS error) contains an N(S) which is not equal to the receive state variable at the receiver. The receiver does not acknowledge (increment its receive state variable) the I frame causing the sequence error, or any I frames which may follow, until an I frame with the correct N(S) is received.

A DTE or DCE which receives one or more I frames having sequence errors but otherwise error-free shall accept the control information contained in the N(R) field and the P bit to perform link control functions; e.g., to receive acknowledgement of previously transmitted I frames and to cause the DTE or DCE to respond (P bit set to 1). Therefore, the retransmitted I frame may contain an N(R) field and P bit that are updated from, and therefore different from, the ones contained in the originally transmitted I frame.

2.3.5.3 *REJ recovery*

The REJ is used to initiate an exception recovery (retransmission) following the detection of a sequence error.

Only one "sent REJ" exception condition from a DTE or DCE is established at a time. A sent REJ exception condition is cleared when the requested I frame is received.

A DTE or DCE receiving REJ initiates sequential (re-)transmission of I frames starting with the I frame indicated by the N(R) obtained in the REJ frame.

2.3.5.4 *Time-out recovery*

If a DTE or DCE, due to a transmission error, does not receive (or receives and discards) a single I frame or the last I frame in a sequence of I frames, it will not detect an out-of-sequence exception condition and therefore will not transmit REJ. The DCE or DTE which transmitted the unacknowledged I frame(s) shall, following the completion of a system specified time-out period (see § 2.4.11.1 below), take appropriate recovery action to determine at which I frame retransmission must begin.

2.3.5.5 *FCS error and invalid frame*

Any frame received with an FCS error or which is invalid (see § 2.2.9 above) will be discarded and no action is taken as the result of that frame.

2.3.5.6 *Rejection condition*

A rejection condition is established upon the receipt of an error free frame which contains an invalid command/response in the control field, an invalid frame format, an invalid N(R) (however see § 2.4.8.1 below for LAP application) or an information field which exceeds the maximum information field length which can be accommodated.

At the DTE or DCE, this exception condition is reported by a CMDR (FRMR) response for appropriate DCE or DTE action, respectively. Once a DCE has established a CMDR (FRMR) exception condition, no additional I frames are accepted until the condition is reset by the DTE, except for examination of the P bit (LAPB) or examination of the P bit and N(R) (LAP). The CMDR (FRMR) response may be repeated at each opportunity until recovery is effected by the DTE, or until the DCE initiates its own recovery.

2.4 *Description of the procedures*

2.4.1 *Procedure to set the mode variable B (applicable if both LAP and LAPB are implemented)*

The DCE will maintain an internal mode variable B, which it will set as follows:

- to 1, upon acceptance of an SABM command from the DTE;
- to 0, upon acceptance of an SARM command from the DTE.

Changes to the mode variable B by the DTE should occur only when the link has been disconnected as described in § 2.4.4.3 or § 2.4.5.3 below.

Should a DCE malfunction occur, the internal mode variable B will be initially set to 1 upon restoration of operation, but prior to link set-up by the DTE.

Whenever B is 1, the DCE will use the LAPB link set-up and disconnection procedures and is said to be in the LAPB (balanced) mode.

Whenever B is 0, the DCE will use the LAP link set-up and disconnection procedures and is said to be in the LAP mode.

The following are applicable to both LAP and LAPB modes: §§ 2.4.2, 2.4.3, 2.4.6, 2.4.11.

The following are applicable only to the LAP mode: §§ 2.4.4, 2.4.7, 2.4.8.

The following are applicable only to the LAPB mode: §§ 2.4.5, 2.4.9, 2.4.10.

2.4.2 *Procedure for addressing (applicable to both LAP and LAPB)*

Frames containing commands transferred from the DCE to the DTE will contain the address A.

Frames containing responses transferred from the DTE to the DCE shall contain the address A.

Frames containing commands transferred from the DTE to the DCE shall contain the address B.

Frames containing responses transferred from the DCE to the DTE will contain the address B.

A and B addresses are coded as follows:

Address	1 2 3 4 5 6 7 8
A	1 1 0 0 0 0 0 0
B	1 0 0 0 0 0 0 0

Note – The DCE will discard all frames received with an address other than A or B: the DTE should do the same.

2.4.3 *Procedure for the use of the P/F bit (applicable to both LAP and LAPB)*

The DTE or DCE receiving a SARM, SABM, DISC, supervisory command or an I frame with the P bit set to 1, will set the F bit to 1 in the next response frame it transmits.

The response frame returned by the DCE to a SARM, SABM or DISC command with the P bit set to 1 will be a UA (or DM) response with the F bit set to 1. The response frame returned by the DCE to an I frame with the P bit set to 1 will be an RR, REJ, RNR, or CMDR (FRMR) response format with the F bit set to 1.

The response frame returned by the DCE to a supervisory command frame with the P bit set to 1 will be an RR, RNR, or CMDR (FRMR) response with the F bit set to 1.

The P bit may be used by the DCE in conjunction with the timer recovery condition (see § 2.4.6.8 below).

Note – Other use of the P bit by the DCE is a subject for further study.

2.4.4 *Procedures for link set-up and disconnection (applicable to LAP)*

2.4.4.1 *Link set-up*

The DCE will indicate that it is able to set up the link by transmitting contiguous flags (active channel state).

The DTE shall indicate a request for setting up the link by transmitting a SARM command to the DCE.

Whenever receiving a SARM command, the DCE will return a UA response to the DTE and set its receive state variable V(R) to 0.

Should the DCE wish to indicate a request for setting up the link, or after transmission of a UA response to a first SARM command from the DTE as a request for setting up the link, the DCE will transmit a SARM command to the DTE and start Timer T1 (see § 2.4.11.1 below). The DTE will confirm the reception of the SARM command by transmitting a UA response. When receiving the UA response the DCE will set its send state variable to 0 and stop its Timer T1.

If Timer T1 runs out before the UA response is received by the DCE, the DCE will retransmit a SARM command and restart Timer T1. After transmission of SARM N2 times by the DCE, appropriate recovery action will be initiated.

The value of N2 is defined in § 2.4.11.2 below.

2.4.4.2 *Information transfer phase*

After having both received a UA response to a SARM command transmitted to the DTE and transmitted a UA response to a SARM command received from the DTE, the DCE will accept and transmit I and S frames according to the procedures described in § 2.4.6 below.

When receiving a SARM command, the DCE will conform to the resetting procedure described in § 2.4.7 below. The DTE may also receive a SARM command according to this resetting procedure.

2.4.4.3 *Link disconnection*

During the information transfer phase the DTE shall indicate a request for disconnecting the link by transmitting a DISC command to the DCE.

Whenever receiving a DISC command, the DCE will return a UA response to the DTE.

During an information transfer phase, should the DCE wish to indicate a request for disconnecting the link, or when receiving from the DTE a first DISC command as a request for disconnecting the link, the DCE will transmit a DISC command to the DTE and start Timer T1 (§ 2.4.11.1 below). The DTE will confirm reception of the DISC command by returning a UA response. After transmitting a SARM command, the DCE will not transmit a DISC command until a UA response is received for this SARM command or until Timer T1 runs out. When receiving a UA response to the DISC command, the DCE will stop its Timer T1.

If Timer T1 runs out before a UA response is received by the DCE, the DCE will retransmit a DISC command and restart Timer T1. After transmission of DISC N2 times by the DCE, appropriate recovery action will be initiated. The value of N2 is defined in § 2.4.11.2 below.

2.4.5 *Procedures for link set-up and disconnection (applicable to LAPB)*

2.4.5.1 *Link set-up*

The DCE will indicate that it is able to set up the link by transmitting contiguous flags (active channel state).

Whenever receiving an SABM command, the DCE will return a UA response to the DTE and set both its send and receive state variables V(S) and V(R) to 0.

Should the DCE wish to set up the link, it will send the SABM command and start Timer T1 (see § 2.4.11.1 below). Upon reception of the UA response from the DTE the DCE resets both its send and receive state variables V(S) and V(R) to 0 ar ' stops its Timer T1.

Should Timer T1 expire before reception of the UA response from the DTE, the DCE will retransmit the SABM command and restart Timer T1. After transmission of the SABM command N2 times by the DCE, appropriate recovery action will be initiated. The value of N2 is defined in § 2.4.11.2 below.

2.4.5.2 *Information transfer phase*

After having transmitted the UA response to an SABM command or having received the UA response to a transmitted SABM command, the DCE will accept and transmit I and S frames according to the procedures described in § 2.4.6 belo v.

When receiving an SABM command while in the information transfer phase, the DCE will conform to the resetting procedure described in § 2.4.9 below.

2.4.5.3 *Link disconnection*

During the information transfer phase, the DTE shall indicate disconnecting of the link by transmitting a DISC command to the DCE.

When receiving a DISC command, the DCE will return a UA response to the DTE and enter the disconnected phase.

Should the DCE wish to disconnect the link, it will send the DISC command and start Timer T1 (see § 2.4.11.1 below). Upon reception of the UA response from the DTE, the DCE will stop its Timer T1.

Should Timer T1 expire before reception of the UA response from the DTE, the DCE will retransmit the DISC command and restart Timer T1. After transmission of the DISC command N2 times by the DCE, appropriate recovery action will be initiated. The value of N2 is defined in § 2.4.11.2 below.

2.4.5.4 *Disconnected phase*

2.4.5.4.1 After having received a DISC command from the DTE and returned a UA response to the DTE, or having received the UA response to a transmitted DISC command, the DCE will enter the disconnected phase.

In the disconnected phase, the DCE may initiate link set-up. In the disconnected phase, the DCE will react to the receipt of an SABM command as described in § 2.4.5.1 above and will transmit a DM response in answer to a received DISC command.

When receiving any other command frame with the P bit set to 1, the DCE will transmit a DM response with the F bit set to 1.

Other frames received in the disconnected phase will be ignored by the DCE.

2.4.5.4.2 When the DCE enters the disconnected phase after detecting error conditions as listed in § 2.4.10 below, or exceptionally after recovery from an internal temporary malfunction; it may also indicate this by sending a DM response rather than a DISC command. In these cases, the DCE will transmit DM and start its Timer T1 (see § 2.4.11.1 below).

If Timer T1 runs out before the reception of an SABM or DISC command from the DTE, the DCE will retransmit the DM response and restart Timer T1. After transmission of the DM response N2 times, the DCE will remain in the disconnected phase and appropriate recovery actions will be initiated. The value of N2 is defined in § 2.4.11.2 below.

2.4.5.5 *Collision of unnumbered commands*

Collision situations shall be resolved in the following way:

2.4.5.5.1 If the sent and received U commands are the same, the DTE and DCE shall send the UA response at the earliest possible opportunity. The DCE shall enter the indicated phase after receiving the UA response.

2.4.5.5.2 If the sent and received U commands are different, the DTE and DCE shall enter the disconnected phase and issue a DM response at the earliest possible opportunity.

2.4.5.6 *Collision of DM response with the SABM or DISC commands*

When a DM response is issued by the DCE as an unsolicited response to request the DTE to issue a mode-setting command as described in § 2.4.5.4.2, a collision between a SABM or DISC command issued by the DTE and the unsolicited DM response issued by the DCE may occur. In order to avoid misinterpretation of the DM received, it is suggested that the DTE always send its SABM or DISC command with the P bit set to 1.

2.4.6 ***Procedures for information transfer (applicable to both LAP and LAPB)***

The procedures which apply to the transmission of I frames in each direction during the information transfer phase are described below.

In the following, "number 1 higher" is in reference to a continuously repeated sequence series, i.e., 7 is 1 higher than 6 and 0 is 1 higher than 7 for modulo 8 series.

2.4.6.1 *Sending I frames*

When the DCE has an I frame to transmit (i.e., an I frame not already transmitted, or having to be retransmitted as described in §§ 2.4.6.5 or 2.4.6.8 below), it will transmit it with an N(S) equal to its current send state variable V(S), and an N(R) equal to its current receive state variable V(R). At the end of the transmission of the I frame, it will increment its send state variable V(S) by 1.

If Timer T1 is not running at the instant of transmission of an I frame, it will be started.

If the send state variable V(S) is equal to the last value of N(R) received plus k (where k is the maximum number of outstanding I frames – see § 2.4.11.4 below), the DCE will not transmit any new I frames, but may retransmit an I frame as described in § 2.4.6.5 or § 2.4.6.8 below.

Note – In order to ensure security of information transfer, the DTE should not transmit any new I frame if its send state variable V(S) is equal to the last value of N(R) it has received from the DCE plus 7.

When the DCE is in the busy condition it may still transmit I frames, provided that the DTE is not busy itself. When the DCE is in the command rejection condition (LAP), it may still transmit I frames. When the DCE is in the frame rejection condition (LAPB), it will stop transmitting I frames.

2.4.6.2 *Receiving an I frame*

2.4.6.2.1 When the DCE is not in a busy condition and receives with the correct FCS an I frame whose send sequence number is equal to the DCE receive state variable V(R), the DCE will accept the information field of this frame, increment by 1 its receive state variable V(R), and act as follows:

i) If an I frame is available for transmission by the DCE, it may act as in § 2.4.6.1 above and acknowledge the received I frame by setting N(R) in the control field of the next transmitted I frame to the value of the DCE receive state variable V(R). Alternatively, the DCE may acknowledge the received I frame by transmitting an RR with the N(R) equal to the value of the DCE receive state variable V(R).

ii) If no I frame is available for transmission by the DCE, it will transmit an RR with the N(R) equal to the value of the DCE receive state variable V(R).

2.4.6.2.2 When the DCE is in a busy condition, it may ignore the information field contained in any received I frame.

Note – Zero length information fields shall not be passed to the packet level and this situation should be indicated to the packet level.

2.4.6.3 *Reception of incorrect frames*

When the DCE receives a frame with an incorrect FCS or receives an invalid frame (see § 2.2.9), this frame will be discarded.

When the DCE receives an I frame whose FCS is correct, but whose send sequence number is incorrect, i.e., not equal to the current DCE receive state variable V(R), it will discard the information field of the frame and transmit an REJ response with the N(R) set to one higher than the N(S) of the last correctly received I frame. The DCE will then discard the information field of all I frames until the expected I frame is correctly received. When receiving the expected I frame, the DCE will then acknowledge the frame as described in § 2.4.6.2 above. The DCE will use the N(R) and P bit indications in the discarded I frames.

2.4.6.4 *Receiving acknowledgement*

When receiving correctly an I or S frame (RR, RNR or REJ), even in the busy or command rejection condition, the DCE will consider the N(R) contained in this frame as an acknowledgement for all the I frames it has transmitted with an N(S) up to and including the received N(R) minus one. The DCE will reset the Timer T1 when it receives correctly an I or S frame with the N(R) higher than the last received N(R) (actually acknowledging some I frames).

If Timer T1 has been reset, and if there are outstanding I frames still unacknowledged, the DCE will restart Timer T1. If Timer T1 then runs out, the DCE will follow the retransmission procedure (in § 2.4.6.5 and § 2.4.6.8 below) with respect to the unacknowledged I frames.

2.4.6.5 *Receiving reject*

When receiving an REJ, the DCE will set its send state variable V(S) to the value of the N(R) received in the REJ control field. It will transmit the corresponding I frame as soon as it is available or retransmit it. (Retransmission will conform to the following:

i) If the DCE is transmitting a supervisory or unnumbered command or response when it receives the REJ, it will complete that transmission before commencing transmission of the requested I frame.

ii) If the DCE is transmitting an I frame when the REJ is received, it may abort the I frame and commence transmission of the requested I frame immediately after abortion.

iii) If the DCE is not transmitting any frame when the REJ is received, it will commence transmission of the requested I frame immediately.

In all cases, if other unacknowledged I frames have already been transmitted following the one indicated in the REJ, then those I frames will be retransmitted by the DCE following the retransmission of the requested I frame.

If the REJ frame was received from the DTE as a command with the P bit set to 1, the DCE will transmit an RR or RNR response with the F bit set to 1 before transmitting or retransmitting the corresponding I frame.

2.4.6.6 *Receiving RNR*

After receiving an RNR, the DCE may transmit or retransmit the I frame with the send sequence number equal to the N(R) indicated in the RNR. If Timer T1 runs out after the reception of RNR, the DCE will follow the procedure described in § 2.4.6.8 below. In any case the DCE will not transmit any other I frames before receiving an RR or REJ.

2.4.6.7 *DCE busy condition*

When the DCE enters a busy condition, it will transmit an RNR response at the earliest opportunity. While in the busy condition, the DCE will accept and process S frames and return an RNR response with the F bit set to 1 if it receives an S or I command frame with the P bit set to 1. To clear the busy condition, the DCE will transmit either an REJ response or an RR response with N(R) set to the current receive state variable V(R) depending on whether or not it discarded information fields of correctly received I frames.

Note – The DTE when encountering a DCE busy condition, may send supervisory command frames with the P bit set to 1. In the event that the DTE has not implemented supervisory commands, it may follow the procedures of the DCE (see § 2.4.6.6) (applicable to LAPB).

2.4.6.8 *Waiting acknowledgement*

The DCE maintains an internal retransmission count variable which is set to 0 when the DCE receives a UA or RNR, or when the DCE receives correctly an I or S frame with the N(R) higher than the last received N(R) (actually acknowledging some outstanding I frames).

If Timer T1 runs out, the DCE will (re-)enter the timer recovery condition, add one to its retransmission count variable and set an internal variable *x* to the current value of its send state variable.

The DCE will restart Timer T1, set its send state variable to the last value of N(R) received from the DTE and retransmit the corresponding I frame with the P bit set to 1 (LAP or LAPB) or transmit an appropriate supervisory command with the P bit set to 1 (LAPB only).

The timer recovery condition is cleared when the DCE receives a valid S frame from the DTE with the F bit set to 1.

If, while in the timer recovery condition, the DCE receives correctly a supervisory frame with the F bit set to 1 and with the N(R) within the range from its current send state variable to *x* included, it will clear the timer recovery condition and set its send state variable to the value of the received N(R).

If, while in the timer recovery condition, the DCE receives correctly a supervisory frame with the F bit set to 0 and with an N(R) within the range from its current send state variable to *x* included, it will not clear the timer recovery condition. The value of the received N(R) may be used to update the send state variable. However, the DCE may decide to keep the last transmitted I frame in store (even if it is acknowledged) in order to be able to retransmit it with the P bit set to 1 when Timer T1 runs out at a later time.

If the retransmission count variable is equal to N2, the DCE initiates a resetting procedure for the direction of transmission from the DCE as described in § 2.4.7.3, § 2.4.9.2 or § 2.4.9.3 below. N2 is a system parameter (see § 2.4.11.2 below).

Note – Although the DCE will implement the internal variable *x*, other mechanisms do exist that achieve the identical functions. Therefore, the internal variable *x* is not necessarily implemented in the DTE.

2.4.9.2 The DTE or DCE shall indicate a resetting by transmitting an SABM command. After receiving an SABM command, the DCE or DTE, respectively, will return, at the earliest opportunity, a UA response to the DTE or DCE, respectively, and reset its send and receive state variables V(S) and V(R) to 0. This also clears a DCE and/or DTE busy condition, if present. Prior to initiating this link resetting procedure, the DTE or DCE may initiate a disconnect procedure as described in § 2.4.5.3 above.

2.4.9.3 Under certain rejection conditions listed in § 2.4.6.8 above and § 2.4.10.2 below, the DCE may ask the DTE to reset the link by transmitting a DM response.

After transmitting a DM response, the DCE will enter the disconnected phase as described in § 2.4.5.4.2 above.

2.4.9.4 Under certain rejection conditions listed in § 2.4.10.1 below, the DCE may ask the DTE to reset the link by transmitting an FRMR response.

After transmitting an FRMR response, the DCE will enter the frame rejection condition. The frame rejection condition is cleared when the DCE receives an SABM or DISC command or DM response. Any other command received while in the frame rejection condition will cause the DCE to retransmit the FRMR response with the same information field as originally transmitted.

The DCE may start Timer T1 on transmission of the FRMR response. If Timer T1 runs out before the reception of an SABM or DISC command from the DTE, the DCE may retransmit the FRMR response and restart Timer T1. After transmission of the FRMR response N2 times the DCE may reset the link as described in § 2.4.9.2 above. The value of N2 is defined in § 2.4.11.2 below.

2.4.10 *Rejection conditions (applicable to LAPB)*

2.4.10.1 The DCE will initiate a resetting procedure as described in § 2.4.9.4 above, when receiving, during the information transfer phase, a frame with the correct FCS, with the address A or B, and with one of the following conditions:

- the frame is unknown as a command or as a response;
- the information field is invalid;
- the N(R) contained in the control field is invalid as described in § 2.4.8.1 above.

The coding of the information field of the FRMR response which is transmitted is given in § 2.3.4.10 above. Bit 13 of this information field is set to 0 if the address of the rejected frame is B. It is set to 1 if the address is A.

2.4.10.2 The DCE will initiate a resetting procedure as described in § 2.4.9.2 or § 2.4.9.3 above when receiving during the information transfer phase a DM response or an FRMR response.

The DCE may initiate a resetting procedure as described in § 2.4.9.2 or § 2.4.9.3 above when receiving during the information transfer phase a UA response or an unsolicited response with the F bit set to 1.

2.4.11 *List of system parameters (applicable to both LAP and LAPB)*

The system parameters are as follows:

2.4.11.1 *Timer T1*

The period of Timer T1 will take into account whether the timer is started at the beginning or the end of the frame in the DCE.

The period of Timer T1, at the end of which retransmission of a frame may be initiated according to the procedures described in §§ 2.4.4 to 2.4.6 above, is a system parameter agreed for a period of time with the Administration.

The proper operation of the procedure requires that Timer T1 be greater than the maximum time between transmission of frames (SARM, SABM, DM, DISC, FRMR, I or supervisory commands) and the reception of the corresponding frame returned as an answer to this frame (UA, DM or acknowledging frame). Therefore, the DTE should not delay the response or acknowledging frame returned to the above frames by more than a value T2 less than T1, where T2 is a system parameter.

The DCE will not delay the response or acknowledging frame returned to a command by more than T2.

2.4.7 *Procedures for resetting (applicable to LAP)*

2.4.7.1 The resetting procedure is used to reinitialize one direction of information transmission, according to the procedure described below. The resetting procedure only applies during the information transfer phase.

2.4.7.2 The DTE will indicate a resetting of the information transmission from the DTE by transmitting an SARM command to the DCE. When receiving an SARM command, the DCE will return, at the earliest opportunity, a UA response to the DTE and set its receive state variable V(R) to 0. This also indicates a clearance of the DCE busy condition, if present.

2.4.7.3 The DCE will indicate a resetting of the information transmission from the DCE by transmitting an SARM command to the DTE and will start Timer T1 (see § 2.4.11.1 below). The DTE will confirm reception of the SARM command by returning a UA response to the DCE. When receiving this UA response to the SARM command, the DCE will set its send state variable to 0 and stop its Timer T1. If Timer T1 runs out before the UA response is received by the DCE, the DCE will retransmit an SARM command and restart Timer T1. After transmission of SARM N2 times, appropriate recovery action will be initiated. The value of N2 is defined in § 2.4.11.2 below.

The DCE will not act on any received response frame which arrives before the UA response to the SARM command. The value of N(R) contained in any correctly received I command frames arriving before the UA response will also be ignored.

2.4.7.4 When receiving a CMDR response from the DTE, the DCE will initiate a resetting of the information transmission from the DCE as described in § 2.4.7.3 above.

2.4.7.5 If the DCE transmits a CMDR response, it enters the command rejection condition. This command rejection condition is cleared when the DCE receives an SARM or DISC command. Any other command received while in the command rejection condition will cause the DCE to retransmit this CMDR response. The coding of the CMDR respone will be as described in § 2.3.4.10 above.

2.4.8 *Rejection conditions (applicable to LAP)*

2.4.8.1 *Rejection conditions causing a resetting of the transmission of information from the DCE*

The DCE will initiate a resetting procedure as described in § 2.4.7.3 above when receiving a frame with the correct FCS, with the address A (coded 11000000) and with one of the following conditions:

- the frame type is unknown as one of the responses used;
- the information field is invalid;
- the N(R) contained in the control field is invalid;
- the response contains an F bit set to 1 except during a timer recovery condition as described in § 2.4.6.8 above.

The DCE will also initiate a resetting procedure as described in § 2.4.7.3 above when receiving an I frame with correct FCS, with the address B (coded 10000000) and with an invalid N(R) contained in the control field.

A valid N(R) must be within the range from the lowest send sequence number N(S) of the still unacknowledged frame(s) to the current DCE send state variable included, even if the DCE is in a rejection condition, but not if the DCE is in the timer recovery condition (see § 2.4.6.8 above).

2.4.8.2 *Rejection conditions causing the DCE to request a resetting of the transmission of information from the DTE*

The DCE will enter the command rejection condition as described in § 2.4.7.5 above when receiving a frame with the correct FCS, with the address B (coded 10000000) and with one of the following conditions:

- the frame type is unknown as one of the commands used;
- the information field is invalid.

2.4.9 *Procedures for resetting (applicable to LAPB)*

2.4.9.1 The resetting procedures are used to initialize both directions of information transmission according to the procedure described below. The resetting procedures only apply during the information transfer phase.

2.4.11.2 *Maximum number of transmissions N2*

The value of the maximum number N2 of transmission and retransmissions of a frame following the running out of Timer T1 is a system parameter agreed for a period of time with the Administration.

2.4.11.3 *Maximum number of bits in an I frame N1*

The maximum number of bits in an I frame is a system parameter which depends upon the maximum length of the information fields transferred across the DTE/DCE interface.

2.4.11.4 *Maximum number of outstanding I frames k*

The maximum number (k) of sequentially numbered I frames that the DTE or DCE may have outstanding (i.e., unacknowledged) at any given time is a system parameter which can never exceed seven. It shall be agreed for a period of time with the Administration.

Note – As a result of the further study proposed in § 2.2.4 above, the permissible maximum number of outstanding I frames may be increased.

3 Description of the packet level DTE/DCE interface

This and subsequent points of the Recommendation relate to the transfer of packets at the DTE/DCE interface. The procedures apply to packets which are successfully transferred across the DTE/DCE interface.

Each packet to be transferred across the DTE/DCE interface shall be contained within the link level information field which will delimit its length, and only one packet shall be contained in the information field.

Note 1 – Possible insertion of more than one packet in the link level information field is for further study.

Note 2 – At present, some networks require the data fields of packets to contain an integral number of octets. The transmission by the DTE of data fields not containing an integral number of octets to the network may cause a loss of data integrity.

Under urgent study are further considerations regarding the trends of future requirements and implementations toward either bit-orientation (any number of bits) or octet-orientation (an integral number of octets) for data fields in X.25 packets.

DTEs wishing universal operation on all networks should transmit all packets with data fields containing only an integral number of octets. Full data integrity can only be assured by exchange of octet-oriented data fields in both directions of transmission.

This point covers a description of the packet level interface for virtual call, permanent virtual circuit and datagram services. As designated in Recommendation X.2 [2], virtual call and permanent virtual circuit services are essential (E) services to be provided by all networks. Datagram service is designated as an additional (A) service which may be provided by some networks.

Note 3 – Under study are considerations regarding the amount of possible duplication between datagram, fast select and possible additional virtual call enhancements with the objective to minimize the variety of interfaces.

Procedures for the virtual circuit service (i.e., virtual call and permanent virtual circuit services) are specified in § 4. Procedures for the datagram service are specified in § 5. Packet formats for all services are specified in § 6. Procedures and formats for optional user facilities are specified in § 7.

3.1 *Logical channels*

To enable simultaneous virtual calls and/or permanent virtual circuits and/or datagrams, logical channels are used. Each virtual call, permanent virtual circuit, and datagram channel is assigned a logical channel group number (less than or equal to 15) and a logical channel number (less than or equal to 255). For virtual calls, a logical channel group number and a logical channel number are assigned during the call set-up phase. The range of logical channels used for virtual calls is agreed with the Administration at the time of subscription to the service (see Annex A). For permanent virtual circuits and datagram channels, logical channel group numbers and logical channel numbers are assigned in agreement with the Administration at the time of subscription to the service (see Annex A).

APPENDIX L

C/30 SITE PREPARATION

L.1 Physical Requirements

L.1.1 System Space

The C/30 system manufactured by the BBN Computer Corporation (BBN Computer) is available either in a cabinet supplied by BBN Computer, or as parts that may be installed in a rack provided by the customer. The requirements of these two packages are very different, as one is completely self-contained, and the other must have cooling and cable access provided by the customer. Many of the specifications are therefore different, and should be carefully noted. We recommend that all machines be ordered with cabinets to minimize potential operational and maintenance problems.

L.1.1.1 C/30 Systems in BBN Computer Cabinets

The enclosure for the C/30 system is 62 inches high, 25 inches wide, and 36 inches deep. With the rear door opened, the depth increases to 55 inches. To allow for service to the system, an additional space of 36 inches should be provided surrounding the entire cabinet. Note that up to four enclosures may be placed side by side before a 36 inch service space is required between cabinets.

L.1.1.2 C/30 Systems in Customer Provided Cabinets

The C/30 systems are also available as parts that are to be installed in a customer provided cabinet. Three distinct pieces will always be included: the power supply/card cage assembly, the microcassette drive, and the cable fantail assemblies. In addition, the system may be purchased with an Uninterruptable Power Supply (UPS).

The power supply/card cage assembly size is determined by the type and size of the system purchased. The small C/30 systems have a power supply/card cage assembly which is 12" high, 17" wide, and 22" deep, with a face plate that is 12.25" high and 19" wide. The large C/30 systems have a power supply/card cage assembly which is 19" high, 17" wide, and 22" deep, with a faceplate that is 19.25" high and 19" wide. Consult with BBN Computer Field Service for the precise information about the C/30 system purchased.

The microcassette drive should be mounted either directly above or below the power supply/card cage assembly. This tape drive is 5" high, 17" wide, and 7" deep, with a faceplate that is 5.25" high and 19" wide.

Inside the cabinet at the rear, and either above or below the power supply/card cage assembly, are the best positions to mount the fantail assemblies. The number of fantail assemblies is determined by the number and type of terminals to be connected to the system (1/2 fantail for each type of terminal - current loop, RS-232, or modem - and 1/2 fantail for each increment of eight terminals of the same type), plus one short fantail for the IMP or TIP systems. The terminal fantails are 5.25" high by 19" wide, and require about 2" of clearance on either side (front and back) for cable access. The short fantail for the IMP or TIP is 3.5" high and 19" wide, but will require about 4" of clearance on either side for cable access. All of the cables that connect the fantails to the interface controller in the card cage are about 6' long, but up to 14" of cable length may be lost depending upon where on the interface controller the cable must reach. Therefore, each fantail should not be mounted more than 4' from the power supply/card cage, and closer if this is possible. However, no fantails may ever be mounted directly behind the card cage, as this would seriously hamper any field service or maintenance to the system. Also, all cables must be dressed back to either side of the cabinet to allow free access to the card cage, and to insure better ventilation.

The Uninterruptable Power Supply option should be installed in the bottom of the cabinet, or on the floor very near the cabinet. The UPS is 10.5" high, 12" wide, and 19" deep. All of the power connections are on the front face of the UPS, so the unit should be installed facing the rear door of the cabinet.

L.1.2 Floor Type

Although a single C/30 system does not occupy much floor space, it can have large numbers of cables for connections to the system terminals. The best way of routing these cables is under a raised floor, which is also the safest, as no cables will be above the floor to trip the operators or other personnel. The raised floor may be of either the pedestal type or the raceway type, but the pedestal type offers the most flexibility for present and future system layout. The floor should be raised about 12" from the permanent floor, but this height may be changed by either the flooring contractor or the air conditioning personnel, if the air ducts are to be installed under the floor. Any openings in the floor should be protected from debris which might fall into them, either by covers, screens, or the system, and the openings should have smooth edges so as not to damage the cables. Cable access holes directly underneath the systems are the most useful, as each BBN Computer cabinet has a 6" by 8" hole in the bottom to allow cables to enter and leave.

Most importantly, the floor must be capable of supporting the weight of the system, which for the C/30 in BBN Computer cabinets is about 190 pounds (86Kg.).

L.1.3 Doorways

All of the doorways between the computer room and the loading dock must be checked to insure sufficient clearance to move the system into the computer room. If the C/30 system is to be installed in a customer supplied cabinet, all the boxes will be less than 36" in any dimension. The C/30 with a BBN Computer cabinet has a shipping size of not more than 71" high, 33" wide, and 44" deep. The cabinet is provided on a pallet which may be moved, but carefully, with a fork lift or pallet mover. If the shipping case must be removed from the cabinet before moving it (to fit through all of the doorways), the cabinet may be moved with a dolly set under either side, but not under the front or rear.

L.2 Power Requirements

L.2.1 AC Voltage

The standard input voltage that is required by the C/30 is 115VAC with no more than 10% variation high or low. When required, the system can be reconfigured for an input voltage of 220VAC, +-10%. In either case, the voltage provided must be single phase, grounded at the circuit box to either the building or another large metal surface. If the voltage is 115VAC the receptacle should be NEMA L5-30R; if the voltage is 220VAC, the receptacle should be NEMA L6-20R. (These receptacles are necessary only if the system is in a BBN cabinet. If the C/30 is installed in a customer provided cabinet, a standard 3-wire receptacle (NEMA 5-15R @115VAC, or NEMA 6-15P @220VAC) is sufficent for either the C/30 or the UPS.)

L.2.2 AC Current

The C/30 system in a BBN Computer cabinet is provided with a single Power Distribution Box which is rated to 30 Amps at 115VAC, and to 20 Amps at 220VAC. The plug on the Power Distribution Box for 115VAC is NEMA L5-30P (30 Amps, 125 Volt, twist-lock). The Power Distribution Box for 220VAC has a NEMA

L6-20P plug (20 Amps, 250 Volts, twist-lock). Each system must have a dedicated circuit breaker of the proper rating for the voltage used.

The C/30 system which is supplied for a customer provided cabinet will not be supplied with a Power Distribution Box. Instead, the system should be connected to the customer's cabinet power distribution, or directly into a wall (or floor) power receptacle. If the system is supplied with an Uninterruptable Power Supply, the system will be connected to the UPS, and then the UPS plugged into the power source. The current requirements of the C/30 without UPS are 6 Amperes at 115VAC (3 Amps at 220VAC), and with the UPS the current will be 8 Amperes at 115VAC (4 Amps at 220VAC).

L.2.3 AC Frequency

The systems and all of the peripherals standardly require 60 Hertz AC power, with a tolerance of +-1%. If requested, the system can be reconfigured to use 50 Hertz AC power (220V only), also with a tolerance of +-1.0%. In either case, loss of power for less than one cycle is tolerated by the internal power supplies. Loss of power for more than one cycle will cause a system reboot. If the system has an Uninterruptable Power Supply, any loss of power for up to 10 minutes will be masked and the system will continue to operate.

L.2.4 Grounding

All power outlets for the systems or the peripherals must be three-wire, grounded outlets. The ground wires should run back to the circuit box without interruption, and there be tied to the metal frame of the building, or to a large metal grounding plate. C/30 systems in BBN Computer cabinets have a ground bar inside the cabinet to which the system power supply, the cassette drive, and the Power Distribution Box are connected. These connections are to the AC Ground of each unit, except for the system power supply, where the AC and DC grounds are tied together.

For C/30 systems which are installed in customer supplied cabinets, the ground bar must also be supplied by the customer. The system power supply has a ground point marked on the rear of the supply, near the circuit breaker, which should be connected to the ground bar with 12 awg wire. On the microcassette drive, the recommended ground point is the bolt on the power supply transformer which is connected to the ground of the transformer. This should be connected to the cabinet ground bar with 12 awg wire. The interface cable from the system to the cassette has a ground wire which must be connected to the nearest screw which is used to fasten the cassette controller to the chasis. No additional grounding connection is required for the Uninterruptable Power Supply option.

L.3 Environmental Requirements

L.3.1 C/30 Systems in BBN Computer Cabinets

L.3.1.1 Ambient Temperature and Power Dissipation

The room in which the system is running should be maintained between 16 and 25 degrees C (60-77 F), with temperature changes of no greater than 4 degrees C (7 F) per hour. The small system and microcassette drive produce about 1190 Btu/hr (350 watts); the large system and cassette drive produce about 2220 Btu/hr (650 watts). With the Uninterruptable Power Supply option, the small system produces about 2190 Btu/hr (640 watts); the large system produces about 3220 Btu/hr (940 watts). If the system console is a video terminal, it produces about 390 Btu/hr (115 watts); if the console is a hard copy terminal, it produces about 15 Btu/hr (46 watts).

Storage temperature for both the system and the console may range from 0 to 45 degrees C (32-113 F).

L.3.1.2 Humidity

The humidity of the system room must be between 30% and 80%, with no condensation.

The storage humidity for the system, not including the cassette tapes, is 10% to 90%, non-condensing. See Section 4.2 for storage information on the cassette tapes.

L.3.1.3 Air Flow

The C/30 system does not require any special air flow past the cabinet, as long as the system room is within the ambient temperature range. The system is provided with its own flushing fan which will draw air into the system from the back door.

L.3.2 C/30 Systems in Customer Provided Cabinets

L.3.2.1 Ambient Temperature and Power Dissipation

The air immediately surrounding the power supply/card cage, micro-cassette drive, system console, and UPS (if ordered) should remain between 16 and 25 degrees C (60-77 F). The small system and cassette drive together produce about 1190 Btu/hr (350 watts); the large system and cassette drive produce about 2220 Btu/hr (650 watts). A video console produces about 390 Btu/hr (115 watts), and a hard copy terminal produces 155 Btu/hr (46 watts). A system with the UPS option produces an additional 1000 Btu/hr (290 watts).

The storage temperature for the system, UPS, and microcassette drive is 0 to 45 degrees C (32-113 F).

L.3.2.2 Humidity

The humidity of the air surrounding the system must remain between 30% and 80%, non-condensing.

The storage humidity of the system, excluding the cassette tapes, is 10% to 90%, also with no condensation.

L.3.2.3 Air Flow

Flushing fans must be provided in the Customer provided cabinet to keep the internal cabinet temperature within the proper range. These fans will need to be larger if the system has the UPS option, if the UPS is installed in the bottom of the same cabinet. Cabling from the system to the fantails should also be considered when designing the flushing fans, as the cables may impede the airflow around the power supply/card cage assembly.

L.4 Customer Supplied Parts

L.4.1 Terminal, Modem, and Host Cables

The C/30 systems are provided with cable fantails, up to 16 ports per fantail assembly from the asynchronous multiplexers, and up to 16 ports per fantail assembly from the synchronous modems and hosts. Cables between the fantail ports and the terminals or modems may be purchased from BBN Computer, or may be provided by the customer. The cables from the fantail ports to the distant host must be provided by the customer.

The following sections list the cable pinouts of the connectors of each fantail port.

L.4.1.1 RS-232 Cable Connector (with BBNCC Model 5423)

This connector is for either RS-232C terminals or modems. It is part of the FPT fantail, and is assembled in groups of 8 ports. If a terminal is to be connected to the port, the cable must be modified to be a null modem cable. This can be done by crossing the DSR pin of the fantail connector to the DTR pin of the terminal connector; crossing the DTR pin of the fantail connector to the DSR pin of the terminal connector; crossing the RTS pin of the fantail connector to the DCD pin of the terminal connector; crossing the DCD pin of the fantail connector to the RTS pin of the terminal connector; and jumpering pin 4 to pin 5 in each connector. If a modem is to be connected to the port, all signals should run straight through the cable with no crossing and no jumpers.

pin 1	Ground
2	Transmitted Data (by system)
3	Received Data (by system)
4	Request To Send
6	Data Set Ready
7	Signal Ground
8	Data Carrier Detect
9	+ VDC
10	- VDC
15	Transmitted Clock
17	Received Clock

20	Data Terminal Ready (from system)

The fantail port has a Cinch DB-25P (male) connector; the cable connector to modem or terminal must have a DB-25S connector or equivalent, which is available from TRW, AMP, and many others.

L.4.1.2 Current Loop Cable Connector (with BBNCC Model 5421)

This fantail port is for asynchronous 20ma current loop terminals. Current source is provided by the system for all these signals. The connectors are assembled in groups of 3 ports, and are found only as a part of the FPT fantail. All signals are straight through the cable without crossing, and no jumpers are required in either connector.

pin 1	Spare
2	Transimitter Source (from system)
3	Ground
4	Receiver Return (to terminal)
5	Spare
6	Spare
7	Transmitter Return (to system)
8	Receiver Source (from terminal)
9	Spare

The fantail port has a Cinch DE-9S (female) connector, cable to terminal must have a DE-9P (male) or equivalent, which is available from TRW, AMP, and many others.

L.4.1.3 EIA Terminal Connector (with BBNCC Model 5422)

This fantail port is designed for terminals with an EIA RS-232 interface to the system. The connectors are part of the FPT fantail assembly, with 8 connectors as ports for the asnchronous multiplexor. There are no crossed signals or jumpers added to this cable.

pin 1	Ground
2	Transmitted Data (by terminal)
3	Received Data (by terminal)
4	Request To Send
5	Clear To Send
6	Data Set Ready
7	Signal Ground
8	Data Carrier Detect
9	VDC
10	- VDC

The fantail port has a Cinch DB-25P (female) connector; cable to terminal must have a DB-25S (male) or equivalent, which is available from TRW, AMP, and many others.

L.4.1.4 RE-232/CCITT V.28 Cable Connector (with BBNCC Model 5432)

This connector is very similiar to the RS-232 terminal and modem cable connector, except that it includes the CTS signal, and it is used only as a part of the FPI fantail assembly, by single units instead of groups of 8 ports. Also, this port will only support synchronous modems, and will not accept terminals. All signals should be straight through the cable, with no crossing or jumpers required.

pin 1	Ground
2	Transmitted Data (by system)
3	Received Data (by system)
4	Request To Send
5	Clear To Send
6	Data Set Ready
7	Signal Ground
8	Data Carrier Detect
9	+ VDC
10	- VDC
15	Transmitted Clock
17	Received Clock
20	Data Terminal Ready (from system)

The fantail port has a Cinch DB-25P (male) connector; cable to modem must have a DB-25S (female) connector or equivalent, which is available from TRW, AMP, and many others.

L.4.1.5 Bell 303 Modem Cable Connector (with BBNCC Model 5431)

This port is designed for connections to the Bell 303 modem. It is a part of the FPI fantail assembly and is provided as a single port, not part of a group.

pair 1	Send Data	pin 8
1	Ground	27
2	Loop Test	11
2	Ground	30
3	Serial Clock Transmit	12
3	Ground	31
4	Read Data	10
4	Ground	29
5	Serial Clock Receive	6
5	Ground	25

Cable must be twisted pair, 22 awg or larger. The suggested cable is Alpha 1323. Fantail port has a Cinch DC-37P (male) connector; cable to modem must have DC-37S (female) connector or equivalent, which is available from TRW, AMP, and many others.

L.4.1.6 Local Host Cable Connection (with BBNCC Model 5441)

This is the port for the connection of a Local Host to a C/30 IMP or TIP. For further information about this connection, refer to the body of this report. Connectors are part of the FPT fantail assembly.

pair 1	Ready for next IMP bit	pin 1
1	Ground	20
2	There's your IMP bit	2
2	Ground	21
3	Last IMP Bit	3

3	Ground	22
4	IMP Data	4
4	Ground	23
5	Ready for Next Host Bit	5
5	Ground	24
6	There's Your Host Bit	6
6	Ground	25
7	Last Host Bit	7
7	Ground	26
8	Host Data	8
8	Ground	27
9	IMP Master Ready	9
9	Ground	28
10	IMP Ready Test	10
10	Ground	29
11	Host Master Ready	11
11	Ground	30
12	Host Ready Test	12
12	Ground	31

The cable for this connection is supplied by BBNCC, and is the same as that specified in Report 1822 for the Pluribus. The fantail port has a Cinch DC-37S (female connector; the cable has a Cinch DC-37P (male) connector at the C/30 end and has a frayed end at the host end.

L.4.1.7 Distant Host Cable Connection (with BBNCC Model 5442)

This is the port for the connection of a Distant Host to a C/30 IMP or TIP. For further information about this connection, refer to the body of this report. Connectors are part of the FPT fantail assembly.

pair 1	Ready For Next IMP Bit +	pin 1
1	Ready For Next IMP Bit -	20
2	There's Your IMP Bit +	15
2	There's Your IMP Bit -	33
3	Last IMP Bit +	17
3	Last IMP Bit -	35
4	IMP Data +	19
4	IMP Data -	37
5	Ready For Next Host Bit +	13
5	Ready For Next Host Bit -	32
6	There's Your Host Bit +	6
6	There's Your Host Bit -	25
7	Last Host Bit +	7
7	Last Host Bit -	26
8	Host Data +	8
8	Host Data -	27
9	IMP Ready Test	9
9	IMP Master Ready	10
10	Host Master Ready	11
10	Host Ready Test	12

GND	Shield Ground	22

Cable must be twisted pair, 22 awg or larger. Recommended cables for distances less than 300 feet are Columbia 6059 or Belden 9768 (22AWG, 12 pair; U.L. Listing #2493). For distances greater than 300 feet, the recommended cables are SigNet Type Tel-U/SN or Direct Burial Cable, REA Spec. PE-23 (19AWG, 12 pair). The fantail port has a Cinch DC-37S (female) connector; cable to IMP or Host must have a DC-37P (male) connector. These connectors are available from TRW, AMP, and many others. If the cable to the IMP or Host is already constucted with the MIL SPEC connector specified in Report 1822 (MS24266R18B31PN), a DIDX adaptor cable may be ordered from BBN Computer to connect the IMP or Host cable to the fantail port connector which is available from TRW, AMP, and many others.

L.4.2 Storage Media

The only storage medium needed to operate the C/30 is a microcassette supplied by BBN Computer. These can be stored in a temperature between 10 and 40 degrees C (50-104 F) with a temperature change of no more than 3.9 degrees C (7 F) per hour. Humidity should be between 4% and 80% with no condensation. Note that the rate of temperature change must not be exceeded in transfer from storage to use.

L.4.3 Computer Room

Aside from the raised floor (if used), many other details must be considered when planning a computer room. Some of these concern the system performance and reliability, and some concern safety precautions for the system and the operators. Cosmetic details, such as the paint colors of the system, should also be considered.

L.4.3.1 Air

The air in the computer room should not only be maintained at the proper temperature and humidity, but must also be clean and as non-contaminated as possible. That is, the air should be filtered by the air conditioning to the room, and if possible, the air pressure of the room should be greater than that of the rooms surrounding it such that an open door will not allow dust to enter the room. Smoking should be prohibited in the computer room. The same precautions should be taken with the room or area where the system media are stored. Dust of any sort may cause damage to the storage media, which may also damage the drive if the media are re-mounted.

L.4.3.2 Floor, Ceiling, and Wall Surfaces

The floor of the computer room, whether raised or not, must be as static-free as possible. This is best done with tile, although some types of tile, particularly asbestos, may cause as much static as carpets. In addition, the floor, ceiling, and walls should be non-combustible or at least fire retardant. If possible, the walls should also be sound absorbing, to reduce the noise level in the computer room caused by the system, the air conditioning, and the operators.

It is often desirable to match the colors of the computer room to the colors of the system. BBN Computer cabinets are provided with only one color scheme. The sides of the cabinets are blue, and the center section of the cabinets are off-white. The blue paint is colorchip #25102 from Federal Standard 595. The off-white paint is colorchip #27778, also from Federal Standard 595.

L.4.3.3 Fire Precautions

In addition to fire retardant surfaces on the floor, ceiling, and walls, fire extinguishers should be installed, and clearly labelled, in the computer room. Extinguishers are necessary for both the electrical systems and for the combustible paper products in the room. Sprinkler systems may also be installed, but care should be taken in planning the sprinklers such that their use will not damage the system, the storage media, or the operators. For more complete information, refer to the National Fire Protection Association's Standard for the Protection of Electronic Computer/Data Processing Equipment, NFPA No. 75.

L.4.3.4 Electrical Precautions

Located near each door of the computer room should be a master power switch for all of the equipment in the room. This is often a part of the local electrical code, and is recommended for all system installations whether required by the code or not.

Request for Comments: 878
Obsoletes RFCs: 851, 802

The ARPANET 1822L Host Access Protocol

RFC 878

Andrew G. Malis
ARPANET Mail: malis@bbn-unix

BBN Communications Corp.
50 Moulton St.
Cambridge, MA 02238

December 1983

This RFC specifies the ARPANET 1822L Host Access Protocol, which is a successor to the existing 1822 Host Access Protocol. 1822L allows ARPANET hosts to use logical names as well as 1822's physical port loca-ions to address each other.

Table of Contents

FIGURES

1 INTRODUCTION

This RFC specifies the ARPANET 1822L Host Access Protocol, which will allow hosts to use logical addressing (i.e., host names that are independent of their physical location on the ARPANET) to communicate with each other. This new host access protocol is known as the ARPANET 1822L (for Logical) Host Access Protocol, and is a successor to the current ARPANET 1822 Host Access Protocol, which is described in sections 3.3 and 3.4 of BBN Report 1822 [1]. Although the 1822L protocol uses different Host-IMP leaders than the 1822 protocol, the IMPs will continue to support the 1822 protocol, and hosts using either protocol can readily communicate with each other (the IMPs will handle the translation automatically).

The RFC's terminology is consistent with that used in Report 1822, and any new terms will be defined when they are first used. Familiarity with Report 1822 (section 3 in particular) is assumed. As could be expected, the RFC makes many references to Report 1822. As a result, it uses, as a convenient abbreviation, "see 1822(x)" instead of "please refer to Report 1822, section x, for further details".

This RFC updates, and obsoletes, RFC 851. The changes from that RFC are:

o Section 2.2.4 was rewritten for clarity.

o Section 2.5 was expanded to further discuss the effects of using 1822L names on host-to-host virtual circuits.

o In section 3.2, the type 1 IMP-to-host message has two new subtypes, the type 9 message has one new subtype, and the type 15, subtype 4 message is no longer defined.

o An appendix describing the mapping between 1822L names and internet (IP) addresses has been added.

All of these changes to RFC 851 are marked by revision bars (as |
shown here) in the right margin. |

2 THE ARPANET 1822L HOST ACCESS PROTOCOL

The ARPANET 1822L Host Access Protocol allows a host to use logical addressing to communicate with other hosts on the ARPANET. Basically, logical addressing allows hosts to refer to each other using an 1822L name (see section 2.1) which is independent of a host's physical location in the network. IEN 183 (also published as BBN Report 4473) [2] gives the use of logical addressing considerable justification. Among the advantages it cites are:

o The ability to refer to each host on the network by a name independent of its location on the network.

o Allowing different hosts to share the same host port on a time-division basis.

o Allowing a host to use multi-homing (where a single host uses more than one port to communicate with the network).

o Allowing several hosts that provide the same service to share the same name.

The main differences between the 1822 and 1822L protocols are the format of the leaders that are used to introduce messages between a host and an IMP, and the specification in those leaders of the source and/or destination host(s). Hosts have the choice of using the 1822 or the 1822L protocol. When a host comes up on an IMP, it declares itself to be an 1822 host or an 1822L host by the type of NOP message (see section 3.1) it uses. Once up, hosts can switch from one protocol to the other by issuing an appropriate NOP. Hosts that do not use the 1822L protocol will still be addressable by and can communicate with hosts that do, and vice-versa.

Another difference between the two protocols is that the 1822 leaders are symmetric, while the 1822L leaders are not. The term symmetric means that in the 1822 protocol, the exact same leader format is used for messages in both directions between the hosts and IMPs. For example, a leader sent from a host over a cable that was looped back onto itself (via a looping plug or faulty hardware) would arrive back at the host and appear to be a legal message from a real host (the destination host of the original message). In contrast, the 1822L headers are not symmetric, and a host can detect if the connection to its IMP is looped by receiving a message with the wrong leader format. This allows the host to take appropriate action upon detection of the loop.

2.1 Addresses and Names

The 1822 protocol defines one form of host specification, and the 1822L protocol defines two additional ways to identify network hosts. These three forms are 1822 addresses, 1822L names, and 1822L addresses.

1822 addresses are the 24-bit host addresses found in 1822 leaders. They have the following format:

```
  1              8 9                              24
 +----------------+---------------------------------+
 |                |                                 |
 |  Host number   |           IMP number            |
 |                |                                 |
 +----------------+---------------------------------+
```

1822 Address Format
Figure 2.1

These fields are quite large, and the ARPANET will never use more than a fraction of the available address space. 1822 addresses are used in 1822 leaders only.

1822L names are 16-bit unsigned numbers that serve as a logical identifier for one or more hosts. 1822L names have a much simpler format:

```
 1                                16
+--------------------------------+
|                                |
|           1822L name           |
|                                |
+--------------------------------+
```

1822L Name Format
Figure 2.2

The 1822L names are just 16-bit unsigned numbers, except that bits 1 and 2 are not both zeros (see below). This allows over 49,000 hosts to be specified.

1822 addresses cannot be used in 1822L leaders, but there may be a requirement for an 1822L host to be able to address a specific physical host port or IMP fake host. 1822L addresses are used for this function. 1822L addresses form a subset of the 1822L name space, and have both bits 1 and 2 off.

```
  1   2   3           8 9             16
+---+---+------------+----------------+
|   |   |            |                |
| 0 | 0 |   host #   |   IMP number   |
|   |   |            |                |
+---+---+------------+----------------+
```

1822L Address Format
Figure 2.3

This format allows 1822L hosts to directly address hosts 0-63 at IMPs 1-255 (IMP 0 does not exist). Note that the highest host numbers are reserved for addressing the IMP's internal fake hosts. At this writing, the IMP has seven fake hosts, so host numbers 57-63 address the IMP fake hosts, while host numbers 0-56 address real hosts external to the IMP. As the number of IMP fake hosts changes, this boundary point will also change.

2.2 Name Translations

There are a number of factors that determine how an 1822L name is translated by the IMP into a physical address on the network. These factors include which translations are legal; in what order different translations for the same name should be attempted; which legal translations shouldn't be attempted because a particular host port is down; and the interoperability between 1822 and 1822L hosts. These issues are discussed in the following sections.

2.2.1 Authorization and Effectiveness

Every host on a C/30 IMP, regardless of whether it is using the 1822 or 1822L protocol to access the network, can have one or more 1822L names (logical addresses). Hosts using 1822L can then use these names to address the hosts in the network independent of their physical locations. Because of the implementation constraints mentioned in the introduction, hosts on non-C/30 IMPs cannot be assigned 1822L names. To circumvent this restriction, however, 1822L hosts can also use 1822L addresses to access all of the other hosts.

At this point, several questions arise: How are these names assigned, how do they become known to the IMPs (so that translations to physical addresses can be made), and how do the IMPs know which host is currently using a shared port? To answer each question in order:

Names are assigned by a central network administrator. When each name is created, it is assigned to a host (or a group of hosts) at one or more specific host ports. The host(s) are allowed to reside at those specific host ports, and nowhere else. If a host moves, it will keep the same name, but the administrator has to update the central database to reflect the new host port. Changes to this database are distributed to the IMPs by the Network Operations Center (NOC). For a while, the host may be allowed to reside at either of (or both) the new and old ports. Once the correspondence between a name and one or more hosts ports where it may be used has been made official by the administrator, that name is said to be authorized. 1822L addresses, which actually refer to physical host ports, are always authorized in this sense.

Once a host has been assigned one or more names, it has to let the IMPs know where it is and what name(s) it is using. There are two cases to consider, one for 1822L hosts and another for 1822 hosts. The following discussion only pertains to hosts on C/30 IMPs.

When an IMP sees an 1822L host come up on a host port, the IMP has no way of knowing which host has just come up (several hosts may share the same port, or one host may prefer to be known by different names at different times). This requires the host to declare itself to the IMP before it can actually send and receive messages. This function is performed by a new host-to-IMP message, the Name Declaration Message (NDM), which lists the names that the host would like to be known by. The IMP checks its tables to see if each of the names is authorized, and sends an NDM Reply to the host saying which names were actually authorized and can now be used for sending and receiving messages (i.e., which names are effective). A host can also use an NDM message to change its list of effective names (it can add to and delete from the list) at any time. The only constraint on the host is that any names it wishes to use can become effective only if they are authorized.

In the second case, if a host comes up on a C/30 IMP using the 1822 protocol, the IMP automatically makes the first name the IMP finds in its tables for that host become effective when it receives the first 1822 NOP from the host. Thus, even though the host is using the 1822 protocol, it can still receive messages from 1822L hosts via its 1822L name. Of course, it can also receive messages from an 1822L host via its 1822L address as well. (Remember, the distinction between 1822L names and addresses is that the addresses correspond to physical locations on the network, while the names are strictly logical identifiers). The IMPs translate between the different leaders and send the proper leader in each case (see section 2.2.4).

The third question above has by now already been answered. When an 1822L host comes up, it uses the NDM message to tell the IMP which host it is (which names it is known by). Even if this is a shared port, the IMP knows which host is currently connected.

Whenever a host goes down, its names automatically become non-effective. When it comes back up, it has to make them effective again.

2.2.2 Translation Policies

Several hosts can share the same 1822L name. If more than one of these hosts is up at the same time, any messages sent to that 1822L name will be delivered to just one of the hosts sharing that name, and a RFNM will be returned as usual. However, the sending host will not receive any indication of which host received the message, and subsequent messages to that name are not guaranteed to be sent to the same host. Typically, hosts providing exactly the same service could share the same 1822L name in this manner.

Similarly, when a host is multi-homed, the same 1822L name may refer to more than one host port (all connected to the same host). If the host is up on only one of those ports, that port will be used for all messages addressed to the host. However, if the host were up on more than one port, the message would be delivered over just one of those ports, and the subnet would choose which port to use. This port selection could change from message to message. If a host wanted to insure that certain messages were delivered to it on specific ports, these messages could use either the port's 1822L address or a specific 1822L name that referred to that port alone.

Three different address selection policies are available for the name mapping process. When translated, each name uses one of the three policies (the policy is pre-determined on a per-name basis). The three policies are:

- Attempt each translation in the order in which the physical addresses are listed in the IMP's translation tables, to find the first reachable physical host address. This list is always searched from the top whenever an uncontrolled packet is to be sent or a new virtual circuit connection has to be created (see section 2.5). This is the most commonly used policy.

- Selection of the closest physical address, which uses the IMP's routing tables to find the translation to the destination IMP with the least delay path whenever an uncontrolled packet is to be sent or a new virtual circuit connection has to be created.

- Use load leveling. This is similar to the second policy, but differs in that searching the address list for a valid translation starts at the address following where the previous translation search ended whenever an uncontrolled packet is to be sent or a new virtual circuit connection has to be created. This attempts to spread out the load from any one IMP's hosts to the various host ports associated with a particular name. Note that this is NOT network-wide load leveling, which would require a distributed algorithm and tables.

2.2.3 Reporting Destination Host Downs

As was explained in report 1822, and as will be discussed in greater detail in section 2.5, whenever regular messages are sent by a host, the IMP opens a virtual circuit connection to each destination host from the source host. A connection will stay open at least as long as there are any outstanding (un-RFNMed) messages using it and both the source and destination hosts stay up.

However, the destination host may go down for some reason during the lifetime of a connection. If the host goes down while there are no outstanding messages to it in the network, then the connection is closed and no other action is taken until the source host submits the next message for that destination. At that time, ONE of the following events will occur:

A1. If 1822 or an 1822L address is being used to specify the destination host, then the source host will receive a type 7 (Destination Host Dead) message from the IMP.

A2. If an 1822L name is being used to specify the destination host, and the name maps to only one authorized host port, then a type 7 message will also be sent to the source host.

A3. If an 1822L name is being used to specify the destination host, and the name maps to more than one authorized host port, then the IMP attempts to open a connection to another authorized and effective host port for that name. If no such connection can be made, the host will receive a type 15 (1822L Name or Address Error), subtype 5 (no effective translations) message (see section 3.2). Note that a type 7 message cannot be returned to the source host, since type 7 messages refer to a particular destination host port, and the name maps to more than one destination port.

Things get a bit more complicated if there are any outstanding messages on the connection when the destination host goes down. The connection will be closed, and one of the following will occur:

B1. If 1822 or an 1822L address is being used to specify the destination host, then the source host will receive a type 7 message for each outstanding message.

B2. If an 1822L name is being used to specify the destination host, then the source host will receive a type 9 (Incomplete | Transmission), subtype 6 (message lost due to logically | addressed host going down) message for each outstanding |

message. The next time the source host submits another message for that same destination name, the previous algorithm will be used (either step A2 or step A3).

The above two algorithms also apply when a host stays up, but declares the destination name for an existing connection to no longer be effective. In this case, however, the type 7 messages above will be replaced by type 15, subtype 3 (name not effective) messages.

Section 2.3 discusses how destination host downs are handled for uncontrolled packets.

2.2.4 1822L and 1822 Interoperability

As has been previously stated, 1822 and 1822L hosts can intercommunicate, and the IMPs will automatically handle any necessary leader and address format conversions. However, not every combination of 1822 and 1822L hosts allows full interoperability with regard to the use of 1822L names, since 1822 hosts are restricted to using physical addresses.

There are two possible situations where any incompatibility could arise: |

o An 1822 host sending a message to an 1822L host: The 1822 host specifies the destination host by its 1822 address. The destination host will receive the message with an 1822L leader containing the 1822L addresses of the source and destination hosts. |

o An 1822L host sending a message to an 1822 host: The 1822L host can use 1822L names or addresses to specify both the source and destination hosts. The destination host will receive the message with an 1822 leader containing the 1822 address of the source host. |

2.3 Uncontrolled Packets

Uncontrolled packets (see 1822(3.6)) present a unique problem for the 1822L protocol. Uncontrolled packets use none of the normal ordering and error-control mechanisms in the IMP, and do not use the normal virtual circuit connection facilities. As a result, uncontrolled packets need to carry all of their overhead with them, including source and destination names. If 1822L names are used when sending an uncontrolled packet, additional information is now required by the subnetwork when the packet is transferred to the destination IMP. This means that less host-to-host data can be contained in the packet than is possible between 1822 hosts.

Uncontrolled packets that are sent between 1822 hosts may contain not more than 991 bits of data. Uncontrolled packets that are sent to and/or from 1822L hosts are limited to 32 bits less, or not more than 959 bits. Packets that exceed this length will result in an error indication to the host, and the packet will not be sent. This error indication represents an enhancement to the previous level of service provided by the IMP, which would simply discard an overly long uncontrolled packet without notification.

Other enhancements that are provided for uncontrolled packet service are a notification to the host of any errors that are detected by the host's IMP when it receives the packet. A host will be notified if an uncontrolled packet contains an error in the 1822L name specification, such as if the name is not authorized or effective, if the remote host is unreachable (which is indicated by none of its names being effective), if network congestion control throttled the packet before it left the source IMP, or for any other reason the source IMP was not able to send the packet on its way.

In most cases, the host will not be notified if the uncontrolled packet was lost once it was transmitted by the source IMP.

However, the IMP will attempt to notify the source host if a logically-addressed uncontrolled packet was mistakenly sent to a host that the source IMP thought was effective, but which turned out to be dead or non-effective at the destination IMP. This non-delivery notice is sent back to the source IMP as an uncontrolled packet from the destination IMP, so the source host is not guaranteed to receive this indication.

If the source IMP successfully receives the non-delivery notice, then the source host will receive a type 15 (1822L Name or Address Error), subtype 6 (down or non-effective port) message. If the packet is resubmitted or another packet is sent to the same destination name, and there are no available effective translations, then the source host will receive a type 15, subtype 5 (no effective translations) message if the destination name has more than one mapping; or will receive either a type 7 (Destination Host Dead) or a type 15, subtype 3 (name not effective) message if the destination name has a single translation.

Those enhancements to the uncontrolled packet service that are not specific to logical addressing will be available to hosts using 1822 as well as 1822L. However, uncontrolled packets must be sent using 1822L leaders in order to receive any indication that the packet was lost once it has left the source IMP.

2.4 Establishing Host-IMP Communications

When a host comes up on an IMP, or after there has been a break in the communications between the host and its IMP (see 1822(3.2)), the orderly flow of messages between the host and the IMP needs to be properly (re)established. This allows the IMP and host to recover from most any failure in the other or in their communications path, including a break in mid-message.

The first messages that a host should send to its IMP are three NOP messages. Three messages are required to insure that at least one message will be properly read by the IMP (the first NOP could be concatenated to a previous message if communications had been broken in mid-stream, and the third provides redundancy for the second). These NOPs serve several functions: they synchronize the IMP with the host, they tell the IMP how much padding the host requires between the message leader and its body, and they also tell the IMP whether the host will be using 1822 or 1822L leaders.

Similarly, the IMP will send three NOPs to the host when it detects that the host has come up. Actually, the IMP will send six NOPs, alternating three 1822 NOPs with three 1822L NOPs. Thus, the host will see three NOPs no matter which protocol it is using. The NOPs will be followed by two Interface Reset messages, one of each style. If the IMP receives a NOP from the host while the above sequence is occurring, the IMP will only send the remainder of the NOPs and the Interface Reset in the proper style. The 1822 NOPs will contain the 1822 address of the host interface, and the 1822L NOPs will contain the corresponding 1822L address.

Once the IMP and the host have sent each other the above messages, regular communications can commence. See 1822(3.2) for further details concerning the ready line, host tardiness, and other issues.

2.5 Counting RFNMs When Using 1822L

When a host submits a regular message using an 1822 leader, the IMP checks for an existing simplex virtual circuit connection (see 1822(3.1)) from the source host to the destination host. If such a connection already exists, it is used. Otherwise, a new connection from the source host port to the destination host port is opened. In either case, there may be at most eight messages outstanding on that connection at any one time. If a host submits a ninth message on that connection before it receives a reply for the first message, then the host will be blocked until the reply is sent for the first message.

Such connections can stay open for some time, but are timed out after three minutes of no activity, or can be closed if there is contention for the connection blocks in either the source or destination IMP. However, a connection will never be closed as long as there are any outstanding messages on it. This allows a source host to count the number of replies it has received for messages to each destination host address in order to avoid being blocked by submitting a ninth outstanding message on any connection.

When a host submits a regular message using an 1822L leader, a similar process occurs, except that in this case, connections are distinguished by the source port/source name/destination name combination. When the message is received from a host, the IMP first looks for an open connection for that same port and source name/destination name pair. If such a connection is found, then it is used, and no further name translation is performed. If, however, no open connection was found, then the destination name is translated, and a connection opened to the physical host port. As long as there are any outstanding messages on the connection it will stay open, and it will have the same restriction that only eight messages may be outstanding at any one time. Thus, a source host can still count replies to avoid being blocked, but they must be counted on a source port and source name/destination name pair basis, instead of just by source port and destination host address as before.

Since connections are based on the source name as well as the destination name, this implies that there may be more than one open connection from physical host port A to physical host port B, which would allow more than 8 outstanding messages simultaneously from the first to the second port. However, for this to occur, either the source or destination names, or both, must differ from one connection to the next. For example, if the names "543" and "677" both translate to physical port 3 on IMP 51, then the host on that port could open four connections to itself by sending messages from "543" to "543", from "543" to "677", from "677" to "543", and from "677" to "677".

As has already been stated, the destination names in regular messages are only translated when connections are first opened. Once a connection is open, that connection, and its destination physical host port, will continue to be used until it is closed. If, in the meantime, a "better" destination host port belonging to the same destination name became available, it would not be used until the next time a new connection is opened to that destination name.

Also, the act of making an 1822L name be non-effective will not |
automatically cause any connections using that name to be closed. |
However, they will be closed after at most three minutes of |
inactivity. A host can, if it wishes, make all of its names at a |
port be noneffective and close all of its connections to and from |
the port by flapping the host's ready line to that IMP port. |

2.6 1822L Name Server

There may be times when a host wants to perform its own translations, or might need the full list of physical addresses to which a particular name maps. For example, a connection-based host-to-host protocol may require that the same physical host port on a multi-homed host be used for all messages using that host-to-host connection, and the host does not wish to trust the IMP to always deliver messages using a destination name to the same host port.

In these cases, the host can submit a type 11 (Name Server Request) message to the IMP, which requests the IMP to translate the destination 1822L name and return a list of the addresses to which it maps. The IMP will respond with a type 11 (Name Server Reply) message, which contains the selection policy in use for that name, the number of addresses to which the name maps, the addresses themselves, and for each address, whether it is effective and its routing distance from the IMP. See section 3.2 for a complete description of the message's contents.

Using this information, the source host could make an informed decision on which of the physical host ports corresponding to an 1822L name to use and then send the messages to that port, rather than to the name.

The IMP also supports a different type of name service. A host needs to issue a Name Declaration Message to the IMP in order to make its names effective, but it may not wish to keep its names in some table or file in the host. In this case, it can ask the IMP to tell it which names it is authorized to use.

In this case, the host submits a type 12 (Port List Request) message to the IMP, and the IMP replies with a type 12 (Port List Reply) message. It contains, for the host port over which the IMP received the request and sent the reply, the number of names that map to the port, the list of names, and whether or not each name is effective. The host can then use this information in order to issue the Name Declaration Message. Section 3.2 contains a complete description of the reply's contents.

3 1822L LEADER FORMATS

The following sections describe the formats of the leaders that precede messages between an 1822L host and its IMP. They were designed to be as compatible with the 1822 leaders as possible. The second, fifth, and sixth words are identical in the two leaders, and all of the existing functionality of the 1822 leaders has been retained. In the first word, the 1822 New Format Flag is now also used to identify the two types of 1822L leaders, and the Handling Type has been moved to the second byte. The third and fourth words contain the Source and Destination 1822L Name, respectively.

3.1 Host-to-IMP 1822L Leader Format

```
1      4 5     8 9            16
+-------+-------+--------------+
|       | 1822L |              |
| Unused|  H2I  | Handling Type|
|       | Flag  |              |
+-------+-------+--------------+
17    20 21 22 24 25          32
+-------+-+-----+--------------+
|       |T|Leader|             |
| Unused|R|Flags |  Message Type|
|       |C|     |              |
+-------+-+-----+--------------+
33                            48
+------------------------------+
|                              |
|         Source Host          |
|                              |
+------------------------------+
49                            64
+------------------------------+
|                              |
|       Destination Host       |
|                              |
+------------------------------+
65                   76 77    80
+----------------------+-------+
|                      |       |
|      Message ID      |Sub-type|
|                      |       |
+----------------------+-------+
81                            96
+------------------------------+
|                              |
|            Unused            |
|                              |
+------------------------------+
```

Host-to-IMP 1822L Leader Format
Figure 3.1

Bits 1-4: Unused, must be set to zero.

Bits 5-8: 1822L Host-to-IMP Flag:

This field is set to decimal 13 (1101 in binary).

Bits 9-16: Handling Type:

This field is bit-coded to indicate the transmission characteristics of the connection desired by the host. See 1822(3.3).

Bit 9: Priority Bit:

Messages with this bit on will be treated as priority messages.

Bits 10-16: Unused, must be zero.

Bits 17-20: Unused, must be zero.

Bit 21: Trace Bit:

If equal to one, this message is designated for tracing as it proceeds through the network. See 1822(5.5).

Bits 22-24: Leader Flags:

Bit 22: A flag available for use by the destination host. See 1822(3.3) for a description of its use by the IMP's TTY Fake Host.

Bits 23-24: Reserved for future use, must be zero.

Bits 25-32: Message Type:

Type 0: Regular Message - All host-to-host communication occurs via regular messages, which have several sub-types, found in bits 77-80. These sub-types are:

0: Standard - The IMP uses its full message and error control facilities, and host blocking may occur.

3: Uncontrolled Packet - The IMP will perform no message-control functions for this type of message, and network flow and congestion control may cause loss of the packet. Also see 1822(3.6) and section 2.3.

1-2,4-15: Unassigned.

Type 1: Error Without Message ID - See 1822(3.3).

Type 2: Host Going Down - see 1822(3.3).

Type 3: Name Declaration Message (NDM) - This message is used by the host to declare which of its 1822L names is or is not effective (see section 2.2.1), or to make all of its names non-effective. The first 16 bits of the data portion of the NDM message, following the leader and any leader padding, contains the number of 1822L names contained in the message. This is followed by the 1822L name entries, each 32 bits long, of which the first 16 bits is a 1822L name and the second 16 bits contains either of the integers zero or one. Zero indicates that the name should not be effective, and one indicates that the name should be effective. The IMP will reply with a NDM Reply message (see section 3.2) indicating which of the names are now effective and which are not. Pictorially, a NDM message has the following format (including the leader, which is printed in hexadecimal, and without any leader padding):

```
 1              16 17            32 33            48
+----------------+----------------+----------------+
|                |                |                |
|      0D00      |      0003      |      0000      |
|                |                |                |
+----------------+----------------+----------------+
 49             64 65            80 81            96
+----------------+----------------+----------------+
|                |                |                |
|      0000      |      0000      |      0000      |
|                |                |                |
+----------------+----------------+----------------+
 97            112 113          128 129          144
+----------------+----------------+----------------+
|                |                |                |
|  # of entries  | 1822L name #1  |     0 or 1     |
|                |                |                |
+----------------+----------------+----------------+
145            160 161          176
+----------------+----------------+
|                |                |
| 1822L name #2  |     0 or 1     |      etc.
|                |                |
+----------------+----------------+
```

NDM Message Format
Figure 3.2

An NDM with zero entries will cause all current effective names for the host to become non-effective.

Type 4: NOP - This allows the IMP to know which style of leader the host wishes to use. A 1822L NOP signifies that the host wishes to use 1822L leaders, and an 1822 NOP signifies that the host wishes to use 1822 leaders. All of the other remarks concerning the NOP message in

1822(3.3) still hold. The host should always issue NOPs in groups of three to insure proper reception by the IMP. Also see section 2.4 for a further discussion on the use of the NOP message.

Type 8: Error with Message ID - see 1822(3.3).

Type 11: Name Server Request - This allows the host to use the IMP's logical addressing tables as a name server. The destination name in the 1822L leader is translated, and the IMP replies with a Name Server Reply message, which lists the physical host addresses to which the destination name maps.

Type 12: Port List Request - This allows the physical host to request the list of names that map to the host port over which this request was received by the IMP. The IMP replies with a Port List Reply message, which lists the names that map to the port.

Types 5-7,9-10,13-255: Unassigned.

Bits 33-48: Source Host:

This field contains one of the source host's 1822L names (or, alternatively, the 1822L address of the host port the message is being sent over). This field is not automatically filled in by the IMP, as in the 1822 protocol, because the host may be known by several names and may wish to use a particular name as the source of this message. All messages from the same host need not use the same name in this field. Each source name, when used, is checked for authorization, effectiveness, and actually belonging to this host. Messages using names that do not satisfy all of these requirements will not be delivered, and will instead result in an error message being sent back into the source host. If the host places its 1822L address in this field, the address is checked to insure that it actually represents the host port where the message originated.

Bits 49-64: Destination Host:

This field contains the 1822L name or address of the destination host. If it contains a name, the name will be checked for effectiveness, with an error message returned to the source host if the name is not effective.

Bits 65-76: Message ID:

This is a host-specified identification used in all type 0 and type 8 messages, and is also used in type 2 messages. When used in type 0 messages, bits 65-72 are also known as the Link Field, and should contain values specified in Assigned Numbers [3] appropriate for the host-to-host protocol being used.

Bits 77-80: Sub-type:

This field is used as a modifier by message types 0, 2, 4, and 8.

Bits 81-96: Unused, must be zero.

3.2 IMP-to-Host 1822L Leader Format

```
 1       4 5       8 9              16
+---------+---------+----------------+
|         |  1822L  |                |
| Unused  |  I2H    | Handling Type  |
|         |  Flag   |                |
+---------+---------+----------------+
 17     20 21  22 24 25             32
+---------+-+-------+----------------+
|         |T|Leader|                 |
| Unused  |R|Flags |   Message Type  |
|         |C|      |                 |
+---------+-+-------+----------------+
 33                                 48
+------------------------------------+
|                                    |
|            Source Host             |
|                                    |
+------------------------------------+
 49                                 64
+------------------------------------+
|                                    |
|          Destination Host          |
|                                    |
+------------------------------------+
 65                      76 77      80
+--------------------------+---------+
|                          |         |
|        Message ID        |Sub-type |
|                          |         |
+--------------------------+---------+
 81                                 96
+------------------------------------+
|                                    |
|           Message Length           |
|                                    |
+------------------------------------+
```

IMP-to-Host 1822L Leader Format
Figure 3.3

Bits 1-4: Unused and set to zero.

Bits 5-8: 1822L IMP-to-Host Flag:

This field is set to decimal 14 (1110 in binary).

Bits 9-16: Handling Type:

This has the value assigned by the source host (see section 3.1). This field is only used in message types 0, 5-9, and 15.

Bits 17-20: Unused and set to zero.

Bit 21: Trace Bit:

If equal to one, the source host designated this message for tracing as it proceeds through the network. See 1822(5.5).

Bits 22-24: Leader Flags:

Bit 22: Available as a destination host flag.

Bits 23-24: Reserved for future use, set to zero.

Bits 25-32: Message Type:

Type 0: Regular Message - All host-to-host communication occurs via regular messages, which have several sub-types. The sub-type field (bits 77-80) is the same as sent in the host-to-IMP leader (see section 3.1).

Type 1: Error in Leader - See 1822(3.4). In addition to its already defined sub-types, this message has two new |

sub-types: |

4: Illegal Leader Style - The host submitted a leader in which bits 5-8 did not contain the value 13, 14, or 15 decimal. |

5: Wrong Leader Style - The host submitted an 1822L leader when the IMP was expecting an 1822 leader, or vice-versa. |

Type 2: IMP Going Down - See 1822(3.4).

Type 3: NDM Reply - This is a reply to the NDM host-to-IMP message (see section 3.1). It will have the same number of entries as the NDM message that is being replying to, and each listed 1822L name will be accompanied by a zero or a one (see figure 3.2). A zero signifies that the name is not effective, and a one means that the name is now effective.

Type 4: NOP - The host should discard this message. It is used during initialization of the IMP/host communication. The Destination Host field will contain the 1822L Address of the host port over which the NOP is being sent. All other fields are unused.

Type 5: Ready for Next Message (RFNM) - See 1822(3.4).

Type 6: Dead Host Status - See 1822(3.4).

Type 7: Destination Host or IMP Dead (or unknown) - See 1822(3.4).

Type 8: Error in Data - See 1822(3.4).

Type 9: Incomplete Transmission - See 1822(3.4). In addition to its already defined sub-types, this message has one new sub-type: |

6: Logically Addressed Host Went Down - A logically addressed message was lost in the network because the destination host to which it was being delivered went down. The message should be resubmitted by the source host, since there may be another effective host port to which the message could be delivered (see section 2.2.3). |

Type 10: Interface Reset - See 1822(3.4).

Type 11: Name Server Reply - This reply to the Name Server Request host-to-IMP message contains, following the leader and any leader padding, a word with the selection policy and the number of physical addresses to which the destination name maps, followed by two words per physical address: the first word contains an 1822L address, and the second word contains a bit signifying whether or not that particular translation is effective and the routing distance (expected network transmission delay, in 6.4 ms units) to the address's IMP. In figure 3.4, which includes the leader without any leader padding, EFF is 1 for effective and 0 for non-effective, and POL is a two-bit number indicating the selection policy for the name (see section 2.2.2):

0: First reachable.

1: Closest physical address.

2: Load leveling.

3: Unused.

```
 1             16 17            32 33            48
+----------------+----------------+----------------+
|                |                |                |
|      0E00      |      000B      |      0000      |
|                |                |                |
+----------------+----------------+----------------+
 49            64 65            80 81            96
+----------------+----------------+----------------+
|                |                |                |
|   dest. name   |      0000      |      0000      |
|                |                |                |
+----------------+----------------+----------------+
 97           112 113          128 129          144
+-+--------------+----------------+-+--------------+
|P|              |                |E|              |
|O|  # of addrs  |  1822L addr #1 |F| routing dist |
|L|              |                |F|              |
+-+--------------+----------------+-+--------------+
145           160 161          176
+----------------+-+--------------+
|                |E|              |
| 1822L addr #2  |F| routing dist |        etc.
|                |F|              |
+----------------+-+--------------+
```

Name Server Reply Format
Figure 3.4

Type 12: Port List Reply - This is the reply to the Port List Request host-to-IMP message. It contains the number of names that map to this physical host port, followed by two words per name: the first word contains an 1822L name that maps to this port, and the second contains either a zero or a one, signifying whether or not that particular translation is effective. The format is identical to the type 3 NDM Reply message (see figure 3.2).

Type 15: 1822L Name or Address Error - This message is sent in response to a type 0 message from a host that contained an erroneous Source Host or Destination Host field. Its sub-types are:

0: The Source Host 1822L name is not authorized or not effective.

1: The Source Host 1822L address does not match the host port used to send the message.

2: The Destination Host 1822L name is not authorized.

3: The physical host to which this singly-homed Destination Host name translated is authorized and up, but not effective. If the host was actually down, a type 7 message would be returned, not a type 15.

5: The multi-homed Destination Host name is authorized, but has no available effective translations.

6: A logically-addressed uncontrolled packet was sent to a dead or non-effective host port. However, if it is resubmitted, there may be another effective host port to which the IMP may be able to attempt to send the packet.

7: Logical addressing is not in use in this network.

8-15: Unassigned.

Types 4,13-14,16-255: Unassigned.

Bits 33-48: Source Host:

For type 0 messages, this field contains the 1822L name or address of the host that originated the message. All replies to the message should be sent to the host specified herein. For message types 5-9 and 15, this field contains the source host field used in a previous type 0 message sent by this host.

Bits 49-64: Destination Host:

For type 0 messages, this field contains the 1822L name or address that the message was sent to. This allows the destination host to detect how it was specified by the source host. For message types 5-9 and 15, this field contains the destination host field used in a previous type 0 message sent by this host.

Bits 65-76: Message ID:

For message types 0, 5, 7-9, and 15, this is the value assigned by the source host to identify the message (see section 3.1). This field is also used by message types 2 and 6.

Bits 77-80: Sub-type:

This field is used as a modifier by message types 0-2, 5-7, 9, and 15.

Bits 81-96: Message Length:

This field is contained in type 0, 3, 11, and 12 messages only, and is the actual length in bits of the message (exclusive of leader, leader padding, and hardware padding) as computed by the IMP.

4 REFERENCES

[1] "Specifications for the Interconnection of a Host and an IMP", BBN Report 1822, December 1981 Revision.

[2] E. C. Rosen et. al., "ARPANET Routing Algorithm Improvements", Internet Experimenter's Note 183 (also published as BBN Report 4473, Vol. 1), August 1980, pp. 55-107.

[3] J. Reynolds and J. Postel, "Assigned Numbers", Request For Comments 870, October 1983, p. 14.

[4] J. Postel, ed., "Internet Protocol - DARPA Internet Program Protocol Specification", Request for Comments 791, September 1981.

[5] J. Postel, "Address Mappings", Request for Comments 796, September 1981.

APPENDIX A

1822L-IP ADDRESS MAPPINGS

Once logical addressing is in active (or universal) use in a network, to the extent that the "official" host tables for that network specify hosts by their logical names rather than by their physical network addresses, it would be desirable for hosts on other networks to also be able to use the same logical names to specify these hosts when sending traffic to them via the internet [4].

Happily, there exists a natural mapping between logical names and internet addresses that fits very nicely with the already standard ARPANET-style address mapping as specified in RFC 796, Address Mappings [5]. The current ARPANET-style class A mapping is as follows (from RFC 796):

```
+--------+ +--------+--------+
|  HOST  | |  ZERO  |  IMP   |     1822 Address
+--------+ +--------+--------+
    8          8        8

+--------+--------+--------+--------+
| net #  |  HOST  |   LH   |  IMP   |     IP Address
+--------+--------+--------+--------+
    8        8        8        8
```

1822 Class A Mapping
Figure A.1

For 1822L names and addresses, the mapping would be: |

```
+--------+--------+
| upper  | lower  |      1822L Name or Address
+--------+--------+
    6        6

+--------+--------+--------+--------+
| net #  | upper  |   LH   | lower  |     IP Address
+--------+--------+--------+--------+
    8        8        8        8
```

1822L Class A Mapping
Figure A.2

For 1822L addresses, this mapping is identical to the 1822 |
mapping. For 1822L names, the IP address would appear to be |
addressing a high-numbered (64-255) 1822 host. Although the LH |
(logical host) field is still defined, its use is discouraged; |
multiple logical names should now be used to multiplex multiple |

functions onto one physical host port. |

This mapping extends to class B networks: |

```
        +--------+--------+
        | upper  | lower  |      1822L Name or Address
        +--------+--------+
            8        8

        +---------------+--------+--------+
        | network number | upper  | lower  |     IP Address
        +---------------+--------+--------+
               16           8        8
```

1822L Class B Mapping
Figure A.3

Finally, logical addressing will allow IMP-based class C networks |
for the first time. Previously, it was very hard to try to |
divide the 8 bits of host specification into some number of host |
bits and some number of IMP bits. However, if ALL of the |
internet-accessible hosts on the network have logical names, |
there is no reason why networks with up to 256 such logical names |
cannot now use class C addresses, as follows: |

```
            +--------+--------+
            |01000000| lower  |     1822L Name
            +--------+--------+
                8        8

            +------------------------+--------+
            |     network number     | lower  |    IP Address
            +------------------------+--------+
                        24                8
```

1822L Class C Mapping
Figure A.4

Those hosts on the network desiring internet access would be |
assigned logical names in the range 40000 to 40377 (octal), and |
the gateway(s) connected to that network would make the |
translation from IP addresses to 1822L names as specified above. |
Note that the network could have many more than 256 hosts, or 256 |
defined names; the only restriction is that hosts that desire |
internet support or access be addressable by a name in the range |
40000 - 40377. Traffic that was strictly local to the network |
could use other names or even 1822L addresses. |

INDEX

2.2 Features of X.25 not discussed in this document are not used. For example, interrupt packets and the D bit (indicating end-to-end significance) are not used.

2.3 Negotiable features (facilities) of X.25 are allowed. For example, sites are free to negotiate larger packet and window sizes.

2.4 Some sites, such as CSNET sites, may attempt to open multiple virtual circuits to a single site. Sites should attempt to handle such incoming calls gracefully: transmit on the additional circuits if possible and accept incoming datagrams from them, but do not accept the CALL REQUEST, only to immediately close the connection or ignore datagrams transmitted on such circuits.

REFERENCE

[1] Comer, D.E. and Korb, J.T., "CSNET Protocol Software: The IP-to-X.25 Interface", SIGCOMM Symposium on Communications Architectures and Protocols, March 1983.

Network Working Group J. T. Korb
Request for Comments: 877 Purdue University
September 1983

A Standard for the Transmission of IP Datagrams Over Public Data Networks

This RFC specifies a standard adopted by CSNET, the VAN gateway, and other organizations for the transmission of IP datagrams over the X.25-based public data networks.

An X.25 virtual circuit is opened on demand when a datagram arrives at the network interface for transmission. A virtual circuit is closed after some period of inactivity (the length of the period depends on the cost associated with an open virtual circuit). A virtual circuit may also be closed if the interface runs out of virtual circuits. An algorithm for managing virtual circuits during peak demand is given in [1].

STANDARDS

1.1 The first octet in the Call User Data Field (the first data octet in the Call Request packet) is used for protocol demultiplexing. The value hex CC (binary 11001100, decimal 204) is used to mean INTERNET PROTOCOL.

1.2 IP datagrams are sent as X.25 "complete packet sequences". That is, datagrams begin on packet boundaries and the M bit ("more data") is used for datagrams that are larger than one packet. There are no additional headers or other data in the packets.

1.3 Unless a larger packet size is negotiated, the maximum size of an IP datagram transmitted over X.25 is 576 octets. If two sites negotiate a large X.25 packet size (for example, 1024 octets), an IP datagram of that size is allowed.

1.4 Either site may close a virtual circuit. If the virtual circuit is closed or reset while a datagram is being transmitted, the datagram is lost.

GENERAL REMARKS

2.1 Protocols above IP, such as TCP, do not affect this standard. In particular, no attempt is made to open X.25 virtual circuits corresponding to TCP connections.

Network Working Group
Request for Comments: 891

D.L. Mills
December 1983

DCN Local-Network Protocols

This RFC is a description of the protocol used in the DCN local networks to maintain connectivity, routing, and timekeeping information. These procedures may be of interest to designers and implementers of other networks.

1. Introduction

This document describes the local-net architecture and protocols of the Distributed Computer Network (DCN), a family of local nets based on Internet technology and an implementation of PDP11-based software called the Fuzzball. DCN local nets have been in operation for about three years and now include clones in the USA, UK, Norway and West Germany. They typically include a number of PDP11 or LSI-11 Fuzzballs, one of which is elected a gateway, and often include other Internet-compatible hosts as well.

The DCN local-net protocols are intended to provide connectivity, routing and timekeeping functions for a set of randomly connected personal computers and service hosts. The design philosophy guiding the Fuzzball implementation is to incorporate complete functionality in every host, which can serve as a packet switch, gateway and service host all at the same time. When a set of Fuzzballs are connected together using a haphazard collection of serial, parallel and contention-bus interfaces, they organize themselves into a network with routing based on minimum delay.

The purpose of this document is to describe the local-net protocols used by the DCN to maintain connectivity, routing and timekeeping functions. The document is an extensive revision and expansion of Section 4.2 of [1] and is divided into two parts, the first of which is an informal description of the architecture, together with explanatory remarks. The second part consists of a semi-formal specification of the entities and protocols used to determine connectivity, establish routing and maintain clock synchronization and is designed to aid in the implementation of cohort systems. The link-level coding is described in the appendix.

2. Narrative Description

The DCN architecture is designed for local nets of up to 256 hosts and gateways using the Internet Protocol (IP) and client protocols. It provides adaptive routing and clock synchronization functions in an arbitrary topology including point-to-point links and multipoint bus systems. It is intended for use in connecting personal computers to each other and to service machines, gateways and other hosts of the Internet community. However, it is not intended for use in large, complex networks and does not support the sophisticated routing and control algorithms of, for example, the ARPANET.

A brief description of the process and addressing structure used in the DCN may be useful in the following. A DCN physical host is a PDP11-compatible processor which supports a number of cooperating sequential processes, each of which is given a unique 8-bit identifier called its port ID. Every DCN physical host contains one or more internet processes, each of which supports a virtual host given a unique 8-bit identifier called its host ID.

Each virtual host can support multiple internet protocols, connections and, in addition, a virtual clock. Each physical host contains a physical clock which can operate at an arbitrary rate and, in addition, a 32-bit logical clock which operates at 1000 Hz and is assumed to be reset each day at 0000 hours UT. Not all physical hosts implement the full 32-bit precision; however, in such cases the resolution of the logical clock may be somewhat less.

There is a one-to-one correspondence between Internet addresses and host IDs. The host ID is formed from a specified octet of the Internet address to which is added a specified offset. The octet number and offset are selected at configuration time and must be the same for all DCN hosts sharing the local net. For class-B and class-C nets normally the fourth octet is used in this way for routing within the local net. In the case of class-B nets, the third octet is considered part of the net number by DCN hosts; therefore, this octet can be used for routing between DCN local nets. For class-A nets normally the third octet (ARPANET logical-host field) is used for routing where necessary.

Each DCN physical host is identified by a host ID for the purpose of detecting loops in routing updates, which establish the minimum-delay paths between the virtual hosts. By convention, the physical host ID is assigned as the host ID of one of its virtual hosts. A link to a neigbor net is associated with a special virtual host, called a gateway, which is assigned a unique host ID.

The links connecting the various physical hosts together and to foreign nets can be distributed in arbitrary ways, so long as the net remains fully connected. If full connectivity is lost, due to a link or host fault, the virtual hosts in each of the surviving segments can continue to operate with each other and, once connectivity is restored, with all of them.

Datagram routing is determined entirely by internet address - there is no local leader as in the ARPANET. Each physical host contains two tables, the Host Table, which is used to determine the outgoing link to each other local-net host, and the Net Table, which is used to determine the outgoing host (gateway) to each other net. The Host Table contains estimates of roundtrip delay and logical-clock offset for all virtual hosts in the net and is indexed by host ID. For the purpose of computing these estimates the delay and offset of each virtual host relative to the physical host in which it resides is assumed zero. The single exception to this is a special virtual host associated with an NBS radio time-code receiver, where the offset is computed relative to the broadcast time.

The Net Table contains an entry for every neighbor net that may be connected to the local net and, in addition, certain other nets that are not neighbors. Each entry contains the net number, as well as the host ID of the local-net gateway to that net. The routing function simply looks up the net number in the Net Table, finds the host ID of the gateway and retrieves the port ID of the net-output process from the Host Table. Other information is included in the Host Table and Net Table as discribed below.

The delay and offset estimates are updated by HELLO messages exchanged on the links connecting physical-host neighbors. The HELLO messages are exchanged frequently, but not so as to materially degrade the throughput of the link for ordinary data messages. A HELLO message contains a copy of the delay and offset information from the Host Table of the sender, as well as information to compute the roundtrip delay and logical-clock offset of the receiver relative to the sender.

The routing algorithm is similar to that (formerly) used in the ARPANET and other places in that the roundtrip (biased) delay estimate calculated to a neighbor is added to each of the delay estimates given in its HELLO message and compared with the corresponding delay estimates in the Host Table. If a delay computed in this way is less than the delay already in the Host Table, the routing to the corresponding virtual host is changed accordingly. The detailed operation of this algorithm, which includes provisions for host up-down logic and loop suppression, is summarized in a later section.

DCN local nets are self-configuring for all hosts and neighbor nets; that is, the routing algorithms will find minimum-delay paths between all hosts and gateways to neighbor nets. In addition, timekeeping information can be exchanged using special HELLO messages between neighboring DCN local nets. For routing beyond neighbor nets additional entries can be configured in the Net Tables as required. In addition, a special entry can be configured in the Net Tables which specifies the host ID of the gateway to all nets not explictly included in the table.

For routing via the ARPANET and its reachable nets a selected local-net host is equipped with an IMP interface and configured with a GGP/EGP Gateway process. This process maintains the Net Table of the local host, including ARPANET leaders, dynamically as part of the GGP/EGP protocol interactions with other ARPANET gateways. GGP/EGP protocol interactions are possibly with non-ARPANET gateways as well.

The portable virtual-host structure used in the DCN encourages a rather loose interpretation of addressing. In order to minimize confusion in the following, the term "host ID" will be applied only to virtual hosts, while "host number" will be applied to the physical host, called generically the DCN host.

2.1. Net and Host Tables

There are two tables in every DCN host which control routing of Internet Protocol (IP) datagrams: the Net Table and the Host Table. The Net Table is used to determine the host ID of the gateway on the route to a foreign net, while the Host Table is used to determine the link, with respect to the DCN host, on the route to a virtual host. The Host Table is maintained dynamically using updates generated by periodic HELLO messages. The Net Table is fixed at configuration time for all DCN hosts except those that support a GGP/EGP Gateway process. In these cases the Net Table is updated as part of the gateway operations. In addition, entries in either table can be changed by operator commands.

The Net Table format is shown in Figure 1.

```
              1                   0
5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Net Name            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Net(2)     |    Net(1)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Index      |    Net(3)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Hops       |  Gateway ID   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
|        Gateway Leader         |
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1. Net Table Entry

The "Net Name" field defines a short (RAD50) name for the net, while the "Net" fields define the class A/B/C net number. The "Gateway ID" field contains the host ID of the first gateway to the net and the "Hops" field the number of hops to it. The remaining fields are used only by the GGP/EGP Gateway process and include the "Index" field, which contains an index into the routing matrix. and the "Gateway Leader" field, which contains the (byte-swapped) local-net leader for the gateway on a neighbor net.

The Net Table contains an indefinite number of entries and is terminated by a special entry with all "Net" fields set to zero. If the "Hops" field of this entry is less than 255, the "Gateway ID" field of this entry is used for all nets not in the table. If the "Hops" field is 255 all nets not explicitly mentioned in the table appear unreachable.

The Host Table format is shown in Figure 2.

```
                  1                   0
  5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |             Name              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |      TTL      |    Port ID    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |             Delay             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Offset             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                               |
 +                               +
 |         Local Leader          |
 +                               +
 |                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                               |
 +       Update Timestamp        +
 |                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2. Host Table Entry

The ordinal position of each Host Table entry corresponds to its host ID. The "Name" field contains a short (RAD50) name for convenient reference. The "Port ID" field contains the port ID of the net-output process on the shortest path to this virtual host and the "Delay" field contains the measured roundtrip delay to it. The "Offset" field contains the difference between the logical clock of this host and the logical clock of the local host. The "Local Leader" field contains information used to construct the local leader of the outgoing packet, for those nets that require it. The "Update Timestamp" field contains the logical clock value when the entry was last updated and the "TTL" field the time (in seconds) remaining until the virtual host is declared down.

All fields except the "Name" field are filled in as part of the routing update process, which is initiated upon arrival of a HELLO message from a neighboring DCN host. This message takes the form of an IP datagram carrying the reserved protocol number 63 and a data field as shown in Figure 3.

```
                     1                   0
                 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
             --- +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Fixed            |            Checksum           |
Area             +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                 |              Date             |
                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                 |                               |
                 +              Time             +
                 |                               |
                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                 |           Timestamp           |
                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                 |    Offset     |   Hosts (n)   |
             --- +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Host             |          Delay Host 0         |
Area             +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                 |         Offset Host 0         |
                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                ...                             ...
                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                 |        Delay Host n-1         |
                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                 |        Offset Host n-1        |
             --- +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3. HELLO Message Format

There are two HELLO message formats, depending on the length of the message. One format, sent by a DCN host to another host on the same local net, includes both the fixed and host areas shown above. The second format, sent in all other cases, includes only the fixed area.

Note that all word fields shown are byte-swapped with respect to the ordinary PDP11 representation. The "Checksum" field contains a checksum covering the fields indicated. The "Date" and "Time" fields are filled in with the local date and time of origination. The "Timestamp" field is used in the computation of the roundtrip delay (see below). The "Offset" field contains the offset of the block af Internet addresses used by the local net. The "Delay Host n" and "Offset Host n" fields represent a copy of the corresponding entries of the Host Table as they exist at the time of origination. The "Hosts (n)" field contains the number of entries in this table.

2.2. Roundtrip Delay Calculations

Periodically, each DCN physical host sends a HELLO message to its neighbor on each of the communication links common to both of them. For each of these links the sender keeps a set of state variables, including a copy of the source-address field of the last HELLO message received.

When constructing a HELLO message the sender sets the destination-address field to this state variable and the source-address field to its own address. It then fills in the "Date" and "Time" fields from its logical clock and the "Timestamp" field from another state variable. It finally copies the "Delay" and "Offset" values from its Host Table into the message.

A host receiving a HELLO message discards it if the format is bad or the checksum fails. If valid, it initializes a link state variable to show that the link is up. Each time a HELLO message is transmitted this state variable is decremented. If it decrements to zero the link is declared down.

The host then checks if the source-address field matches the state variable containing the last address stored. If not, the link has been switched to a new host, so the state variables are flushed and the link forced into a recovery state. The host then checks if the destination-address field matches its own address. If so, the message has been looped (legal only in the case of a broadcast net) and the roundtrip delay information is corrected. The host and net areas are ignored in this case. If not, the host and net areas of the message are processed to update the Host and Net Tables.

Roundtrip delay calculations are performed in the following way. The link input/output processes assigned each link maintain an internal state variable which is updated as each HELLO message is received and transmitted. When a HELLO message is received this variable takes the value of the "Time" field minus the current time-of-day. When the next HELLO message is transmitted, the value assigned the "Timestamp" field is computed as the low-order 16-bits of this variable plus the current time-of-day. The value of this variable is forced to zero if either the link is down of the system logical clock has been reset since the last HELLO message was received.

If a HELLO message is received with zero "Timestamp" field, no processing other than filling in the internal state variable. Otherwise, the roundtrip delay is computed as the low-order 16-bits of the current time-of-day minus the value of this field. In order to assure the highest accuracy, the calculation is performed only if the length of the last transmitted HELLO message (in octets) matches the length of the received HELLO message.

The above technique renders the calculation independent of the clock offsets and intervals between HELLO messages at either host, protects against errors that might occur due to lost HELLO messages and works even when a neighbor host simply forwards the HELLO message back to the originator without modifying it. The latter behavior, typical of ARPANET IMPs and gateways, as well as broadcast nets, relys on the loop-detection mechanism so that correct calculations can be made and, furthermore, that spurious host updates can be avoided.

2.3. Host Updates

When a HELLO message arrives which results in a valid roundtrip delay calculation, a host update process is performed. This consists of adding the roundtrip delay to each of the "Delay Host n" entries in the HELLO message in turn and comparing each of these calculated delays to the "Host Delay" field of the corresponding Host Table entry. Each entry is then updated according to the following rules:

1. If the link connects to another DCN host on the same net and the port ID (PID) of the link output process matches the "Port ID" field of the entry, then update the entry.

2. If the link connects to another DCN host on the same net, the PID of the link output process does not match the "Port ID" field and the calculated delay is less than the "Host Delay" field by at least a specified switching threshold (currently 100 milliseconds), then update the entry.

3. If the link connects to a foreign net and is assigned a host ID corresponding to the entry, then update the entry. In this case only, use as the calculated delay the roundtrip delay.

4. If none of the above conditions are met, or if the virtual host has been declared down and the "TTL" field contains a nonzero value, then no update is performed.

The update process consists of replacing the "Delay" field with the calculated delay, the "Port ID" field with the PID of the link output process, the "Update Timestamp" field with the current time of day, and the "TTL" field by a specified value (currently 120) in seconds. If the calculated delay exceeds a specified maximum interval (currently 30 seconds), the virtual host is declared down by setting the corresponding "Delay" field to the maximum and the remaining fields as before. For the purposes of delay calculations values less than a specified minimum (currently 100 milliseconds) are rounded up to that minimum.

The "Offset" field is also replaced during the update process. When the HELLO message arrives, The value of the current logical clock is subtracted from the "Time" field and the difference added to one-half the roundtrip delay. The resulting sum, which represents the offset of the local clock to the clock of the sender, is added to the corresponding "Offset" field of the HELLO message and the sum replaces the "Offset" field of the Host Table. Thus, the "Offset" field in the Host Table for a particular virtual host is replaced only if that host is up and on the minimum-delay path to the DCN host.

The purpose of the switching threshold in (2) above and the minimum delay specification in the update process is to avoid unnecessary switching between links and transient loops which can occur due to normal variations in propagation delays. The purpose of the "TTL" field test in (4) above is to insure consistantcy by purging all paths to a virtual host when that virtual host goes down.

In addition to the updates performed as HELLO messages arrive, each virtual host in a DCN host also performs a periodic update of its own Host Table entry. The update procedure is identical to the above, except that the calculated delay and offset are taken as zero. At least one of the virtual hosts in a DCN host must have the same host ID as the host number assigned the DCN host itself and all must be assigned the same net number. Other than these, there are no restrictions on the number or addresses of internet processes resident in a single DCN host.

It should be appreciated that virtual hosts are truly portable and can migrate about the net, should such a requirement arise. The host update protocols described here insure that the routing procedures always converge to the minimum-delay paths via operational links and DCN hosts. In the case of broadcast nets such as Ethernets, the procedures are modified slightly as described below. In this case the HELLO messages are used to determine routing from the various Ethernet hosts to destinations off the cable, as well as to provide time synchronization.

2.4. Timeouts

The "TTL" field in every Host Table entry is decremented once a second in normal operation. Thus, if following a host update another update is not received within an interval corresponding to the value initialized in that field, it decrements to zero, at which point the virtual host is declared down and the Host Table entry set as described above. The 120-second interval used currently provides for at least four HELLO messages to be generated by every neighbor on every link during that interval, since the maximum delay between HELLO messages is 30 seconds on the lowest-speed link (1200 bps). Thus, if no HELLO messages are lost, the maximum number of links between any virtual host and any other is four.

The "TTL" field is initialized at 120 seconds when an update occurs and when the virtual host is declared down. During the interval this field decrements to zero immediately after being declared down, updates are ignored. This provides a decent interval for the bad news to propagate throughout the net and for the Host Tables in all DCN hosts to reflect the fact. Thus, the formation of routing loops is prevented.

The IP datagram forwarding procedures call for decrementing the "time-to-live" field in the IP header once per second or at each point where it is forwarded, whichever comes first. The value used currently for this purpose is 30, so that an IP datagram can live in the net no longer than that number of seconds. This is thus the maximum delay allowed on any path between two virtual hosts. If this maximum delay is exceeded in calculating the roundtrip delay for a Host Table entry, the corresponding virtual host will be declared down.

The interval between HELLO messages on any link depends on the data rate supported by the link. As a general rule, this interval is set at 16 times the expected roundtrip time for the longest packet to be sent on that link. For 1200-bps asynchronous transmission and packet lengths to 256 octets, this corresponds to a maximum HELLO message interval of about 30 seconds.

Although the roundtrip delay calculation, upon which the routing process depends, is relatively insensitive to net traffic and congestion, stochastic variations in the calculated values ordinarily occur due to coding (bit or character stuffing) and medium perturbations. In order to suppress loops and needless path changes a minimum switching threshold is incorporated into the routing mechanism (see above). The interval used for this threshold, as well as for the minimum delay on any path, is 100 milliseconds.

3. Formal Specification

The following sections provide a formal framework which describe the DCN HELLO protocol. This protocol is run between neighboring DCN hosts that share a common point-to-point link and provides automatic connectivity determination, routing and timekeeping functions.

The descriptions to follow are organized as follows: First a summary of data structures describes the global variables and packet formats. Then three processes which implement the protocol are described: the CLOCK, HELLO and HOST processes. The description of these processes is organized into sections that describe (1) the local variables used by that process, (2) the parameters and constants and (3) the events that initiate processing together with the procedures they evoke. In the case of variables a distinction is made between state variables, which retain their contents between procedure calls, and temporaries, which have a lifetime extending only while the process is running. Except as noted below, the initial contents of state variables are unimportant.

3.1. Data Structures

3.1.1. Global Variables

ADDRESS
This is a 32-bit bit-string temporary variable used to contain an Internet address.

CLOCK-HID
This is an eight-bit integer state variable used to contain the host ID of the local-net host to be used as the master clock. It is initialized to the appropriate value depending upon the net configuration.

DATE
This is a 16-bit bit-string state variable used to contain the date in RT-11 format. Bits 0-4 contain the year, with zero corresponding to 1972, bits 5-9 contain the day of the month and

bits 10-14 contain the month, starting with one for January.

DATE-VALID
This is a one-bit state variable used to indicate whether the local date and time are synchronized with the master clock. A value of one indicates the local clock is not synchronized with the master clock. This variable is set to one initially and when the local time-of-day rolls over past midnight. It is set to zero each time a valid date and time update has been received from the master clock.

DELAY
This is a 16-bit integer temporary variable which represents the roundtrip delay in milliseconds to a host.

HID
This is an eight-bit integer temporary variable containing the host ID of some host on the local net.

There is a one-to-one correspondence between the Internet addresses of local hosts and their HIDs. The mapping between them is selected on the basis of the octet number of the Internet address. For DCN hosts it is the fourth octet, while for hosts directly connected to a class-A ARPANET IMP or gateway, it is the third octet (logical-host field). The contents of this octet are to be added to ADDRESS-OFFSET to form the HID associated with the address.

HOLD
This is an eight-bit counter state variable indicating whether timestamps are valid or not. While HOLD is nonzero, timestamps should be considered invalid. When set to some nonzero value, the counter decrements to zero at a 1-Hz rate. Its initial value is zero.

HOST-TABLE
This is a table of NHOSTS entries indexed by host ID (HID). There is one entry for each host in the local net. Each entry has the following format:

HOST-TABLE.DELAY
This is a 16-bit field containing the computed roundtrip delay in milliseconds to host HID.

HOST-TABLE.OFFSET
This is a 16-bit field containing the computed signed offset in milliseconds which must be added to the local apparent clock to agree with the apparent clock of host HID.

HOST-TABLE.PID
This is an eight-bit field containing the PID of the net-output process selected by the routing algorithm to forward packets to host HID.

HOST-TABLE.TTL
This is an eight-bit field used as a time-to-live indicator. It is decremented by the HOST process once each second and initialized to a chosen value when a HELLO message is received. The table is initialized with the HOST-TABLE.DELAY field set to MAXDELAY for all entries. The contents of the other fields are unimportant.

LOCAL-ADDRESS
This is a 32-bit bit-string state variable used to contain the local host Internet address.

NET-TABLE
This is a table of NNETS entries with the following format:

NET-TABLE.HID
This is an eight-bit field containing the host ID of the pseudo-process to forward packets to the NET-TABLE.NET net.

NET-TABLE.NET
This is a 24-bit field containing an Internet class-A (eight bits), class-B (16 bits) or class-C (24 bits) net number. Note that the actual field width for class-B net numbers is 24 bits in order to provide a subnet capability, in which the high-order eight bits of the 16-bit host address is interpreted as the subnet number.

The table is constructed at configuration time and must include an entry for every net that is a potential neighbor. A neighbor net is defined as a net containing a host that can be directly connected to a host on the local net. The entry for such a net is initialized with NET-TABLE.NET set to the neighbor net number and NET-TABLE.HID set to an arbitrary vitual-host ID not assigned any other local-net virtual host.

The remaining entries in NET-TABLE are initialized at initial-boot time with the NET-TABLE.NET fields set to zero and the NET-TABLE.HID fields set to a configuration-selected host ID to be used to forward packets to all nets other than neighbor nets. In the case where a gateway module is included in the local host configuration, the GGP and/or EGP protocols will be used to maintian these entries; while, in the case where no gateway module is included, only one such entry is required.

OFFSET
This is a 16-bit signed integer temporary variable which represents the offset in milliseconds to be added to the apparent clock time to yield the apparent clock time of the neighbor host.

3.1.2. Parameters

ADDRESS-OFFSET
This is an integer which represents the value of the Internet address field corresponding to the first host in HOST-TABLE.

NHOSTS
    This is an integer which defines the number of entries in HOST-TABLE.

NNETS
    This is an integer which defines the number of entries in MET-TABLE.

3.1.3. HELLO Packet Fields

PKT.ADDRESS-OFFSET
    This eight-bit is copied from ADDRESS-OFFSET by the sender.

PKT.DATESTAMP
    Bits 0-14 of this 16-bit field are copied from DATE by the sender, while bit 15 is copied from DATE-VALID.

PKT.DATE-VALID
    This one-bit field is bit 15 of PKT.DATESTAMP.

PKT.DESTINATION
    This 32-bit field is part of the IP header. It is copied from HLO.NEIGHBOR-ADDRESS by the sender.

PKT.HOST-TABLE
    This is a table of PKT.NHOSTS entries, each entry of which consists of two fields. The entries are indexed by host ID and have the following format:

PKT.HOST-TABLE.DELAY
    This 16-bit field is copied from the corresponding HOST-TABLE.DELAY field by the sender.

PKT.HOST-TABLE.OFFSET
    This 16-bit field is copied from the corresponding HOST-TABLE.OFFSET field by the sender.

PKT.LENGTH
    This 16-bit field is part of the IP header. It is set by the sender to the number of octets in the packet.

PKT.NHOSTS
    This eight-bit field is copied from NHOST by the sender.

PKT.SOURCE
    This 16-bit field is part of the IP header. It is copied from LOCAL-ADDRESS by the sender.

PKT.TIMESTAMP
    This 32-bit field contains the apparent time the packet was transmitted in milliseconds past midnight UT.

PKT.TSP
This 16-bit field contains a variable used in roundtrip delay calculations.

3.2 CLOCK Process (CLK)

The timekeeping system maintains three clocks: (1) the physical clock, which is determined by a hardware oscillator/counter; (2) the apparent clock, which maintains the time-of-day used by client processes and (3) the actual clock, which represents the time-of-day provided by an outside reference. The apparent and actual clocks are maintained as 48-bit quantities with 32 bits of significance available to client processes. These clocks run at a rate of 1000 Hz and are reset at midnight UT.

The CLOCK process consists of a set of state variables along with a set of procedures that are called as the result of hardware interrupts and client requests. An interval timer is assumed logically separate from the local clock mechanism, although both could be derived from the same timing source.

3.2.1. Local Variables

CLK.CLOCK
This is a 48-bit fixed-point state variable used to represent the apparent time-of-day. The decimal point is to the right of bit 16 (numbering from the right at bit 0). Bit 16 increments at a rate equivalent to 1000 Hz independent of the hardware clock. (In the case of programmable-clock hardware the value of CLK.CLOCK must be corrected as described below.)

CLK.COUNT
This is a hardware register that increments at rate R. It can be represented by a simple line clock, which causes interrupts at the line-frequency rate, or by a programmable clock, which contains a 16-bit register that is programmed to count at a 1000-Hz rate and causes an interrupt on overflow. The register is considered a fixed-point variable with decimal point to the right of bit 0.

CLK.DELTA
This is a 48-bit signed fixed-point state variable used to represent the increment to be added to CLK.CLOCK to yield the actual time-of-day. The decimal point is to the right of bit 16.

3.2.3. Parameters

ADJUST-FRACTION
This is an integer which defines the shift count used to compute a fraction that is used as a multiplier of CLK.DELTA to correct CLK.CLOCK once each clock-adjust interval. A value of seven is suggested.

ADJUST-INTERVAL
This is an integer which defines the clock-adjust interval in milliseconds. A value of 500 (one-half second) is suggested for the line clock and 4000 (four seconds) for the 1000-Hz clock.

CLOCK-TICK
This is a fixed-point integer which defines the increment in milliseconds to be added to CLK.CLOCK as the result of a clock tick. The decimal point is to the right of bit 16. In the case of a line-clock interrupt, the value of CLOCK-TICK should be 16.66666 (60 Hz) or 20.00000 (50 Hz). In the case of a 1000-Hz programmable-clock overflow, the value should be 65536.00000.

HOLD-INTERVAL
This is an integer which defines the number of seconds that HOLD will count down after CLK.CLOCK has been reset. The resulting interval must b at least as long as the maximum HELLO-INTERVAL used by any HELLO process.

3.2.3. Events and Procedures

INCREMENT-CLOCK Event
This event is evoked as the result of a tick interrupt, in the case of a line clock, or a counter overflow, in the case of the 1000-Hz clock. It causes the logical clock to be incremented by the value of CLOCK-TICK.

1. Add the value of CLOCK-TICK to CLK.CLOCK.

ADJUST-CLOCK Event
This event is evoked once every ADJUST-INTERVAL milliseocnds to slew the apparent clock time to the actual clock time as set by the SET-CLOCK procedure. This is done by subtracting a fraction of the correction factor CLK.DELTA from the value of CLK.DELTA and adding the same fraction to CLK.CLOCK. This continues until either the next SET-CLOCK call or CLK.DELTA has been reduced to zero.

The suggested values for ADJUST-INTERVAL and ADJUST-FRACTION represent a maximum slew rate of less than +-2 milliseconds per second, in the case of 1000-Hz clock. The action is to smooth noisy clock corrections received from neighboring systems to obtain a high-quality local reference, while insuring the apparent clock time is always monotonically increasing.

1. Shift the 48-bit value of CLK.DELTA arithmetically ADJUST-FRACTION bits to the right, discarding bits from the right and saving the result in a temporary variable F. Assuming the decimal point of F to be positioned to the right of bit 16 and sign-extending as necessary, subtract F from CLK.DELTA and add F to CLK.CLOCK.

DECREMENT-HOLD Event
This event is evoked once per second to decrement the value of HOLD.

1. If the value of HOLD is zero, do nothing; otherwise, decrement its value.

READ-CLOCK Procedure

This procedure is called by a client process. It returns the apparent time-of-day computed as the integer part of the sum CLK.CLOCK plus CLK.COUNT. Note that the precision of the value returned is limited to +-1 millisecond, so that client processes must expect the apparent time to "run backward" occasionally due to drift correctins. When this happens the backward step will never be greater than one millisecond and will never occur more often than twice per second.

1. In the case of line clocks CLK.COUNT is always zero, while in the case of programmable clocks the hardware must be interrogated to extract the value of CLK.COUNT. If following interrogation a counter-overflow condition is evident, add CLOCK-TICK to CLK.CLOCK and interrogate the hardware again.

2. When the value of CLK.COUNT has been determined compute the sum CLK.COUNT + CLK.CLOCK. If this sum exceeds the number of milliseconds in 24 hours (86,400,000), reduce CLK.CLOCK by 86,400,000, set HOLD-INTERVAL -> HOLD, set CLOCK-VALID (bit 15 of DATE) to one, roll over DATE to the next calender day and start over. If not, return the integer part of the sum as the apparent time-of-day.

   The CLOCK-VALID bit is set to insure that a master-clock update is received at least once per day. Note that, in the case of uncompensated crystal oscillators of the type commonly used as the 1000-Hz time base, a drift of several parts per million can be expected, which would result in a time drift of several tenths of a second per day, if not corrected.

SET-CLOCK Procedure
This procedure is called by a client process. It sets a time-of-day correction factor in milliseconds. The argument represents a 32-bit signed fixed-point quantity with decimal point to the right of bit 0 that is to be added to CLK.CLOCK so that READ-CLOCK subsequently returns the actual time-of-day.

1. If the correction factor is in the range -2**(16-ADJUST-FRACTION) to +2**(16-ADJUST-FRACTION) - 1 (about +-128 milliseconds with the suggested value of ADJUST-FRACTION), the value of the argument replaces CLK.DELTA and the procedure is complete. If not, add the value of the sign-extended argument to CLK.CLOCK and set CLK.DELTA to zero. In addition, set HOLD-INTERVAL -> HOLD, since this represents a relatively large step-change in apparent time. The value of HOLD represents the remaining number of seconds in which timestamps should be considered invalid and is used by the HELLO process to suppress roundtrip delay calculations which might involve invalid timestamps.

3.3. HELLO Process

The HELLO process maintains clock synchronization with a neighbor HELLO process using the HELLO protocol. It also participates in the routing algorithm. There is one HELLO process and one set of local state variables for each link connecting the host to one of its neighbors.

3.3.1. Local variables

HLO.BROADCAST
This is a one-bit switch state variable. When set to zero a point-to-point link is assumed. When set ot one a broadcast (e.g. Ethernet) link is assumed.

HLO.KEEP-ALIVE
This is an eight-bit counter state variable used to indicate whether the link is up. It is initialized with a value of zero.

HLO.LENGTH
This is a 16-bit integer state variable used to record the length in octets of the last HELLO message sent.

HLO.NEIGHBOR-ADDRESS
This is a 32-bit integer state variable used to contain the neighbor host Internet address.

HLO.PID
This is an eight-bit integer state variable used to identify the net-output process associated with this HELLO process. It is initialized by the kernel when the process is created and remains unchanged thereafter.

HLO.POLL
This is a one-bit switch state variable. When set the HELLO process spontaneously sends HELLO messages. When not set the HELLO process responds to HELLO messages, but does not send them spontaneously.

HLO.TIMESTAMP
This is a 32-bit integer temporary variable used to record the time of arrival of a HELLO message.

HLO.TSP
This is a 16-bit signed integer state variable used in roundtrip delay calculations.

3.3.2. Parameters

HELLO-INTERVAL
This is an integer which defines the interval in seconds between HELLO messages. It ranges from about eight to a maximum of 30 seconds, depending on line speed.

HOLD-DOWN-INTERVAL
This is an integer which defines the interval in seconds a host will be considered up following receipt of a HELLO message indicating that host is up. A value of 120 is suggested.

KEEP-ALIVE-INTERVAL
This is an integer which defines the interval, in units of HELLO-INTERVAL, that a HELLO process will consider the link up. A value of four is suggested.

MAXDELAY
This is an integer which defines the maximum roundtrip delay in seconds on a path to any reachable host. A value of 30 is suggested.

MINDELAY
This is an integer which defines the minimum switching threshold in milliseconds below which routes will not be changed. A value of 100 is suggested.

3.3.3. Events and Procedures

INPUT-PACKET Event
When a packet arrives the net-input process first sets HLO.TIMESTAMP to the value returned by the READ-CLOCK procedure, then checks the packet for valid local leader, IP header format and checksum. If the protocol field in the IP header indicates a HELLO message, the packet is passed to the HELLO process. If any errors are found the packet is dropped.

The HELLO process first checks the packet for valid HELLO header format and checksum. If any errors are found the packet is dropped. Otherwise, it proceeds as follows:

1. If PKT.SOURCE is equal to LOCAL-ADDRESS, then the line to the neighbor host is looped. If this is a broadcast link (HLO.BROADCAST is set to one), then ignore this nicety; if not, this is considered an error and further processing is abandoned. Note that, in special configurations involving other systems (e.g. ARPANET IMPs and gateways) it may be useful to use looped HELLO to monitor connectivity. The DCN implementation provides this feature, but is not described here.

2. Set KEEP-ALIVE-INTERVAL -> HLO.KEEP-ALIVE. This indicates the maximum number of HELLO intervals the HLO.TSP field is considered valid.

3. Set PKT.TIMESTAMP - HLO.TIMESTAMP -> HLO.TSP. This is part of the roundtrip delay calculation. The value of HLO.TSP will be updated and returned to the neighbor in the next HELLO message transmitted. Next, compute the raw roundtrip delay and offset: HLO.TIMESTAMP - PKT.TSP -> DELAY and HLO.TSP + DELAY/2 -> OFFSET. Note: in the case of a broadcast link (HLO.BROADCAST set to one) set DELAY to zero.

4. Perform this step only in the case of non-broadcast links (HLO.BROADCAST set to zero). If PKT.SOURCE is not equal to HLO.NEIGHBOR-ADDRESS, then a new neighbor has appeared on this link. Set PKT.SOURCE -> HLO.NEIGHBOR ADDRESS, MAXDELAY -> DELAY and proceed to the next step. This will force the line to be declared down and result in a hold-down cycle. Otherwise, if either PKT.TSP is zero or HOLD is nonzero, then the DELAY calculation is invalid and further processing is abandoned. Note that a hold-down cycle is forced in any case if a new neigbor is recognized.

5. If processing reaches this point the DELAY and OFFSET variables can be assumed valid as well as the remaining data in the packet. First, if DELAY < MINDELAY, set MINDELAY -> DELAY. This avoids needless path switching when the difference in delays is not significant and has the effect that on low-delay links the routing algorithm degenerates to min-hop rather than min-delay. Then set HLO.PID -> PID. There are two cases:

   Case 1: PKT.NHOSTS is zero.
   This will be the case when the neighbor host has just come up or is on a different net or subnet. Set NEIGHBOR-ADDRESS -> ADDRESS and call the ROUTE procedure, which will return the host ID. Then call the UPDATE procedure. In the case of errors, do nothing but return.

   Case 2: PKT.NHOSTS is nonzero.
   This is the case when the neighbor host is on the same net or subnet. First, save the values of DELAY and OFFSET in temporary variables F and G. Then, for each value of HID from zero to NHOSTS-1 consider the corresponding PKT.HOSTS-TABLE entry and do the following: Set F + PKT.HOST-TABLE.DELAY -> DELAY and G + PKT.HOST-TABLE.OFFSET -> OFFSET and call the UPDATE procedure This completes processing.

   ROUTE Procedure
   This procedure returns the host ID in HID of the host represented by the global variable ADDRESS.

1. First, determine if the host represented by ADDRESS is on the same local net as LOCAL-ADDRESS. For the purposes of this comparison bits 0-7 and 16-31 are compared for class-A nets and bits 8-31 are compared for class-B and class-C nets. This provides for a subnet capability, where the bits 0-7 and 16-23 (class-A) or 8-15 (class-B) are used as a subnet number.

Case 1: The host is on the same net or subnet.
Extract the address field of ADDRESS, subtract ADDRESS-OFFSET and store the result in HID. If 0 <= HID < NHOSTS, the procedure completes normally; otherwise it terminates in an error condition.

Case 2: The host is not on the same net or subnet.
Search the NET-TABLE for a match of the net fields of LOCAL-ADDRESS and NET-TABLE.NET. If found set NET-TABLE.HID -> HID and return normally. If the NET-TABLE.NET field is zero, indicating the last entry in the table, set HET-TABLE.HID -> HID and return normally. Note that, in the case of hosts including GGP/EGP gateway modules, if no match is found the procedure terminates in an error condition.

UPDATE Procedure

This procedure updates the entry of HOST-TABLE indicated by HID using three global variables: DELAY, OFFSET and PID. Its purpose is to update the HOST-TABLE entry corresponding to host ID HID. In the following all references are to this entry.

1. If PID is not equal to HOST-TABLE.PID, the route to host HID is not via the net-output process associated with this HELLO process. In this case, if DELAY + MINDELAY > HOST-TABLE.DELAY, the path is longer than one already in HOST-TABLE, so the procedure does nothing.

2. This step is reached only if either the route to host HID is via the net-output process associated with this HELLO process or the newly reported path to this host is shorter by at least MINDELAY. There are two cases:

   Case 1: HOST-TABLE.DELAY < MAXDELAY.
   The existing path to host HID is up and this is a point-to-point link (HLO.BROADCAST is set to zero). If DELAY < MAXDELAY the newly reported path is also up. Proceed to the next step. Otherwise, initiate a hold-down cycle by setting MAXDELAY -> HOST-TABLE.DELAY and HOLD-DOWN-INTERVAL -> HOST-TABLE.TTL and return.

   Case 2: HOST-TABLE.DELAY >= MAXDELAY.
   The existing path to host HID is down. If DELAY < MAXDELAY and HOST-TABLE.TTL is zero, the hold-down period has expired and the newly reported path has just come up. Proceed to the next step. Otherwise simply return.

3. In this step the HOST-DELAY entry is updated. Set DELAY -> HOST-TABLE.DELAY, HOLD-DOWN-INTERVAL -> HOST-TABLE.TTL and HLO.PID -> HOST-TABLE.PID.

4. For precise timekeeping, the offset can be considered valid only if the length of the last HELLO packet transmitted is equal to the length of the last one received. Thus, if HLO.LENGTH equal to PKT.LENGTH, set OFFSET -> HOST-TABLE.OFFSET; otherwise, leave this field alone. Finally, if HID is equal to CLOCK-HID and bit 15 (the DATE-VALID bit) of DATE is zero, set PKT.DATESTAMP -> DATE and call the SET-CLOCK procedure of the CLOCK process with argument HLO TIMESTAMP.

OUTPUT-PACKET Event
This event is evoked once every HELLO-INTERVAL seconds. It determines if a HELLO message is to be transmitted, transmits it and updates state variables.

1. If HLO.KEEP-ALIVE is nonzero decrement its value.

2. If HLO.POLL is zero and HLO.KEEP-ALIVE is zero, do not send a HELLO message. If either is nonzero initialize the packet fields as follows: LOCAL-ADDRESS -> PKT.SOURCE, HLO.NEIGHBOR-ADDRESS -> PKT.DESTINATION and DATE -> PKT.DATESTAMP. Note: PKT.DESTINATION is set to zero if this is a broadcast link (HLO.BROADCAST set to one). Also, note that bit 15 of DATE is the DATE-VALID bit. If this bit is one the receiver will not update its master clock from the information in the transmitted packet. This is significant only if the sending host is on the least-delay path to the master clock. Set PKT.TIMESTAMP to the value returned from the READ-CLOCK procedure. If HLO.KEEP-ALIVE is zero or HOLD is nonzero, set PKT.TSP to zero; otherwise, set PKT.TIMESTAMP + HLO.TSP -> PKT.TSP.

3. Determine if the neighbor is on the same net or subnet. If the neighbor is on a different net set PKT.NHOSTS to zero and proceed with the next step. Otherwise, set NHOSTS -> PKT.NHOSTS and for each value of HID from zero to PKT.HOSTS-1 copy the HOST-TABLE.DELAY and HOST-TABLE.OFFSET fields of the corresponding HOST-TABLE entry in order into the packet. For each entry copied test if the HOST-TABLE.PID field matches the HLO.PID of the HELLO process. If so, a potential routing loop is possible. In this case use MAXDELAY for the delay field in the packet instead.

4. Finally, set HLO.LENGTH to the number of octets in the packet and send the packet.

3.4. HOST Process (HOS)

This process maintains the routing tables. It is activated once per second to scan HOST-TABLE and decrement the HOST-TABLE.TTL field of each entry. It also performs housekeeping functions.

3.4.1 Local variables

HOS.PID
This is an eight-bit integer used to identify the HOST process. It is initialized by the kernel when the process is created and remains unchanged thereafter.

HOS.HID
This is an eight-bit temporary variable.

3.4.2 Events and Procedures

SCAN Event
This event is evoked once each second to scan the HOST-TABLE and perform housekeeping functions.

1. For each value of a temporary variable F from zero to NHOSTS-1 do the following. Set LOCAL-ADDRESS -> ADDRESS and call the ROUTE procedure, which will return the host ID HID. If F is equal to HID, then set both DELAY and OFFSET to zero, HOS.PID -> PID and call the UPDATE procedure. This will cause all packets received with the local address to be routed to this process.

   If HOST-TABLE.TTL is zero skip this step. Otherwise, decrement HOST-TABLE.TTL by one. If the result is nonzero skip the remainder of this step. Otherwise, if HOST-TABLE.DELAY <MAXDELAY set HOLDOFF-INTERVAL -> HOST-TABLE.TTL and MAXDELAY -> HOST-TABLE.DELAY. The effect of this step is to declare a hold-down cycle when a host goes down.

4 References

1. Mills, D.L. Final Report on Internet Research. ARPA Packet Switching Program. Technical Report TSLAB 82-7, COMSAT Laboratories, December 1982.

Appendix A. Link-Level Packet Formats

A.1. Serial Links Using Program-Interrupt Interfaces

Following is a description of the frame format used on asynchronous and synchronous serial links with program-interrupt interfaces such as the DEC DLV11 and DPV11. This format provides transparency coding for all messages, including HELLO messages, but does not provide error detection or retransmission functions. It is designed to be easily implemented and compatible as far as possible with standard industry protocols.

The protocol is serial-by-bit, with the same interpretation on the order of transmission as standard asynchronous and synchronous interface devices; that is, the low-order bit of each octet is transmitted first. The data portion of the frame consists of one Internet datagram encoded according to a "character-stuffing" transparency convention:

1. The frame begins with the two-octet sequence DLE-STX, in the case of asynchronous links, or the four-octet sequence SYN-SYN-DLE-STX, in the case of synchronous links. The data portion is transmitted next, encoded as described below, followed by the two-octet sequence DLE-ETX. No checksum is transmitted or expected. If it is necessary for any reason to transmit time-fill other than in the data portion, the DEL (all ones) is used.

2. Within the data portion of the frame the transmit buffer is scanned for a DLE. Each DLE found causes the sequence DLE-DLE to be transmitted. If it is necessary for some reason for the transmitter to insert time-fill within the data portion, the sequence DLE-DEL is used.

3. While scanning the data stream within the data portion of the frame the sequence DLE-DLE is found, a single DLE is inserted in the receive buffer. If the sequence DLE-ETX is found, the buffer is passed on for processing. The sequence DLE-DEL is discarded. Any other two-octet sequence beginning with DLE and ending with other than DLE, ETX or DEL is considered a protocol error (see note below).

Note: In the case of synchronous links using program-interrupt interfaces such as the DPV11, for example, a slightly modified protocol is suggested when both ends of the link concur. These interfaces typically provide a parameter register which can be loaded with a code used both to detect the receiver synchronizing pattern and for time-fill when the transmit buffer register cannot be serviced in time for the next character.

The parameter register must be loaded with the SYN code for this protocol to work properly. However, should it be necessary to transmit time-fill, a single SYN will be transmitted, rather than the DLE-DEL sequence specified. Disruptions due to these events can be minimized by use of the following rules:

1. If the transmitter senses a time-fill condition (usually by a control bit assigned for this purpose) between frames or immediately following transmission of a DLE, the condition is ignored.

2. If the transmitter senses a time-fill condition at other times it sends the sequence DLE-CAN.

3. If the receiver finds a SYN either between frames or immediately followoing DLE, the SYN is discarded without affecting sequence decoding.

4. If the receiver finds the sequence DLE-CAN in the data portion, it discards the sequence and the immediately preceding octet.

These rules will work in cases where a single SYN has been inserted by the transmitter and even when a SYN has been inserted in the DLE-CAN sequence. If an overrun (lost data) condition is sensed at the receiver, the appropriate action is to return to the initial-synchronization state. This should also be the action if any code other than STX is found following the initial DLE. or if any code other than DLE, ETX, DEL or CAN is found following a DLE in the data portion.

A.2. Serial Links Using DDCMP Devices

Following is a description of the frame format used on DEC DDCMP links with DMA interfaces such as the DEC DMV11 and DMR11. These interfaces implement the DEC DDCMP protocol, which includes error detection and retransmission capabilities. The DDCMP frame format is as follows:

```
+-------------+-----+-----+-----+-----+-----+------+------+------+
| SYN SYN SOH |Count|Flag |Resp | Seq | Adr | CRC1 | Data | CRC2 |
+-------------+-----+-----+-----+-----+-----+------+------+------+
bits   24       14     2     8     8     8     16     ...     16
```

With respect to this diagram, each octet is transmitted starting from the leftmost octet, with the bits of each octet transmitted low-order bit first. The contents of all fields except the "Data" field are managed by the interface. The Internet datagram is placed in this field as-is, with no character or bit stuffing (the extent of this field is indicated by the interface in the "Count" field.

A.3. Serial Links Using HDLC Devices

Following is a description of the frame format used on HDLC links with program-interrupt interfaces such as the DEC DPV11.

```
          +--------+--------+--------+--------+--------+--------+
          |  Flag  |  Addr  |  Ctrl  |  Data  |  CRC   |  Flag  |
          +--------+--------+--------+--------+--------+--------+
coding    01111110 00000000 00000000 xxxxxxxx cccccccc 01111110
```

With respect to this diagram, each octet is transmitted starting from the leftmost octet, with the bits of each octet transmitted low-order bit first. The code xxxxxxxx represents the data portion and cccccccc represents the checksum. The bits between the "Flag" fields are encoded with a bit-stuffing convention in which a zero bit is stuffed following a string of five one bits. The "Addr" and "Ctrl" fields are not used and the checksum is ignored. The Internet datagram is placed in the "Data" field, which must be a multiple of eight bits in length.

A.4. ARPANET 1822 Links Using Local or Distant Host Interfaces

Following is a description of the frame format used with ARPANET 1822 Local or Distant Host interfaces. These interfaces can be used to connect a DCN host to an ARPANET IMP, Gateway or Port Expander or to connect two DCN hosts together. When used to connect a DCN host to an ARPANET IMP, Gateway or Port Expander, a 96-bit 1822 leader is prepended ahead of the Internet datagram. The coding of this leader is as described in BBN Report 1822. When used to connect two DCN hosts together, no leader is used and the frame contains only the Internet datagram.

A.5. ARPANET 1822 Links Using HDH Interfaces

Following is a description of the frame format used with ARPANET 1822 HDH interfaces. These interfaces can be used to connect a DCN host to an ARPANET IMP or Gateway or to connect two DCN hosts together. In either case, the frame format is as described in Appendix J of BBN Report 1822.

A.6. X.25 LAPB Links Using RSRE Interfaces

Following is a description of the frame format used on X.25 LAPB links with the Royal Signals and Radar Establishment interfaces. These interfaces implement the X.25 Link Access Protocol - Balanced (LAPB), also known as the frame-level protocol, using a frame format similar to that described under A.3 above. Internet datagrams are placed in the data portion of I frames and encoded with the bit-stuffing procedure described in A.3. There is no packet-level format used with these interfaces.

A.7. Ethernet Links

Following is a description of the frame format used on Ethernet links.

```
            +-----------+-----------+------+------+-----+
            | Dest Addr | Srce Addr | Type | Data | CRC |
            +-----------+-----------+------+------+-----+
bits             48          48        16     ...   32
```

With respect to this diagram, each field is transmitted starting from the leftmost field, with the bits of each field transmitted low-order bit first. The "Dest Addr" and "Srce Addr" contain 48-bit Ethernet addresses, while the "Type" field contains the assigned value for IP datagrams (0800 hex) or for

ARP datagrams (0806 hex). The Internet datagram is placed in the "Data" field and followed by the 32-bit checksum. The Address Resolution Protocol (ARP) is used to establish the mapping between Ethernet address and Internet addresses.

Network Working Group
Request for Comments: 948

Ira Winston
University of Pennsylvania
June 1985

TWO METHODS FOR THE TRANSMISSION OF IP DATAGRAMS OVER IEEE 802.3 NETWORKS

Status of this Memo

This memo describes two methods of encapsulating Internet Protocol (IP) [1] datagrams on an IEEE 802.3 network [2]. This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Introduction

The IEEE 802 project has defined a family of standards for Local Area Networks (LANs) that deals with the Physical and Data Link Layers as defined by the ISO Open System Interconnection Reference Model (ISO/OSI). Several Physical Layer standards (802.3, 802.4, and 802.5) [2, 3, 4] and one Data Link Layer Standard (802.2) [5] have been defined. The IEEE Physical Layer standards specify the ISO/OSI Physical Layer and the Media Access Control Sublayer of the ISO/OSI Data Link Layer. The 802.2 Data Link Layer standard specifies the Logical Link Control Sublayer of the ISO/OSI Data Link Layer.

The 802.3 standard is based on the Ethernet Version 2.0 standard [6]. The Ethernet Physical Layer and the 802.3 Physical Layer are compatible for all practical purposes however, the Ethernet Data Link Layer and the 802.3/802.2 Data Link Layer are incompatible.

There are many existing Ethernet network installations that transmit IP datagrams using the Ethernet compatible standard described in [7]. IEEE 802.3 Physical Layer compatible connections can be added to these networks using an an Ethernet Data Link Layer compatible method for transmitting IP datagrams without violating the 802.3 standard. Alternatively, an 802.2/802.3 Data Link Layer compatible method for transmitting IP datagrams can be used.

Ethernet Compatible Method

IEEE 802.3 networks must use 48-bit physical addresses and 10 megabit/second bandwidth in order to be Ethernet compatible.

The IEEE 802.3 packet header is identical to Ethernet packet header except for the meaning assigned to one of the fields in the header. In an Ethernet packet header this field is used as a protocol type field and in an 802.3 packet header the field is used as a length field. The maximum allowed length field value on a 10 megabit/second

802.3 network is 1500. The 802.3 standard states that packets with a length field greater than the maximum allowed length field may be ignored, discarded, or used in a private manner. Therefore, the length field can be used in a private manner as a protocol type field as long as the protocol types being used are greater than 1500. The protocol type for IP, ARP and trailer encapsulation are all greater than 1500. Using this technique, the method for transmitting IP datagrams on Ethernet networks described in [7] can be used to transmit IP datagrams on IEEE 802.3 networks in an Ethernet compatible manner.

IEEE 802.2/802.3 Compatible Method

Frame Format

IP datagrams are transmitted in standard 802.2/802.3 LLC Type 1 Unnumbered Information format with the DSAP and SSAP fields of the 802.2 header set to 96, the IEEE assigned global SAP value for IP [8]. The data field contains the IP header followed immediately by the IP data.

IEEE 802.3 packets have minimum size restrictions based on network bandwidth. When necessary, the data field should be padded (with octets of zero) to meet the 802.3 minimum frame size requirements. This padding is not part of the IP packet and is not included in the total length field of the IP header.

IEEE 802.3 packets have maximum size restrictions based on the network bandwidth. Implementations are encouraged to support full-length packets.

Gateway implementations MUST be prepared to accept full-length packets and fragment them when necessary.

Host implementations should be prepared to accept full-length packets, however hosts MUST NOT send datagrams longer than 576 octets unless they have explicit knowledge that the destination is prepared to accept them. A host may communicate its size preference in TCP based applications via the TCP Maximum Segment Size option [9].

Note: Datagrams on 802.3 networks may be longer than the general Internet default maximum packet size of 576 octets. Hosts connected to an 802.3 network should keep this in mind when sending datagrams to hosts not on the same 802.3 network. It may be appropriate to send smaller datagrams to avoid unnecessary fragmentation at intermediate gateways. Please see [9] for further information on this point.

Address Mappings

The mapping of 32-bit Internet addresses to 16-bit or 48-bit 802.3 addresses can be done in several ways. A static table could be used, or a dynamic discovery procedure could be used.

Static Table

Each host could be provided with a table of all other hosts on the local network with both their 802.3 and Internet addresses.

Dynamic Discovery

Mappings between 32-bit Internet addresses and 802.3 addresses could be accomplished through a protocol similar to the Ethernet Address Resolution Protocol (ARP) [10]. Internet addresses are assigned arbitrarily on some Internet networks. Each host's implementation must know its own Internet address and respond to 802.3 Address Resolution packets appropriately. It should also use ARP to translate Internet addresses to 802.3 addresses when needed.

Broadcast Address

The broadcast Internet address (the address on that network with a host part of all binary ones) should be mapped to the broadcast 802.3 address (of all binary ones).

The use of the ARP dynamic discovery procedure is strongly recommended.

Trailer Formats

Some versions of Unix 4.2bsd use a different encapsulation method in order to get better network performance with the VAX virtual memory architecture. Consenting systems on the same 802.3 network may use this format between themselves. Details of the trailer encapsulation method may be found in [11].

Byte Order

As described in Appendix B of the Internet Protocol specification [1], the IP datagram is transmitted over 802.2/802.3 networks as a series of 8-bit bytes.

Conclusion

The two encapsulation methods presented can be mixed on the same local area network; however, this would partition the network into two incompatible subnetworks. One host on a network could support both methods and act as a gateway between the two subnetworks; however, this would introduce a significant performance penalty and should be avoided.

The IEEE 802.2/802.3 compatible encapsulation method is preferable to the Ethernet compatible method because the IEEE 802.2 and IEEE 802.3 standards have been accepted both nationally and internationally and because the same encapsulation method could be used on other IEEE 802 Physical Layer implementations. However, there are many existing installations that are using IP on Ethernet and a controlled transition from Ethernet to IEEE 802.2/802.3 is necessary.

To this end, all new implementations should allow for a static choice of encapsulation methods and all existing implementations should be modified to provide this static choice as well. During the transition, all hosts on the same network would use the Ethernet compatible method. After 802.2/802.3 support has been added to all existing implementations, the IEEE 802.2/802.3 method would be used and the transition would be complete.

References

[1] Postel, J. "Internet Protocol". RFC-791, USC/Information Sciences Institute, September 1981.

[2] The Institute of Electronics and Electronics Engineers, Inc. "IEEE Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications". The Institute of Electronics and Electronics Engineers, Inc., New York, New York, 1985.

[3] The Institute of Electronics and Electronics Engineers, Inc. "IEEE Standards for Local Area Networks: Token-Passing Bus Access Method and Physical Layer Specifications". The Institute of Electronics and Electronics Engineers, Inc., New York, New York, 1985.

[4] The Institute of Electronics and Electronics Engineers, Inc. "IEEE Standards for Local Area Networks: Token Ring Access Method and Physical Layer Specifications". The Institute of Electronics and Electronics Engineers, Inc., New York, New York, 1985.

[5] The Institute of Electronics and Electronics Engineers, Inc. "IEEE Standards for Local Area Networks: Logical Link Control". The Institute of Electronics and Electronics Engineers, Inc., New York, New York, 1985.

[6] "The Ethernet, Physical and Data Link Layer Specifications, Version 2.0". Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, 1982.

[7] Hornig, C. "A Standard for the Transmission of IP Datagrams over Ethernet Networks". RFC-894, Symbolics Cambridge Research Center, April 1984.

[8] Reynolds, J., and Postel, J. "Assigned Numbers". RFC-943, USC/Information Sciences Institute, April 1985.

[9] Postel, J. "The TCP Maximum Segment Size Option and Related Topics". RFC-879, USC/Information Sciences Institute, November 1983.

[10] Plummer, D. "An Ethernet Address Resolution Protocol". RFC-826, Symbolics Cambridge Research Center, November 1982.

[11] Leffler, S., and Karels, M. "Trailer Encapsulations". RFC-893, University of California at Berkeley, April 1984.

Network Working Group	Charles Hornig
Request for Comments: 894	Symbolics Cambridge Research Center
April 1984

A Standard for the Transmission of IP Datagrams over Ethernet Networks

Status of this Memo

This RFC specifies a standard method of encapsulating Internet Protocol (IP) [1] datagrams on an Ethernet [2]. This RFC specifies a standard protocol for the ARPA-Internet community.

Introduction

This memo applies to the Ethernet (10-megabit/second, 48-bit addresses). The procedure for transmission of IP datagrams on the Experimental Ethernet (3-megabit/second, 8-bit addresses) is described in [3].

Frame Format

IP datagrams are transmitted in standard Ethernet frames. The type field of the Ethernet frame must contain the value hexadecimal 0800. The data field contains the IP header followed immediately by the IP data.

The minimum length of the data field of a packet sent over an Ethernet is 46 octets. If necessary, the data field should be padded (with octets of zero) to meet the Ethernet minimum frame size. This padding is not part of the IP packet and is not included in the total length field of the IP header.

The minimum length of the data field of a packet sent over an Ethernet is 1500 octets, thus the maximum length of an IP datagram sent over an Ethernet is 1500 octets. Implementations are encouraged to support full-length packets. Gateway implementations MUST be prepared to accept full-length packets and fragment them if necessary. If a system cannot receive full-length packets, it should take steps to discourage others from sending them, such as using the TCP Maximum Segment Size option [4].

Note: Datagrams on the Ethernet may be longer than the general Internet default maximum packet size of 576 octets. Hosts connected to an Ethernet should keep this in mind when sending datagrams to hosts not on the same Ethernet. It may be appropriate to send smaller datagrams to avoid unnecessary fragmentation at intermediate gateways. Please see [4] for further information on this point.

Address Mappings

The mapping of 32-bit Internet addresses to 48-bit Ethernet addresses can be done several ways. A static table could be used, or a dynamic discovery procedure could be used.

Static Table

Each host could be provided with a table of all other hosts on the local network with both their Ethernet and Internet addresses.

Dynamic Discovery

Mappings between 32-bit Internet addresses and 48-bit Ethernet addresses could be accomplished through the Address Resolution Protocol (ARP) [5]. Internet addresses are assigned arbitrarily on some Internet network. Each host's implementation must know its own Internet address and respond to Ethernet Address Resolution packets appropriately. It should also use ARP to translate Internet addresses to Ethernet addresses when needed.

Broadcast Address

The broadcast Internet address (the address on that network with a host part of all binary ones) should be mapped to the broadcast Ethernet address (of all binary ones, FF-FF-FF-FF-FF-FF hex).

The use of the ARP dynamic discovery procedure is strongly recommended.

Trailer Formats

Some versions of Unix 4.2bsd use a different encapsulation method in order to get better network performance with the VAX virtual memory architecture. Consenting systems on the same Ethernet may use this format between themselves.

No host is required to implement it, and no datagrams in this format should be sent to any host unless the sender has positive knowledge that the recipient will be able to interpret them. Details of the trailer encapsulation may be found in [6].

(Note: At the present time Unix 4.2bsd will either always use trailers or never use them (per interface), depending on a boot-time option. This is expected to be changed in the future. Unix 4.2bsd also uses a non-standard Internet broadcast address with a host part of all zeroes, this may also be changed in the future.)

Byte Order

As described in Appendix B of the Internet Protocol specification [1], the IP datagram is transmitted over the Ethernet as a series of 8-bit bytes.

References

[1] Postel, J., "Internet Protocol", RFC-791, USC/Information Sciences Institute, September 1981.

[2] "The Ethernet - A Local Area Network", Version 1.0, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, September 1980.

[3] Postel, J., "A Standard for the Transmission of IP Datagrams over Experimental Ethernet Networks", RFC-895, USC/Information Sciences Institute, April 1984.

[4] Postel, J., "The TCP Maximum Segment Size Option and Related Topics", RFC-879, USC/Information Sciences Institute, November 1983.

[5] Plummer, D., "An Ethernet Address Resolution Protocol", RFC-826, Symbolics Cambridge Research Center, November 1982.

[6] Leffler, S., and M. Karels, "Trailer Encapsulations", RFC-893, University of California at Berkeley, April 1984.

Network Working Group
Request for Comments: 895

Jon Postel
ISI
April 1984

A Standard for the Transmission of IP Datagrams over Experimental Ethernet Networks

Status of this Memo

This RFC specifies a standard method of encapsulating Internet Protocol (IP) [1] datagrams on an Experimental Ethernet [2]. This RFC specifies a standard protocol for the ARPA Internet community.

Introduction

This memo applies to the Experimental Ethernet (3-megabit/second, 8-bit addresses). The procedure for transmission of IP datagrams on the Ethernet (10-megabit/second, 48-bit addresses) is described in [3].

Frame Format

IP datagrams are transmitted in standard Experimental Ethernet frames. The type field of the Ethernet frame must contain the value 513 (1001 octal). The data field contains the IP header followed immediately by the IP data.

If necessary, the data field should be padded to meet the Experimental Ethernet minimum frame size. This padding is not part of the IP packet and is not included in the total length field of the IP header.

The maximum length of an IP datagram sent over an Experimental Ethernet is 1536 octets. Implementations are encouraged to support full-length packets. Gateway implementations MUST be prepared to accept full-length packets and fragment them if necessary. If a system cannot receive full-length packets, it should take steps to discourage others from sending them, such as using the TCP Maximum Segment Size option [4].

Note: Datagrams on the Ethernet may be longer than the general Internet default maximum packet size of 576 octets. Hosts connected to an Ethernet should keep this in mind when sending datagrams to hosts not on the same Ethernet. It may be appropriate to send smaller datagrams to avoid unnecessary fragmentation at intermediate gateways. Please see [4] for further information on this point.

Address Mappings

The mapping between 32-bit Internet addresses to 8-bit Experimental Ethernet addresses can be done several ways.

The easiest thing to do is to use the last eight bits of host number part of the Internet address as the host's address on the Experimental Ethernet. This is the recommended approach.

Broadcast Address

The broadcast Internet address (the address on that network with a host part of all binary ones) should be mapped to the broadcast Experimental Ethernet address (address zero).

Trailer Formats

Some versions of Unix 4.2bsd use a different encapsulation method in order to get better network performance with the VAX virtual memory architecture. Consenting systems on the same Ethernet may use this format between themselves.

No host is required to implement it, and no datagrams in this format should be sent to any host unless the sender has positive knowledge that the recipient will be able to interpret them. Details of the trailer encapsulation may be found in [6].

(Note: At the present time Unix 4.2bsd will either always use trailers or never use them (per interface), depending on a boot-time option. This is expected to be changed in the future. Unix 4.2bsd also uses a non-standard Internet broadcast address with a host part of all zeroes, this will also be changed in the future.)

Byte Order

As described in Appendix B of the Internet Protocol specification [1], the IP datagram is transmitted over the Ethernet as a series of 8-bit bytes.

References

[1] Postel, J., "Internet Protocol", RFC-791, USC/Information Sciences Institute, September 1981.

[2] Metcalfe, R. and D. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks", Communications of the ACM, V.19, N.7, pp 395-402, July 1976.

[3] Hornig, C., "A Standard for the Transmission of IP Datagrams over Ethernet Networks", RFC-894, Symbolics Cambridge Research Center, April 1984.

[4] Postel, J., "The TCP Maximum Segment Size Option and Related Topics", RFC-879, USC/Information Sciences Institute, November 1983.

[5] Plummer, D., "An Ethernet Address Resolution Protocol", RFC-826, Symbolics Cambridge Research Center, November 1982.

[6] Leffler, S., and M. Karels, "Trailer Encapsulations", RFC-893, University of California at Berkeley, April 1984.

Network Working Group
Request For Comments: 826

David C. Plummer
(DCP@MIT-MC)
November 1982

An Ethernet Address Resolution Protocol
-- or --
Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware

Abstract

The implementation of protocol P on a sending host S decides, through protocol P's routing mechanism, that it wants to transmit to a target host T located some place on a connected piece of 10Mbit Ethernet cable. To actually transmit the Ethernet packet a 48.bit Ethernet address must be generated. The addresses of hosts within protocol P are not always compatible with the corresponding Ethernet address (being different lengths or values). Presented here is a protocol that allows dynamic distribution of the information needed to build tables to translate an address A in protocol P's address space into a 48.bit Ethernet address.

Generalizations have been made which allow the protocol to be used for non-10Mbit Ethernet hardware. Some packet radio networks are examples of such hardware.

The protocol proposed here is the result of a great deal of discussion with several other people, most notably J. Noel Chiappa, Yogen Dalal, and James E. Kulp, and helpful comments from David Moon.

[The purpose of this RFC is to present a method of Converting Protocol Addresses (e.g., IP addresses) to Local Network Addresses (e.g., Ethernet addresses). This is a issue of general concern in the ARPA Internet community at this time. The method proposed here is presented for your consideration and comment. This is not the specification of a Internet Standard.]

Notes:

This protocol was originally designed for the DEC/Intel/Xerox 10Mbit Ethernet. It has been generalized to allow it to be used for other types of networks. Much of the discussion will be directed toward the 10Mbit Ethernet. Generalizations, where applicable, will follow the Ethernet-specific discussion.

DOD Internet Protocol will be referred to as Internet.

Numbers here are in the Ethernet standard, which is high byte first. This is the opposite of the byte addressing of machines such as PDP-11s and VAXes. Therefore, special care must be taken with the opcode field (ar$op) described below.

An agreed upon authority is needed to manage hardware name space values (see below). Until an official authority exists, requests should be submitted to

David C. Plummer
Symbolics, Inc.
243 Vassar Street
Cambridge, Massachusetts 02139

Alternatively, network mail can be sent to DCP@MIT-MC.

The Problem:

The world is a jungle in general, and the networking game contributes many animals. At nearly every layer of a network architecture there are several potential protocols that could be used. For example, at a high level, there is TELNET and SUPDUP for remote login. Somewhere below that there is a reliable byte stream protocol, which might be CHAOS protocol, DOD TCP, Xerox BSP or DECnet. Even closer to the hardware is the logical transport layer, which might be CHAOS, DOD Internet, Xerox PUP, or DECnet. The 10Mbit Ethernet allows all of these protocols (and more) to coexist on a single cable by means of a type field in the Ethernet packet header. However, the 10Mbit Ethernet requires 48.bit addresses on the physical cable, yet most protocol addresses are not 48.bits long, nor do they necessarily have any relationship to the 48.bit Ethernet address of the hardware. For example, CHAOS addresses are 16.bits, DOD Internet addresses are 32.bits, and Xerox PUP addresses are 8.bits. A protocol is needed to dynamically distribute the correspondences between a <protocol, address> pair and a 48.bit Ethernet address.

Motivation:

Use of the 10Mbit Ethernet is increasing as more manufacturers supply interfaces that conform to the specification published by DEC, Intel and Xerox. With this increasing availability, more and more software is being written for these interfaces. There are two alternatives: (1) Every implementor invents his/her own method to do some form of address resolution, or (2) every implementor uses a standard so that his/her code can be distributed to other systems without need for modification. This proposal attempts to set the standard.

Definitions:

```
Define the following for referring to the values put in the TYPE
field of the Ethernet packet header:
        ether_type$XEROX_PUP,
        ether_type$DOD_INTERNET,
        ether_type$CHAOS,
and a new one:
        ether_type$ADDRESS_RESOLUTION.
Also define the following values (to be discussed later):
        ares_op$REQUEST (= 1, high byte transmitted first) and
        ares_op$REPLY   (= 2),
and
        ares_hrd$Ethernet (= 1).
```

Packet format:

To communicate mappings from <protocol, address> pairs to 48.bit Ethernet addresses, a packet format that embodies the Address Resolution protocol is needed. The format of the packet follows.

```
    Ethernet transmission layer (not necessarily accessible to
         the user):
        48.bit: Ethernet address of destination
        48.bit: Ethernet address of sender
        16.bit: Protocol type = ether_type$ADDRESS_RESOLUTION
    Ethernet packet data:
        16.bit: (ar$hrd) Hardware address space (e.g., Ethernet,
                         Packet Radio Net.)
        16.bit: (ar$pro) Protocol address space.  For Ethernet
                         hardware, this is from the set of type
                         fields ether_typ$<protocol>.
         8.bit: (ar$hln) byte length of each hardware address
         8.bit: (ar$pln) byte length of each protocol address
        16.bit: (ar$op)  opcode (ares_op$REQUEST | ares_op$REPLY)
        nbytes: (ar$sha) Hardware address of sender of this
                         packet, n from the ar$hln field.
        mbytes: (ar$spa) Protocol address of sender of this
                         packet, m from the ar$pln field.
        nbytes: (ar$tha) Hardware address of target of this
                         packet (if known).
        mbytes: (ar$tpa) Protocol address of target.
```

Packet Generation:

As a packet is sent down through the network layers, routing determines the protocol address of the next hop for the packet and on which piece of hardware it expects to find the station with the immediate target protocol address. In the case of the 10Mbit Ethernet, address resolution is needed and some lower layer (probably the hardware driver) must consult the Address Resolution module (perhaps implemented in the Ethernet support module) to convert the <protocol type, target protocol address> pair to a 48.bit Ethernet address. The Address Resolution module tries to find this pair in a table. If it finds the pair, it gives the corresponding 48.bit Ethernet address back to the caller (hardware driver) which then transmits the packet. If it does not, it probably informs the caller that it is throwing the packet away (on the assumption the packet will be retransmitted by a higher network layer), and generates an Ethernet packet with a type field of ether_type$ADDRESS_RESOLUTION. The Address Resolution module then sets the ar$hrd field to ares_hrd$Ethernet, ar$pro to the protocol type that is being resolved, ar$hln to 6 (the number of bytes in a 48.bit Ethernet address), ar$pln to the length of an address in that protocol, ar$op to ares_op$REQUEST, ar$sha with the 48.bit ethernet address of itself, ar$spa with the protocol address of itself, and ar$tpa with the protocol address of the machine that is trying to be accessed. It does not set ar$tha to anything in particular, because it is this value that it is trying to determine. It could set ar$tha to the broadcast address for the hardware (all ones in the case of the 10Mbit Ethernet) if that makes it convenient for some aspect of the implementation. It then causes this packet to be broadcast to all stations on the Ethernet cable originally determined by the routing mechanism.

Packet Reception:

When an address resolution packet is received, the receiving Ethernet module gives the packet to the Address Resolution module which goes through an algorithm similar to the following. Negative conditionals indicate an end of processing and a discarding of the packet.

```
?Do I have the hardware type in ar$hrd?
Yes: (almost definitely)
  [optionally check the hardware length ar$hln]
  ?Do I speak the protocol in ar$pro?
  Yes:
    [optionally check the protocol length ar$pln]
    Merge_flag := false
    If the pair <protocol type, sender protocol address> is
        already in my translation table, update the sender
        hardware address field of the entry with the new
        information in the packet and set Merge_flag to true.
    ?Am I the target protocol address?
    Yes:
      If Merge_flag is false, add the triplet <protocol type,
          sender protocol address, sender hardware address> to
          the translation table.
      ?Is the opcode ares_op$REQUEST?  (NOW look at the opcode!!)
      Yes:
        Swap hardware and protocol fields, putting the local
            hardware and protocol addresses in the sender fields.
        Set the ar$op field to ares_op$REPLY
        Send the packet to the (new) target hardware address on
            the same hardware on which the request was received.
```

Notice that the <protocol type, sender protocol address, sender hardware address> triplet is merged into the table before the opcode is looked at. This is on the assumption that communcation is bidirectional; if A has some reason to talk to B, then B will probably have some reason to talk to A. Notice also that if an entry already exists for the <protocol type, sender protocol address> pair, then the new hardware address supersedes the old one. Related Issues gives some motivation for this.

Generalization: The ar$hrd and ar$hln fields allow this protocol and packet format to be used for non-10Mbit Ethernets. For the 10Mbit Ethernet <ar$hrd, ar$hln> takes on the value <1, 6>. For other hardware networks, the ar$pro field may no longer correspond to the Ethernet type field, but it should be associated with the protocol whose address resolution is being sought.

Why is it done this way??

Periodic broadcasting is definitely not desired. Imagine 100 workstations on a single Ethernet, each broadcasting address resolution information once per 10 minutes (as one possible set of parameters). This is one packet every 6 seconds. This is almost reasonable, but what use is it? The workstations aren't generally going to be talking to each other (and therefore have 100 useless entries in a table); they will be mainly talking to a mainframe, file server or bridge, but only to a small number of other workstations (for interactive conversations, for example). The protocol described in this paper distributes information as it is needed, and only once (probably) per boot of a machine.

This format does not allow for more than one resolution to be done in the same packet. This is for simplicity. If things were multiplexed the packet format would be considerably harder to digest, and much of the information could be gratuitous. Think of a bridge that talks four protocols telling a workstation all four protocol addresses, three of which the workstation will probably never use.

This format allows the packet buffer to be reused if a reply is generated; a reply has the same length as a request, and several of the fields are the same.

The value of the hardware field (ar$hrd) is taken from a list for this purpose. Currently the only defined value is for the 10Mbit Ethernet (ares_hrd$Ethernet = 1). There has been talk of using this protocol for Packet Radio Networks as well, and this will require another value as will other future hardware mediums that wish to use this protocol.

For the 10Mbit Ethernet, the value in the protocol field (ar$pro) is taken from the set ether_type$. This is a natural reuse of the assigned protocol types. Combining this with the opcode (ar$op) would effectively halve the number of protocols that can be resolved under this protocol and would make a monitor/debugger more complex (see Network Monitoring and Debugging below). It is hoped that we will never see 32768 protocols, but Murphy made some laws which don't allow us to make this assumption.

In theory, the length fields (ar$hln and ar$pln) are redundant, since the length of a protocol address should be determined by the hardware type (found in ar$hrd) and the protocol type (found in ar$pro). It is included for optional consistency checking, and for network monitoring and debugging (see below).

The opcode is to determine if this is a request (which may cause a reply) or a reply to a previous request. 16 bits for this is overkill, but a flag (field) is needed.

The sender hardware address and sender protocol address are absolutely necessary. It is these fields that get put in a translation table.

The target protocol address is necessary in the request form of the packet so that a machine can determine whether or not to enter the sender information in a table or to send a reply. It is not necessarily needed in the reply form if one assumes a reply is only provoked by a request. It is included for completeness, network monitoring, and to simplify the suggested processing algorithm described above (which does not look at the opcode until AFTER putting the sender information in a table).

The target hardware address is included for completeness and network monitoring. It has no meaning in the request form, since it is this number that the machine is requesting. Its meaning in the reply form is the address of the machine making the request. In some implementations (which do not get to look at the 14.byte ethernet header, for example) this may save some register shuffling or stack space by sending this field to the hardware driver as the hardware destination address of the packet.

There are no padding bytes between addresses. The packet data should be viewed as a byte stream in which only 3 byte pairs are defined to be words (arhrd, arpro and ar$op) which are sent most significant byte first (Ethernet/PDP-10 byte style).

Network monitoring and debugging:

The above Address Resolution protocol allows a machine to gain knowledge about the higher level protocol activity (e.g., CHAOS, Internet, PUP, DECnet) on an Ethernet cable. It can determine which Ethernet protocol type fields are in use (by value) and the protocol addresses within each protocol type. In fact, it is not necessary for the monitor to speak any of the higher level protocols involved. It goes something like this:

When a monitor receives an Address Resolution packet, it always enters the <protocol type, sender protocol address, sender hardware address> in a table. It can determine the length of the hardware and protocol address from the ar$hln and ar$pln fields of the packet. If the opcode is a REPLY the monitor can then throw the packet away. If the opcode is a REQUEST and the target protocol address matches the protocol address of the monitor, the monitor sends a REPLY as it normally would. The monitor will only get one mapping this way, since the REPLY to the REQUEST will be sent directly to the requesting host. The monitor could try sending its own REQUEST, but this could get two monitors into a REQUEST sending loop, and care must be taken.

Because the protocol and opcode are not combined into one field, the monitor does not need to know which request opcode is associated with which reply opcode for the same higher level protocol. The length fields should also give enough information to enable it to "parse" a protocol addresses, although it has no knowledge of what the protocol addresses mean.

A working implementation of the Address Resolution protocol can also be used to debug a non-working implementation. Presumably a hardware driver will successfully broadcast a packet with Ethernet type field of ether_type$ADDRESS_RESOLUTION. The format of the packet may not be totally correct, because initial implementations may have bugs, and table management may be slightly tricky. Because requests are broadcast a monitor will receive the packet and can display it for debugging if desired.

An Example:

Let there exist machines X and Y that are on the same 10Mbit
Ethernet cable. They have Ethernet address EA(X) and EA(Y) and
DOD Internet addresses IPA(X) and IPA(Y) . Let the Ethernet type
of Internet be ET(IP). Machine X has just been started, and
sooner or later wants to send an Internet packet to machine Y on
the same cable. X knows that it wants to send to IPA(Y) and
tells the hardware driver (here an Ethernet driver) IPA(Y). The
driver consults the Address Resolution module to convert <ET(IP),
IPA(Y)> into a 48.bit Ethernet address, but because X was just
started, it does not have this information. It throws the
Internet packet away and instead creates an ADDRESS RESOLUTION
packet with

```
        (ar$hrd) = ares_hrd$Ethernet
        (ar$pro) = ET(IP)
        (ar$hln) = length(EA(X))
        (ar$pln) = length(IPA(X))
        (ar$op)  = ares_op$REQUEST
        (ar$sha) = EA(X)
        (ar$spa) = IPA(X)
        (ar$tha) = don't care
        (ar$tpa) = IPA(Y)
```

and broadcasts this packet to everybody on the cable.

Machine Y gets this packet, and determines that it understands
the hardware type (Ethernet), that it speaks the indicated
protocol (Internet) and that the packet is for it
((ar$tpa)=IPA(Y)). It enters (probably replacing any existing
entry) the information that <ET(IP), IPA(X)> maps to EA(X). It
then notices that it is a request, so it swaps fields, putting
EA(Y) in the new sender Ethernet address field (ar$sha), sets the
opcode to reply, and sends the packet directly (not broadcast) to
EA(X). At this point Y knows how to send to X, but X still
doesn't know how to send to Y.

Machine X gets the reply packet from Y, forms the map from
<ET(IP), IPA(Y)> to EA(Y), notices the packet is a reply and
throws it away. The next time X's Internet module tries to send
a packet to Y on the Ethernet, the translation will succeed, and
the packet will (hopefully) arrive. If Y's Internet module then
wants to talk to X, this will also succeed since Y has remembered
the information from X's request for Address Resolution.

```
Related issue:
---------------

It may be desirable to have table aging and/or timeouts.  The
implementation of these is outside the scope of this protocol.
Here is a more detailed description (thanks to MOON@SCRC@MIT-MC).

If a host moves, any connections initiated by that host will
work, assuming its own address resolution table is cleared when
it moves.  However, connections initiated to it by other hosts
will have no particular reason to know to discard their old
address.  However, 48.bit Ethernet addresses are supposed to be
unique and fixed for all time, so they shouldn't change.  A host
could "move" if a host name (and address in some other protocol)
were reassigned to a different physical piece of hardware.  Also,
as we know from experience, there is always the danger of
incorrect routing information accidentally getting transmitted
through hardware or software error; it should not be allowed to
persist forever.  Perhaps failure to initiate a connection should
inform the Address Resolution module to delete the information on
the basis that the host is not reachable, possibly because it is
down or the old translation is no longer valid.  Or perhaps
receiving of a packet from a host should reset a timeout in the
address resolution entry used for transmitting packets to that
host; if no packets are received from a host for a suitable
length of time, the address resolution entry is forgotten.  This
may cause extra overhead to scan the table for each incoming
packet.  Perhaps a hash or index can make this faster.

The suggested algorithm for receiving address resolution packets
tries to lessen the time it takes for recovery if a host does
move.  Recall that if the <protocol type, sender protocol
address> is already in the translation table, then the sender
hardware address supersedes the existing entry.  Therefore, on a
perfect Ethernet where a broadcast REQUEST reaches all stations
on the cable, each station will be get the new hardware address.

Another alternative is to have a daemon perform the timeouts.
After a suitable time, the daemon considers removing an entry.
It first sends (with a small number of retransmissions if needed)
an address resolution packet with opcode REQUEST directly to the
Ethernet address in the table.  If a REPLY is not seen in a short
amount of time, the entry is deleted.  The request is sent
directly so as not to bother every station on the Ethernet.  Just
forgetting entries will likely cause useful information to be
forgotten, which must be regained.

Since hosts don't transmit information about anyone other than
themselves, rebooting a host will cause its address mapping table
to be up to date.  Bad information can't persist forever by being
passed around from machine to machine; the only bad information
that can exist is in a machine that doesn't know that some other
machine has changed its 48.bit Ethernet address.  Perhaps
manually resetting (or clearing) the address mapping table will
suffice.

This issue clearly needs more thought if it is believed to be
important.  It is caused by any address resolution-like protocol.
```

Network Working Group
Request for Comments: 903

Finlayson, Mann, Mogul, Theimer
Stanford University
June 1984

A Reverse Address Resolution Protocol

Ross Finlayson, Timothy Mann, Jeffrey Mogul, Marvin Theimer
Computer Science Department
Stanford University
June 1984

Status of this Memo

This RFC suggests a method for workstations to dynamically find their protocol address (e.g., their Internet Address), when they know only their hardware address (e.g., their attached physical network address).

This RFC specifies a proposed protocol for the ARPA Internet community, and requests discussion and suggestions for improvements.

I. Introduction

Network hosts such as diskless workstations frequently do not know their protocol addresses when booted; they often know only their hardware interface addresses. To communicate using higher-level protocols like IP, they must discover their protocol address from some external source. Our problem is that there is no standard mechanism for doing so.

Plummer's "Address Resolution Protocol" (ARP) [1] is designed to solve a complementary problem, resolving a host's hardware address given its protocol address. This RFC proposes a "Reverse Address Resolution Protocol" (RARP). As with ARP, we assume a broadcast medium, such as Ethernet.

II. Design Considerations

The following considerations guided our design of the RARP protocol.

A. ARP and RARP are different operations. ARP assumes that every host knows the mapping between its own hardware address and protocol address(es). Information gathered about other hosts is accumulated in a small cache. All hosts are equal in status; there is no distinction between clients and servers.

On the other hand, RARP requires one or more server hosts to maintain a database of mappings from hardware address to protocol address and respond to requests from client hosts.

B. As mentioned, RARP requires that server hosts maintain large databases. It is undesirable and in some cases impossible to maintain such a database in the kernel of a host's operating system. Thus, most implementations will require some form of interaction with a program outside the kernel.

C. Ease of implementation and minimal impact on existing host software are important. It would be a mistake to design a protocol that required modifications to every host's software, whether or not it intended to participate.

D. It is desirable to allow for the possibility of sharing code with existing software, to minimize overhead and development costs.

III. The Proposed Protocol

We propose that RARP be specified as a separate protocol at the data-link level. For example, if the medium used is Ethernet, then RARP packets will have an Ethertype (still to be assigned) different from that of ARP. This recognizes that ARP and RARP are two fundamentally different operations, not supported equally by all hosts. The impact on existing systems is minimized; existing ARP servers will not be confused by RARP packets. It makes RARP a general facility that can be used for mapping hardware addresses to any higher level protocol address.

This approach provides the simplest implementation for RARP client hosts, but also provides the most difficulties for RARP server hosts. However, these difficulties should not be insurmountable, as is shown in Appendix A, where we sketch two possible implementations for 4.2BSD Unix.

RARP uses the same packet format that is used by ARP, namely:

```
ar$hrd (hardware address space) -  16 bits
ar$pro (protocol address space) -  16 bits
ar$hln (hardware address length) - 8 bits
ar$pln (protocol address length) - 8 bits
ar$op  (opcode) - 16 bits
ar$sha (source hardware address) - n bytes,
                                   where n is from the ar$hln field.
ar$spa (source protocol address) - m bytes,
                                   where m is from the ar$pln field.
ar$tha (target hardware address) - n bytes
ar$tpa (target protocol address) - m bytes
```

arhrd, arpro, ar$hln and ar$pln are the same as in regular ARP (see [1]).

Suppose, for example, that 'hardware' addresses are 48-bit Ethernet addresses, and 'protocol' addresses are 32-bit Internet Addresses. That is, we wish to determine Internet Addresses corresponding to known Ethernet addresses. Then, in each RARP packet, ar$hrd = 1 (Ethernet), ar$pro = 2048 decimal (the Ethertype of IP packets), ar$hln = 6, and ar$pln = 4.

There are two opcodes: 3 ('request reverse') and 4 ('reply reverse'). An opcode of 1 or 2 has the same meaning as in [1]; packets with such opcodes may be passed on to regular ARP code. A packet with any other opcode is undefined. As in ARP, there are no "not found" or "error" packets, since many RARP servers are free to respond to a request. The sender of a RARP request packet should timeout if it does not receive a reply for this request within a reasonable amount of time.

The arsha, arspa, $ar$tha, and ar$tpa fields of the RARP packet are interpreted as follows:

When the opcode is 3 ('request reverse'):

ar$sha is the hardware address of the sender of the packet.

ar$spa is undefined.

ar$tha is the 'target' hardware address.

In the case where the sender wishes to determine his own protocol address, this, like ar$sha, will be the hardware address of the sender.

ar$tpa is undefined.

When the opcode is 4 ('reply reverse'):

ar$sha is the hardware address of the responder (the sender of the reply packet).

ar$spa is the protocol address of the responder (see the note below).

ar$tha is the hardware address of the target, and should be the same as that which was given in the request.

ar$tpa is the protocol address of the target, that is, the desired address.

Note that the requirement that ar$spa in opcode 4 packets be filled in with the responder's protocol is purely for convenience. For instance, if a system were to use both ARP and RARP, then the inclusion of the valid protocol-hardware address pair (arspa, arsha) may eliminate the need for a subsequent ARP request.

IV. References

[1] Plummer, D., "An Ethernet Address Resolution Protocol", RFC 826, MIT-LCS, November 1982.

Appendix A. Two Example Implementations for 4.2BSD Unix

The following implementation sketches outline two different approaches to implementing a RARP server under 4.2BSD.

A. Provide access to data-link level packets outside the kernel. The RARP server is implemented completely outside the kernel and interacts with the kernel only to receive and send RARP packets. The kernel has to be modified to provide the appropriate access for these packets; currently the 4.2 kernel allows access only to IP packets. One existing mechanism that provides this capability is the CMU "packet-filter" pseudo driver. This has been used successfully at CMU and Stanford to implement similar sorts of "user-level" network servers.

B. Maintain a cache of database entries inside the kernel. The full RARP server database is maintained outside the kernel by a user process. The RARP server itself is implemented directly in the kernel and employs a small cache of database entries for its responses. This cache could be the same as is used for forward ARP.

The cache gets filled from the actual RARP database by means of two new ioctls. (These are like SIOCIFADDR, in that they are not really associated with a specific socket.) One means: "sleep until there is a translation to be done, then pass the request out to the user process"; the other means: "enter this translation into the kernel table". Thus, when the kernel can't find an entry in the cache, it puts the request on a (global) queue and then does a wakeup(). The implementation of the first ioctl is to sleep() and then pull the first item off of this queue and return it to the user process. Since the kernel can't wait around at interrupt level until the user process replies, it can either give up (and assume that the requesting host will retransmit the request packet after a second) or if the second ioctl passes a copy of the request back into the kernel, formulate and send a response at that time.

RFC 907

HOST ACCESS PROTOCOL SPECIFICATION

July 1984

prepared for

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Bolt Beranek and Newman Laboratories
10 Moulton Street
Cambridge, Massachusetts 02238

Table of Contents

FIGURES

RFC 907 Host Access Protocol
July 1984 Specification

Preface (Status of this Memo)

This document specifies the Host Access Protocol (HAP). Although HAP was originally designed as the network-access level protocol for the DARPA/DCA sponsored Wideband Packet Satellite Network, it is intended that it evolve into a standard interface between hosts and packet-switched satellite networks such as SATNET and T'CNET (aka MATNET) as well as the Wideband Network. The HAP specification presented here is a minor revision of, and supercedes, the specification presented in Chapter 4 of BBN Report No. 4469, the "PSAT Technical Report". As such, the details of the current specification are still most closely matched to the characteristics if the Wideband Satellite Network. Revisions to the specification in the "PSAT Technical Report" include the definition of three new control message types (Loopback Request, Link Going Down, and NOP), a "Reason" field in Restart Request control messages, new Unnumbered Response codes, and new values for the setup codes used to manage streams and groups.

HAP is an experimental protocol, and will undergo further revision as new capabilities are added and/or different satellite networks are supported. Implementations of HAP should be performed in coordination with satellite network development and operations personnel.

1 Introduction

The Host Access Protocol (HAP) specifies the network-access level communication between an arbitrary computer, called a host, and a packet-switched satellite network. The satellite network provides message delivery services for geographically separated hosts: Messages containing data which are meaningful to the hosts are submitted to the network by an originating (source) host, and are passed transparently through the network to an indicated destination host. To utilize such services, a host interfaces to the satellite network via an access link to a dedicated packet-switching computer, known as a Satellite Interface Message Processor (Satellite IMP or SIMP). HAP defines the different types of control messages and (host-to-host) data messages that may be exchanged over the access link connecting a host and a SIMP. The protocol establishes formats for these messages, and describes procedures for determining when each type of message should be transmitted and what it means when one is received.

The term "Interface Message Processor" originates in the ARPANET, where it refers to the ARPANET's packet-switching nodes. SIMPs differ from ARPANET IMPs in that SIMPs form a network via connections to a common multiaccess/broadcast satellite channel, whereas ARPANET IMPs are interconnected by dedicated point-to-point terrestrial communications lines. This fundamental difference between satellite-based and ARPANET-style networks results in different mechanisms for the delivery of messages from source to destination hosts and for internal network coordination. Additionally, satellite networks tend to offer different type of service options to their connected hosts than do ARPANET-style networks. These options are included in the Host Access Protocol presented here.

Several types of Satellite IMPs have been developed on a variety of processors for the support of three different packet-switched satellite networks. The original SIMP was employed in the Atlantic Packet Satellite Network (SATNET). It was developed from one of the models of ARPANET IMP, and was implemented on a Honeywell 316 minicomputer. The 316 SIMPs were succeeded in SATNET by SIMPs based on BBN C/30 Communications Processor hardware. The C/30 SIMPs have also been employed in the Mobile

Access Terminal Network (MATNET). The SATNET and MATNET SIMPs implement a network-access level protocol known as Host/SATNET Protocol. Host/SATNET Protocol is the precursor to HAP and is documented in Internet Experiment Note (IEN) No. 192. The Wideband Satellite Network, like SATNET, has undergone an evolution in the development of its SIMP hardware and software. The original Wideband Network SIMP is known as the Pluribus Satellite IMP, or PSAT, having been implemented on the BBN Pluribus Multiprocessor. Its successor, the BSAT, is based on the BBN Butterfly Multiprocessor. Both the PSAT and the BSAT communicate with their connected network hosts via HAP.

Section 2 presents an overview of HAP. Details of HAP formats and message exchange procedures are contained in Sections 3 through 10. Further explanation of many of the topics addressed in this HAP specification can be found in BBN Report No. 4469, the "PSAT Technical Report".

The protocol used to provide sufficiently reliable message exchange over the host-SIMP link is assumed to be transparent to the network-access protocol defined in this document. Examples of such link-level protocols are ARPANET 1822 local and distant host, ARPANET VDH protocol, and HDLC.

2 Overview

HAP can be characterized as a full duplex nonreliable protocol with an optional flow control mechanism. HAP messages flow simultaneously in both directions between the SIMP and the host. Transmission is nonreliable in the sense that the protocol does not provide any guarantee of error-free sequenced delivery. To the extent that this functionality is required on the access link (e.g., non-collocated SIMP and host operating over a communication circuit), it must be supported by the link-level protocol below HAP. The flow control mechanism operates independently in each direction except that enabling or disabling the mechanism applies to both sides of the interface.

HAP supports host-to-host communication in two modes corresponding to the two types of HAP data messages, datagram messages and stream messages. Each type of message can be up to approximately 16K bits in length. Datagram messages provide the basic transmission service in the satellite network. Datagram messages transmitted by a host experience a nominal two satellite hop end-to-end network delay. (Note that this delay, of about 0.6 sec excluding access link delay, is associated with datagram transmission between hosts on different SIMPs. The transmission delay between hosts on the same SIMP will be much smaller assuming the destination is not a group address. See Section 3 and 6.2.) A datagram control header, passed to the SIMP by the host along with message text, determines the processing of the message within the satellite network independent of any previous exchanges.

Stream messages provide a one satellite hop delay (approximately 0.3 sec) for volatile traffic, such as speech, which cannot tolerate the delay associated with datagram transmission. Hosts may also use streams to support high duty cycle applications which require guaranteed channel bandwidth. Host streams are established by a setup message exchange between the host and the network prior to the commencement of data flow. Although established host streams can have their characteristics modified by subsequent setup messages while they are in use, the fixed allocation properties of streams relative to datagrams impose rather strict requirements on the source of the traffic using the stream. Stream traffic arrivals must match the stream allocation both in interarrival time and message size if reasonable efficiency is to be achieved. The characteristics and use of datagrams and streams are described in detail in Sections 3 and 4 of this document.

Both datagram and stream transmission in the satellite network use logical addressing. Each host on the network is assigned a permanent 16-bit logical address which is independent of the physical port on the SIMP to which it is attached. These 16-bit logical addresses are provided in all Host-to-SIMP and SIMP-to-Host data messages.

Hosts may also be members of groups. Group addressing is provided primarily to support the multi-destination delivery required for conferencing applications. Like streams, group addresses are dynamically created and deleted by the use of setup messages exchanged between a host and the network. Membership in a group may consist of an arbitrary subset of all the permanent network hosts. A message addressed to a group address is delivered to all hosts that are members of that group.

Although HAP does not guarantee error-free delivery, error control is an important aspect of the protocol design. HAP error control is concerned with both local transfers between a host and its local SIMP and transfers from SIMP-to-SIMP over the satellite channel. The SIMP offers users a choice of network error protection options based on the network's ability to selectively send messages over the satellite channel at different coding rates. These forward error correction (FEC) options are referred to as reliability levels. Three reliability levels (low, medium, and high) are available to the host.

In addition to forward error correction, a number of checksum mechanisms are employed in the satellite network to add an error detection capability. A host has an opportunity when sending a message to indicate whether the message should be delivered to its destination or discarded if a data error is detected by the network. Each message received by a host from the network will have a flag indicating whether or not an error was detected in that particular message. A host can decide on a per-message basis whether or not it wants to accept or discard transmissions containing data errors.

For connection of a host and SIMP in close proximity, error rates due to external noise or hardware failures on the access circuit may reasonably be expected to be much smaller than the best satellite channel error rate. Thus for this case, little is gained by using error detection and retransmission on the access circuit. A 16-bit header checksum is provided, however, to insure that SIMPs do not act on incorrect control information. For relatively long distances or noisy connections, retransmissions over the access circuit may be required to optimize performance for both low and high reliability traffic. It is expected that link-level error control procedures (such as HDLC) will be used for this purpose.

Datagram and stream messages being presented to the network by a host may not be accepted for a number of reasons: priority too low, destination dead, lack of buffers in the source SIMP, etc. The host faces a similar situation with respect to handling messages from the SIMP. To permit the receiver of a message to inform the sender of the local disposition of its message, an acceptance/refusal (A/R) mechanism is implemented. The mechanism is the external manifestation of the SIMP's (or host's) internal flow and congestion control algorithm. If A/Rs are enabled, an explicit or implicit acceptance or refusal for each message is returned to the host by the SIMP (and conversely). This allows the host (or SIMP) to retry refused messages at its discretion and can provide information useful for optimizing the sending of subsequent messages if the reason for refusals is also provided. The A/R mechanism can be disabled to provide a "pure discard" interface.

Each message submitted to the SIMP by a host is marked as being in one of four priority classes, from priority 3 (highest) through priority 0 (lowest). The priority class is used by the SIMP for arbitrating contention for scarce network resources (e.g., channel time). That is, if the network cannot deliver all of the offered messages, high priority messages will be delivered in preference to low priority messages. In the case of datagrams, priority level is used by the SIMP for ordering satellite channel reservation requests at the source SIMP and message delivery at the destination SIMP. In the case of streams, priority is associated with the ability of one stream to preempt another stream of lower priority at setup time.

While the A/R mechanism allows control of individual message transfers, it does not facilitate regulation of priority flows. Such regulation is handled by passing advisory status information (GOPRI) across the Host-SIMP interface indicating which priorities are currently being accepted. As long as this information, relative to the change in priority status, is passed frequently, the sender can avoid originating messages which are sure to be refused.

HAP defines both data messages (datagram messages and stream messages) and control messages. Data messages are used to send information between network hosts. Control messages are exchanged between a host and the network to manage the local access link. HAP can also be viewed in terms of two distinct protocol layers, the message layer and the setup layer. The message layer is associated with the transmission of individual datagram messages and stream messages. The setup layer protocol is associated with the establishment, modification, and deletion of streams and groups. Setup layer exchanges are actually implemented as datagrams transmitted between the user host and an internal SIMP "service host."

Every HAP message consists of an integral number of 16-bit words. The first several words of the message always contain control information and are referred to as the message header. The first word of the message header identifies the type of message which follows. The second word of the message header is a checksum which covers all header information. Any message whose received header checksum does not match the checksum computed on the received header information must be discarded. The format of the rest of the header depends on the specific message type.

The formats and use of the individual message types are detailed in the following sections. A common format description is used for this purpose. Words in a message are numbered starting at zero (i.e., zero is the first word of a message header). Bits within a word are numbered from zero (least significant) to fifteen (most significant). The notation used to identify a particular field location is:

```
<WORD#>{-<WORD#>}  [ <BIT#>{-<BIT#>} ]  <description>
```

where optional elements in {} are used to specify the (inclusive) upper limit of a range. The reader should refer to these field identifiers for precise field size specifications. Fields which are common to several message types are defined in the first section which uses them. Only the name of the field will usually appear in the descriptions in subsequent sections.

Link-level protocols used to support HAP can differ in the order in which they transmit the bits constituting HAP messages. For HDLC and ARPANET VDH, each word of a HAP message is transmitted starting with the least significant bit (bit 0) and ending with the most significant bit (bit 15). The words of the message are transmitted from word 0 to word N. For ARPANET 1822 local and distant host interfaces, the order of bit transmission within each word is the reverse of that for HDLC and VDH, i.e., the transmission is from bit 15 to bit 0.

3 Datagram Messages

Datagram messages are one of the two types of message level data messages used to support host-to-host communication. Each datagram can contain up to 16,384 bits of user data. Datagram messages transmitted by a host to a host on a remote SIMP experience a nominal two satellite hop end-to-end network delay (about 0.6 sec), excluding delay on the access links. This network delay is due to the reservation per message scheduling procedure for datagrams which only allocates channel time to the message for the duration of the actual transfer. Since datagram transfers between permanent hosts on the same SIMP do not require satellite channel scheduling prior to data transmission, the network delay in this case will be much smaller and is determined strictly by SIMP processing time. Datagrams sent to group addresses are treated as if they were addressed to remote hosts and are always sent over the satellite channel. It is expected that datagram messages will be used to support the majority of computer-to-computer and terminal-to-computer traffic which is bursty in nature.

The format of datagram messages and the purpose of each of the header control fields is described in Figure 1.

```
              15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      0      | 0|LB|GOPRI|   XXXX   | F|     MESSAGE NUMBER    |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      1      |                HEADER CHECKSUM                |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      2      |                      A/R                      |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      3      | 0|IL| D| E| TTL  | PRI  | RLY |     RLEN       |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      4      |            DESTINATION HOST ADDRESS           |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      5      |              SOURCE HOST ADDRESS              |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     6-N     |                     DATA                      |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 1 . DATAGRAM MESSAGE

0[15] Message Class. This bit identifies the message as a data message or a control message.

0 = Data Message
1 = Control Message

0[14] Loopback Bit. This bit allows the sender of a message to determine if its own messages are being looped back. The host and the SIMP each use different settings of this bit for their transmissions. If a message arrives with the loopback bit set equal to its outgoing value, then the message has been looped.

0 = Sent by Host
1 = Sent by SIMP

0[12-13] Go-Priority. In SIMP-to-Host messages, this field provides advisory information concerning the lowest priority currently being accepted by the SIMP. The host may optionally choose to provide similar priority information to the SIMP.

0 = Low Priority
1 = Medium-Low Priority
2 = Medium-High Priority
3 = High Priority

0[9-11] Reserved.

0[8] Force Channel Transmission Flag. This flag can be set by the source host to force the SIMP to transmit the message over the satellite channel even if the message contains permanent destination and source host addresses corresponding to hosts which are physically connected to the same SIMP.

0 = Normal operation
1 = Force channel transmission

0[0-7] Message Number. This field contains the identification of the message used by the acceptance/refusal (A/R) mechanism (when enabled). If the message number is zero, A/R is disabled for this specific message. See Section 5 for a detailed description of the A/R mechanism.

1[0-15] Header Checksum. This field contains a checksum which covers words 0-5. It is computed as the negation of the 2's-complement sum of words 0-5 (excluding the checksum word itself).

2[0-15] Piggybacked A/R. This field may contain an acceptance/refusal word providing A/R status on traffic flowing in the opposite direction. Its inclusion may eliminate the need for a separate A/R control message (see Section 5). A value of zero for this word is used to indicate that no piggybacked A/R information is present.

3[15] Data Message Type. This bit identifies whether the message is a datagram message or a stream message.

0 = Datagram Message
1 = Stream Message

3[14] Internet/Local Flag. This flag is set by a source host to specify to a destination host whether the data portion of the message contains a standard DoD Internet header. This field is passed transparently by the source and destination SIMPs for traffic between external satellite network hosts. This field is examined by internal SIMP hosts (e.g., the network service host) in order to support Internet operation.

0 = Internet
1 = Local

3[13] Discard Flag. This flag allows a source host to instruct the satellite network (including the destination host) what to do with the message when data errors are detected (assuming the header checksum is correct).

0 = Discard message if data errors detected.
1 = Don't discard message if data errors detected.

The value of this flag, set by the source host, is passed on to the destination host.

3[12] Data Error Flag. This flag is used in conjunction with the Discard Flag to indicate to the destination host whether any data errors have been detected in the message prior to transmission over the SIMP-to-Host access link. It is used only if Discard Flag = 1. It should be set to zero by the source host.

0 = No Data Errors Detected
1 = Data Errors Detected

3[10-11] Time-to-Live Designator. The source host uses this field to specify the maximum time that a message should be allowed to exist within the satellite network before being deleted. Messages may be discarded by the network prior to this maximum elapsed time.

0 = 1 seconds
1 = 2 seconds
2 = 5 seconds
3 = 10 seconds

The Time-to-Live field is undefined in messages sent from a SIMP to a host.

3[8-9] Priority. The source host uses this field to specify the priority with which the message should be handled within the network.

0 = Low Priority
1 = Medium-Low Priority
2 = Medium-High Priority
3 = High Priority

The priority of each message is passed to the destination host by the destination SIMP.

3[6-7] Reliability. The source host uses this field to specify the basic bit error rate requirement for the data portion of this message. The source SIMP uses this field to determine the satellite channel transmission parameters required to provide that bit error rate.

0 = Low Reliability
1 = Medium Reliability

2 = High Reliability
3 = Reserved

The Reliability field is undefined in messages sent from a SIMP to a host.

3[0-5]	Reliability Length. This source host uses this field to specify a portion of the user data which should be transmitted at the highest reliability level (lowest bit error rate). Both the six message header words and the first Reliability Length words of user data will be transmitted at Reliability=2 while the remainder of the user data will be transmitted at whatever reliability level is specified in field 3[6-7]. The reliability length mechanism gives the user the ability to transmit private header information (e.g., IP and TCP headers) at a higher reliability level than the remainder of the data. The Reliability Length field is undefined in messages sent from a SIMP to a host.
4[0-15]	Destination Host Address. This field contains the satellite network logical address of the destination host.
5[0-15]	Source Host Address. This field contains the satellite network logical address of the source host.
6-N	Data. This field contains up to 16,384 bits (1024 16-bit words) of user data.

4 Stream Messages

Stream messages are the second type of message level data messages. As noted in Section 2, streams exist primarily to provide a one satellite hop delay for volatile traffic such as speech. Hosts may also use streams to support high duty cycle applications which require guaranteed channel bandwidth.

Streams must be created before stream messages can flow from host to host. The protocol to accomplish stream creation is described in Section 6.1. Once established, a stream is associated with a recurring channel allocation within the satellite network. This fixed allocation imposes rather strict requirements on the host using the stream if efficient channel utilization is to be achieved. In particular, stream messages must match the stream allocation both in terms of message size and message interarrival time.

Within the bounds of its stream allocation, a host is permitted considerable flexibility in how it may use a stream. Although the priority, reliability, and reliability length of each stream message is fixed at stream creation time, the destination logical address can vary from stream message to stream message. A host can, therefore, multiplex a variety of logical flows onto a single host stream. The format of stream messages is described in Figure 2.

```
         15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
0       | 0|LB|GOPRI|   XXXX    |     MESSAGE NUMBER    |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
1       |                HEADER CHECKSUM                |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
2       |                      A/R                      |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
3       | 1|IL| D| E| TTL |       HOST STREAM ID        |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
4       |           DESTINATION HOST ADDRESS            |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
5       |              SOURCE HOST ADDRESS              |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
6-N     |                     DATA                      |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 2 . STREAM MESSAGE

0[15]	Message Class = 0 (Data Message).
0[14]	Loopback Bit.
0[12-13]	Go-Priority.
0[8-11]	Reserved.
0[0-7]	Message Number. This field serves the same purpose as the message number field in the datagram message. Moreover, a single message number sequence is used for both datagram and stream messages (see Section 5).
1[0-15]	Header Checksum. Covers Words 0-5.
2[0-15]	Piggybacked A/R.

3[15]	Data Message Type = 1 (Stream).
3[14]	Internet/Local Flag.
3[13]	Discard Flag.
3[12]	Data Error Flag.
3[10-11]	Time-to-live Designator. 0 = Reserved 1 = 1 second 2 = Reserved 3 = Reserved
3[0-9]	Host Stream ID. The service host uses this field to identify the host stream over which the message is to be sent by the SIMP. Host stream IDs are established at stream creation time via host exchanges with their network service host (see Section 6.1).
4[0-15]	Destination Host Address.
5[0-15]	Source Host Address.
6-N	Data. This field contains up to 16,000 bits of user data (multiple of 16-bits).

5 Flow Control Messages

The SIMP supports an acceptance/refusal (A/R) mechanism in each direction on the host access link. The A/R mechanism is enabled for the link by the host by setting a bit in the Restart Complete control message (see Section 8). Each datagram and stream message contains an 8-bit message number used to identify the message for flow control purposes. Both the host and the SIMP increment this number modulo 256 in successive messages they transmit. Up to 127 messages may be outstanding in each direction at any time. If the receiver of a message is unable to accept the message, a refusal indication containing the message number of the refused message and the reason for the refusal is returned. The refusal indication may be piggybacked on data messages in the opposite direction over the link or may be sent in a separate control message in the absence of reverse traffic.

Acceptance indications are returned in a similar manner, either piggybacked on data messages or in a separate control message. An acceptance is returned by the receiver to indicate that the identified message was not refused. Acceptance indications returned by the SIMP do not, however, imply a guarantee of delivery or even any assurance that the message will not be intentionally discarded by the network at a later time. They are sent primarily to facilitate buffer management in the host.

To reduce the number of A/R messages exchanged, a single A/R indication can be returned for multiple (lower numbered) previously unacknowledged messages. Explicit acceptance of message number N implies implicit acceptance of outstanding messages with numbers N-1, N-2, etc., according to the definition of acceptance outlined above. (Note that explicit acceptance of message number N does not imply that all of the unacknowledged outstanding messages have been received.) An analogous interpretation of refusal message number allows the receiver of a group of messages to reject them as a group assuming that they all are being refused for the same reason. As a further efficiency measure, HAP permits a block of A/R indications to be aggregated into a single A/R control message. Such a message might be used, for example, to reject a group of messages where the refusal code on each is different.

In some circumstances the overhead associated with processing A/R messages may prove unattractive. For these cases, it is possible to disable the A/R mechanism and operate the HAP interface in a purely discard mode. The ability to effect this on a link basis has already been noted (see Sections 2 and 8). In addition, messages with sequence number zero are taken as messages for which the A/R mechanism is selectively disabled. To permit critical feedback, even when operating in discard mode, HAP defines an "Unnumbered Response" control message.

The format shown in Figure 3 is used both for piggybacking A/R indications on data messages (word 2), and for providing A/R information in separate control messages. When separate control messages are used to transmit A/R indications, the format shown in Figure 4 applies. Flow control information and other information which cannot be sent as an A/R indication is sent in an Unnumbered Response control message. The format of this type of message is illustrated in Figure 5.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|AR|     REFUSAL CODE     |  A/R MESSAGE NUMBER   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 3 . ACCEPTANCE/REFUSAL WORD

[15] Acceptance/Refusal Type. This field identifies whether A/R information is an acceptance or a refusal.

0 = Acceptance
1 = Refusal

[8-14] Refusal Code. When the Acceptance/Refusal Type = 1, this field gives the Refusal Code.

```
 0 = Priority not being accepted
 1 = Source SIMP congestion
 2 = Destination SIMP congestion
 3 = Destination host dead
 4 = Destination SIMP dead
 5 = Illegal destination host address
 6 = Destination host access not allowed
 7 = Illegal source host address
 8 = Message lost in access link
 9 = Nonexistent stream ID
10 = Illegal source host for stream ID
11 = Message length too long
12 = Stream message too early
13 = Illegal control message type
14 = Illegal refusal code in A/R
15 = Illegal reliability value
16 = Destination host congestion
```

[0-7] A/R Message Number. This field contains the number of

the message to which this acceptance/refusal refers. It also applies to all outstanding messages with earlier numbers. Note that this field can never be zero since a message number of zero implies that the A/R mechanism is disabled.

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   0     | 1|LB|GOPRI|   XXXX    |  LENGTH   |     1     |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   1     |                HEADER CHECKSUM                |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   2     |                      A/R                      |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   .     .                      ...                      .
   .     .                      ...                      .
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   N     |                      A/R                      |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 4 . ACCEPTANCE/REFUSAL MESSAGE

0[15]	Message Class = 1 (Control Message).
0[14]	Loopback Bit.
0[12-13]	Go-Priority.
0[8-11]	Reserved.
0[4-7]	Message Length. This field contains the total length of this message in words (N+1).
0[0-3]	Control Message Type = 1 (Acceptance/Refusal).
1[0-15]	Header Checksum. The checksum covers words 0-N.
2[0-15]	Acceptance/Refusal Word.
3-N	Additional Acceptance/Refusal Words (optional).

```
            15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   0       | 1|LB|GOPRI|   XXXX    | RES-CODE  |     5     |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   1       |                HEADER CHECKSUM                |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   2       |                 RESPONSE INFO                 |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   3       |                 RESPONSE INFO                 |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 5 . UNNUMBERED RESPONSE

0[15]	Message Class = 1 (Control Message).
0[14]	Loopback Bit.
0[12-13]	Go-Priority.
0[8-11]	Reserved.
0[4-7]	Response Code.

```
        3 = Destination unreachable
        5 = Illegal destination host address
        7 = Illegal source host address
        9 = Nonexistent stream ID
       10 = Illegal stream ID
       13 = Protocol violation
       15 = Can't implement loop
```

0[0-3]	Control Message Type = 5 (Unnumbered Response).
1[0-15]	Header Checksum. Covers words 0-3.

```
2[0-15]   Response Information. If Response Code is:

               3, Destination Host Address
               5, Destination Host Address
               7, Source Host Address
               9, Stream ID (right justified)
              10, Stream ID (right justified)
              13, Word 0 of offending message
              15, Word 0 of Loopback Request message

3[0-15]   Response Information. If Response Code is:

               3,5,7, or 9. Undefined
               10, Source Host Address
               13, Word 3 of offending message, or zero if
                   no word 3
               15, Word 2 of Loopback Request message
```

6 Setup Level Messages

Setup level protocol is provided to support the establishment, modification, and deletion of groups and streams in the packet satellite network. A host wishing to perform one of these generic operations interacts with the network service host (logical address zero). The service host causes the requested action to be carried out and serves as the intermediary between the user and the rest of the network. In the process of implementing the requested action, various network data bases are updated to reflect the current state of the referenced group or stream.

The communication between the host and the service host is implemented via special-purpose datagrams called setup messages. Each interaction initiated by a host involves a 3-way exchange where: (1) the user host sends a Request to the service host, (2) the service host returns a Reply to the user host, and (3) the user host returns a Reply Acknowledgment to the service host. This procedure is used to insure reliable transmission of requests and replies. In order to allow more than one setup request message from a host to be outstanding, each request is assigned a unique Request ID. The associated Reply and subsequent Reply Acknowledgment are identified by the Request ID that they contain. Hosts should generally expect a minimum delay of about two satellite round-trip times between the transmission of a setup Request to the SIMP and the receipt of the associated Reply. (Note that the Join Group Request and the Leave Group Request require only local communication between a host and its SIMP. The response time for these requests, therefore, is dependent solely on SIMP processing time and should be considerably shorter than two round-trip times.) This delay establishes a maximum rate at which changes can be processed by the SIMP. The user should receive a reply to a setup request requiring global communication within 2 seconds and to a setup request requiring local communication within 1 second. The host should respond to a SIMP Reply with a Reply Acknowledgment within 1 second.

Setup exchanges can also be initiated by the SIMP. SIMP-initiated setup messages are used to notify a host of changes in the status of an associated group or stream. Each notification involves a 2-way exchange where: (1) the service host sends a Notification to the user host, and (2) the user host returns a Notification Acknowledgment to the service host. In order to allow more than one Notification to be outstanding, each is assigned a unique Notification ID. The Notification Acknowledgment returned by the user host to the service host must contain the Notification ID.

The general format of every setup message is:

```
<DATAGRAM MESSAGE HEADER>
<OPTIONAL INTERNET HEADER>
<SETUP MESSAGE HEADER>
<SETUP MESSAGE BODY>
```

The service host accepts setup requests in either Internet or non-Internet format. Replies from the service host will be in the same form as the request, that is, Internet requests get Internet replies, and non-Internet requests get non-Internet replies.

The format of the combined datagram message header and setup message header is illustrated in Figure 6. The body of the setup messages depends on the particular setup message type. Stream request and reply messages are described in Section 6.1. Group request and reply messages are described in Section 6.2. To simplify the presentation in both of these sections, the setup messages are assumed to be exchanged between a local host and SIMP even though Internet group and stream setups are supported (see Figure 6). The format of notifications, which consists of only a single word beyond the basic setup header, is shown in Figure 7. Since the SIMP does not retain the optional Internet header information that can be included in setup requests, Internet notifications are not supported. The format of acknowledgment messages associated with request/reply and notification setups is illustrated in Figure 8.

```
               15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
              +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     0-5      |            DATAGRAM MESSAGE HEADER            |
              +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     6-N      |          <OPTIONAL INTERNET HEADER>           |
              +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     N+1      |      SETUP TYPE       |      SETUP CODE       |
              +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     N+2      |                SETUP CHECKSUM                 |
              +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     N+3      |                   SETUP  ID                   |
              +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 6 . SETUP MESSAGE HEADER

0-5 Datagram Message Header. Each setup message begins with the six word datagram message header (see Section 3).

6-N Internet Header (Optional). These fields, when present, conform to the DoD Standard Internet Protocol (IP). The Internet header size is a minimum of 10 words but can be longer depending on the use of optional IP facilities. (Internet notification messages are not supported.)

N+1[8-15] Setup Type. This field determines the type of setup message.

0 = Acknowledgment
1 = Request
2 = Reply
3 = Notification

N+1[0-7] Setup Code. For requests, this field identifies the

Request Type.

1 = Create group address
2 = Delete group address
3 = Join group
4 = Leave group
5 = Create stream
6 = Delete stream
7 = Change stream parameters
8 = Reserved

For Replies, this field provides the Reply Code. Some of the Reply Codes can be returned to any setup request and others are request specific.

0 = Group or stream created
1 = Group or stream deleted
2 = Group joined
3 = Group left
4 = Stream changed
5 = Reserved
6 = Bad request type
7 = Reserved
8 = Network trouble
9 = Bad key
10 = Group address/stream ID nonexistent
11 = Not member of group/creator of stream
12 = Stream priority not being accepted
13 = Reserved
14 = Reserved
15 = Illegal interval
16 = Reserved
17 = Insufficient network resources
18 = Requested bandwidth too large
19 = Reserved
20 = Reserved
21 = Maximum messages per slot not consistent with slot size
22 = Reply lost in network
23 = Illegal reliability value

For Notifications, this field contains the Notification Type.

    0 = Stream suspended
    1 = Stream resumed
    2 = Stream deleted
    3 = Group deleted by host
    4 = Group deleted by SIMP
    5 = All streams deleted
    6 = All groups deleted

For Acknowledgments, this field contains the Acknowledgment Type.

    0 = Reply acknowledgment
    1 = Notification acknowledgment

N+2[0-15] Setup Checksum. The checksum covers the three setup message header words and the setup message body data words. Setups received with bad checksums must be discarded.

N+3[0-15] Setup ID. This field is assigned by the host to uniquely identify outstanding requests (Request ID) and by the service host to uniquely identify outstanding notifications (Notification ID).

```
             15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0-5       |            DATAGRAM MESSAGE HEADER            |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   6        |            3          |  NOTIFICATION TYPE    |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   7        |                SETUP CHECKSUM                 |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   8        |                NOTIFICATION ID                |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   9        |               NOTIFICATION INFO               |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 7 . NOTIFICATION MESSAGE

0-5	Datagram Message Header (see Section 3).
6[8-15]	Setup Type = 3 (Notification).
6[0-7]	Notification Type. 0 = Stream suspended 1 = Stream resumed 2 = Stream deleted 3 = Group deleted by host 4 = Group deleted by SIMP 5 = All streams deleted 6 = All groups deleted
7[0-15]	Setup Checksum. Covers words 6-9.
8[0-15]	Notification ID.
9[0-15]	Notification Information. This field contains the 16-bit group address in the case of a group

notification (types 3 and 4) and the 10-bit host stream ID (right justified) in the case of a stream notification (types 0-2). This field is zero for Notification Types 5 and 6, which pertain to ALL streams and groups, respectively.

```
           15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0-5     |              DATAGRAM MESSAGE HEADER          |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   6      |          0            |       ACK TYPE        |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   7      |                 SETUP CHECKSUM                |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   8      |                   SETUP  ID                   |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 8 . SETUP ACKNOWLEDGMENT

0-5	Datagram Message Header.
6[8-15]	Setup Type = 0 (Acknowledgment).
6[0-7]	Acknowledgment Type. 0 = Reply acknowledgment 1 = Notification acknowledgment
7[0-15]	Setup Checksum. Covers words 6-8.
8[0-15]	Setup ID. This is either a Request ID or a Notification ID.

6.1 Stream Setup Messages

Hosts use streams to support high duty cycle applications and applications requiring a one satellite hop network transmission delay. Host streams must be set up before stream data messages can flow. The stream setup messages defined by HAP are Create Stream Request, Create Stream Reply, Delete Stream Request, Delete Stream Reply, Change Stream Parameters Request, and Change Stream Parameters Reply. The use of these messages is illustrated in the scenario of exchanges between a host and its local SIMP shown in Figure 9 where the host establishes a stream, sends some data, modifies the stream characteristics, sends some more data, and finally closes down the stream.

It is worthwhile noting that the setup exchanges in Figure 9 are completely between the host originating the stream and its local SIMP. Other SIMPs and hosts are essentially unaware of the existence of the stream. Stream messages received by a destination host are, therefore, processed identically to datagram messages. (All SIMPs must, of course, be aware of the channel allocation associated with a host stream in order to perform satellite channel scheduling.) Not illustrated, but implicit in this scenario, are the optional A/R indications associated with each of the stream setup messages.

```
                                          Host       SIMP

Create Stream Request                          ------>
Create Stream Reply                            <------
Reply Acknowledgment                           ------>
Stream Message                                 ------>
      .
      .
Stream Message                                 ------>
Change Stream Parameters Request               ------>
Change Stream Parameters Reply                 <------
Reply Acknowledgment                           ------>
Stream Message                                 ------>
      .
      .
Stream Message                                 ------>
Delete Stream Request                          ------>
Delete Stream Reply                            <------
Reply Acknowledgment                           ------>
```

Figure 9 . STREAM EXAMPLE

Host streams have six characteristic properties which are selected at stream setup time. These properties, which apply to every message transmitted in the stream, are: (1) slot size, (2) interval, (3) reliability, (4) reliability length, (5) priority, and (6) maximum messages per slot. To establish a stream, the host sends the Create Stream Request message illustrated in Figure 10 to the SIMP. After the satellite network has processed the Create Stream Request, the SIMP will respond to the host with a Create Stream Reply message formatted as shown in Figure 11. Assuming that the reply code in the Create Stream Reply is zero indicating that the stream has been created successfully, the host may proceed to transmit stream data messages after sending a

Reply Acknowledgment.

During the lifetime of a stream, the host which created it may decide that some of its six characteristic properties should be modified. All of the properties except the stream interval can be modified using the Change Stream Parameters Request message. The format of this command is illustrated in Figure 12. After the network has processed the Change Stream Parameters Request, the SIMP will respond by sending a Change Stream Parameters Reply to the host with the format shown in Figure 13. A host requesting a reduced channel allocation should decrease its sending rate immediately without waiting for receipt of the Change Stream Parameters Reply. A host requesting an increased allocation should not proceed to transmit according to the new set of parameters without first having received a Reply Code of 4 indicating that the requested change has taken effect.

When the host which created the host stream determines that the stream is no longer needed and the associated satellite channel allocation can be freed up, the host sends its local SIMP a Delete Stream Request message formatted as indicated in Figure 14. After the network has processed the Delete Stream Request, the SIMP will respond by sending a Delete Stream Reply to the host with the format shown in Figure 15.

```
              15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0-5        |             DATAGRAM MESSAGE HEADER           |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   6         |           1           |           5           |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   7         |                SETUP CHECKSUM                 |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   8         |                  REQUEST ID                   |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   9         |  MAX MES  | INT | PRI | RLY |       RLEN      |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  10         |                  SLOT SIZE                    |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 10 . CREATE STREAM REQUEST

0-5	Datagram Message Header.
6[8-15]	Setup Type = 1 (Request).
6[0-7]	Request Type = 5 (Create Stream).
7[0-15]	Setup Checksum. Covers words 6-10.
8[0-15]	Request ID.
9[12-15]	Maximum Messages Per Slot. This field specifies the the maximum number of stream messages that will ever be delivered to the SIMP by the host for transmission in one stream slot.
9[10-11]	Interval. This field specifies the interval, in number of 21.2 ms frames, between stream slots.

0 = 1 frame
1 = 2 frames
2 = 4 frames
3 = 8 frames

As an example, an interval of 4 frames corresponds to an allocation of Slot Size words every 85 ms.

9[8-9] Priority. This field specifies the priority at which all messages in the host stream should be handled.

0 = Low priority
1 = Medium Low Priority
2 = Medium High Priority
3 = High Priority

9[6-7] Reliability. This field specifies the basic bit-error rate requirement for the data portion of all messages in the host stream.

0 = Low Reliability
1 = Medium Reliability
2 = High Reliability
3 = Reserved

9[0-5] Reliability Length. This field specifies how many words beyond the stream message header should be transmitted at maximum reliability for all messages in the host stream.

10[0-15] Slot Size. This field specifies the slot size in 16-bit words of stream message text. Stream message header words are excluded from this count. The host can partition this allocation on a slot-by-slot basis among a variable number of messages as long as the maximum number of messages per slot does not exceed MAX MES.

```
           15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0-5     |             DATAGRAM MESSAGE HEADER           |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   6      |           2           |      REPLY CODE       |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   7      |                 SETUP CHECKSUM                |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   8      |                   REQUEST ID                  |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   9      |     XXXXX      |        HOST STREAM ID        |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 11 . CREATE STREAM REPLY

0-5	Datagram Message Header.
6[8-15]	Setup Type = 2 (Reply).
6[0-7]	Reply Code.

```
 0 = Stream created
 8 = Network trouble
12 = Stream priority not being accepted
17 = Insufficient network resources
18 = Requested bandwidth too large
21 = Maximum messages per slot not consistent
     with slot size
22 = Reply lost in network
23 = Illegal reliability value
```

7[0-15]	Setup Checksum. Covers words 6-9.
8[0-15]	Request ID.

9[10-15]	Reserved.
9[0-9]	Host Stream ID. This field contains a host stream ID assigned by the network. It must be included in all stream data messages sent by the host to allow the SIMP to associate the message with stored stream characteristics and the reserved satellite channel time.

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
0-5      |            DATAGRAM MESSAGE HEADER             |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 6       |           1           |           7            |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 7       |                SETUP CHECKSUM                  |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 8       |                  REQUEST ID                    |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 9       |      XXXXX      |        HOST STREAM ID        |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
10       |  MAX MES  | INT | PRI | RLY |       RLEN       |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
11       |                  SLOT SIZE                     |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 12 . CHANGE STREAM PARAMETERS REQUEST

0-5	Datagram Message Header.
6[8-15]	Setup Type = 1 (Request).
6[0-7]	Request Type = 7 (Change Stream Parameters).
7[0-15]	Setup Checksum. Covers words 6-11.
8[0-15]	Request ID.
9[10-15]	Reserved.
9[0-9]	Host Stream ID.
10[12-15]	New Maximum Messages Per Slot.

10[10-11]	Interval. This field must specifiy the same interval as was specified in the Create Stream Request message for this stream.
10[8-9]	New Priority.
10[6-7]	New Reliability.
10[0-5]	New Reliability Length.
11[0-15]	New Slot Size.

```
            15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0-5      |           DATAGRAM MESSAGE HEADER             |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   6       |         2             |      REPLY CODE       |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   7       |                SETUP CHECKSUM                 |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   8       |                  REQUEST ID                   |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 13 . CHANGE STREAM PARAMETERS REPLY

```
0-5        Datagram Message Header.

  6[8-15]  Setup Type = 2 (Reply).

  6[0-7]   Reply Code.

                 4 = Stream changed
                 8 = Network trouble
                10 = Stream ID nonexistent
                11 = Not creator of stream
                12 = Stream priority not being accepted
                15 = Illegal interval
                17 = Insufficient network resources
                18 = Requested bandwidth too large
                21 = Maximum messages per slot not consistent with
                     slot size
                22 = Reply lost in network
                23 = Illegal reliability value

  7[0-15]  Setup Checksum.  Covers words 6-8.

  8[0-15]  Request ID.
```

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 0-5     |             DATAGRAM MESSAGE HEADER           |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  6      |           1           |           6           |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  7      |                 SETUP CHECKSUM                |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  8      |                   REQUEST ID                  |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  9      |      XXXXX      |        HOST STREAM ID       |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 14 . DELETE STREAM REQUEST

0-5	Datagram Message Header.
6[8-15]	Setup Type = 1 (Request).
6[0-7]	Request Type = 6 (Delete Stream).
7[0-15]	Setup Checksum. Covers words 6-9.
8[0-15]	Request ID.
9[10-15]	Reserved.
9[0-9]	Host Stream ID.

```
           15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
0-5       |            DATAGRAM MESSAGE HEADER            |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 6        |           2           |       REPLY CODE      |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 7        |                SETUP CHECKSUM                 |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 8        |                  REQUEST ID                   |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 15 . DELETE STREAM REPLY

0-5 Datagram Message Header.

6[8-15] Setup Type = 2 (Reply).

6[0-7] Reply Code.

```
     1 = Stream deleted
     8 = Network trouble
    10 = Stream ID nonexistent
    11 = Not creator of stream
    17 = Insufficient network resources
    22 = Reply lost in network
```

7[0-15] Setup Checksum. Covers words 6-8.

8[0-15] Request ID.

6.2 Group Setup Messages

Group addressing allows hosts to take advantage of the broadcast capability of the satellite network and is primarily provided to support the multi-destination delivery required for conferencing applications. Group addresses are dynamically created and deleted via setup messages exchanged between hosts and the network. Membership in a group may consist of an arbitrary subset of all the permanent network hosts. A datagram message or stream message addressed to a group is always sent over the satellite channel and delivered to all hosts that are members of that group. The group setup messages are Create Group Request, Create Group Reply, Delete Group Request, Delete Group Reply, Join Group Request, Join Group Reply, Leave Group Request, and Leave Group Reply.

The use of group setup messages is shown in Figure 16. The figure illustrates a scenario of exchanges between two hosts and their local SIMPs. In the scenario one host, Host A, creates a group which is joined by a second host, Host B. After the two hosts have exchanged some data mesages addressed to the group, Host B decides to leave the group and Host A decides to delete the group. As in the scenario in Section 6.1, A/R indications have been omitted for clarity.

Part of the group creation procedure involves the service host returning a 48-bit key along with a 16-bit group address to the host creating the group. The creating host must pass the key along with the group address to the other hosts which it wants as group members. These other hosts must supply the key along with the group address in their Join Group Requests. The key is used by the network to authenticate these operations and thereby minimize the probability that unwanted hosts will deliberately or inadvertently become members of the group. The procedure used by a host to distribute the group address and key is not within the scope of HAP.

```
                            Host    SIMP     SIMP   Host
                             A       A        B      B

Create Group Request          ------>
Create Group Reply            <------
Reply Acknowledgment          ------>
      .
      .
                              >>Group Address,Key>>
      .
      .
Join Group Request                             <------
Join Group Reply                               ------>
Reply Acknowledgment                           <------

Data Message 1                ------>
Data Message 1                <------          ------>
Data Message 2                                 <------
Data Message 2                <------          ------>
Leave Group Request                            <------
Leave Group Reply                              ------>
Reply Acknowledgment                           <------
Delete Group Request          ------>
Delete Group Reply            <------
Reply Acknowledgment          ------>
```

Figure 16 . GROUP EXAMPLE

Any host no longer wishing to participate in a group may choose to drop out. This can be accomplished by either a Leave or a Delete. Both Leave and Delete operations are authenticated using the 48-bit key. Leave is a local operation between a host and its SIMP which removes the requesting host from the group membership list but does not alter the global existence of the group. A Delete, on the other hand, expunges all knowledge of the group from every SIMP in the network. HAP will permit any member of a group to delete the group at any time. Thus, group addresses can be deleted even if the host which originally created the group has left the group or has crashed. Moreover, groups may exist for which there are currently no members because each member has executed a Leave while none has executed a Delete. It is the responsibility of the hosts to coordinate and manage the use of groups.

The Create Group Request message sent to the service host to establish a group address is illustrated in Figure 17. After the network has processed the Create Group Request, the service host will respond by sending a Create Group Reply to the host as illustrated in Figure 18.

A host may become a member of a group once it knows the address and key associated with the group by sending the service host the Join Group Request message shown in Figure 19. The service host will respond to the Join Group Request with a Join Group Reply formatted as indicated in Figure 20. The host which creates a group automatically becomes a member of that group without any need for an explicit Join Group Request.

At any time after becoming a member of a group, a host may choose to drop out of the group. To effect this the host sends the service host a Leave Group Request formatted as shown in Figure 21. The service host will respond to the Leave Group Request with a Leave Group Reply formatted as shown in Figure 22.

Any member of a group can request that the service host delete an existing group via a Delete Group Request. The format of the Delete Group Request setup message is illustrated in Figure 23. After the network has processed the Delete Group Request, the service host will respond to the host with a Delete Group Reply formatted as illustrated in Figure 24.

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 0-5     |             DATAGRAM MESSAGE HEADER           |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 6       |           2           |      REPLY CODE       |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 7       |                 SETUP CHECKSUM                |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 8       |                   REQUEST ID                  |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 9       |                 GROUP ADDRESS                 |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 10      |                      KEY                      |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 11      |                      KEY                      |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 12      |                      KEY                      |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 18 . CREATE GROUP REPLY

0-5	Datagram Message Header.
6[8-15]	Setup Type = 2 (Reply).
6[0-7]	Reply Code.

```
 0 = Group created
 8 = Network trouble
17 = Insufficient network resources
22 = Reply lost in network
```

7[0-15]	Setup Checksum. Covers words 6-12.
8[0-15]	Request ID.

```
            15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 0-5       |            DATAGRAM MESSAGE HEADER            |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  6        |           1           |           1           |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  7        |                SETUP CHECKSUM                 |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  8        |                  REQUEST ID                   |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 17 . CREATE GROUP REQUEST

0-5	Datagram Message Header.
6[8-15]	Setup Type = 1 (Request).
6[0-7]	Request Type = 1 (Create Group).
7[0-15]	Setup Checksum. Covers words 6-8.
8[0-15]	Request ID.

9[0-15] Group Address. This field contains a 16-bit logical address assigned by the network which may be used by the host as a group address.

10-12 Key. This field contains a 48-bit key assigned by the network which is associated with the group address. It must be provided for subsequent Join, Leave, and Delete requests which reference the group address.

```
             15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   0-5      |             DATAGRAM MESSAGE HEADER           |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    6       |          1            |          3            |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    7       |                SETUP CHECKSUM                 |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    8       |                  REQUEST ID                   |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    9       |                GROUP ADDRESS                  |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   10       |                     KEY                       |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   11       |                     KEY                       |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   12       |                     KEY                       |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 19 . JOIN GROUP REQUEST

0-5	Datagram Message Header.
6[8-15]	Setup Type = 1 (Request).
6[0-7]	Request Type = 3 (Join Group).
7[0-15]	Setup Checksum. Covers words 6-12.
8[0-15]	Request ID.
9[0-15]	Group Address. This is the logical address of the group that the host wishes to join.
10-12	Key. This is the key associated with the group

address.

```
              15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      0-5    |              DATAGRAM MESSAGE HEADER          |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
       6     |          2            |      REPLY CODE       |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
       7     |                SETUP CHECKSUM                 |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
       8     |                  REQUEST ID                   |
             +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 20 . JOIN GROUP REPLY

```
0-5          Datagram Message Header.

  6[8-15]    Setup Type = 2 (Reply).

  6[0-7]     Reply Code.

                  2 = Group joined
                  9 = Bad key
                 10 = Group address nonexistent
                 17 = Insufficient network resources

  7[0-15]    Setup Checksum.  Covers words 6-8.

  8[0-15]    Request ID.
```

```
             15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   0-5      |              DATAGRAM MESSAGE HEADER          |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    6       |           1           |           4           |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    7       |                 SETUP CHECKSUM                |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    8       |                  REQUEST ID                   |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    9       |                 GROUP ADDRESS                 |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   10       |                      KEY                      |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   11       |                      KEY                      |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   12       |                      KEY                      |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 21 . LEAVE GROUP REQUEST

0-5	Datagram Message Header.
6[8-15]	Setup Type = 1 (Request).
6[0-7]	Request Type = 4 (Leave Group).
7[0-15]	Setup Checksum. Covers words 6-12.
8[0-15]	Request ID.
9[0-15]	Group Address. This is the logical address of the group that the host wishes to leave.
10-12	Key. This is the key associated with the group

address.

```
           15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 0-5      |              DATAGRAM MESSAGE HEADER          |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  6       |           2           |      REPLY CODE       |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  7       |                 SETUP CHECKSUM                |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  8       |                   REQUEST ID                  |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 24 . DELETE GROUP REPLY

0-5 Datagram Message Header.

6[8-15] Setup Type = 2 (Reply).

6[0-7] Reply Code.

```
 1 = Group deleted
 8 = Network trouble
 9 = Bad key
10 = Group address nonexistent
11 = Not member of group
17 = Insufficient network resources
22 = Reply lost in network
```

7[0-15] Setup Checksum. Covers words 6-8.

8[0-15] Request ID.

7 Link Monitoring

While the access link is operating, statistics on traffic load and error rate are maintained by the host and SIMP. The host and SIMP must exchange status messages once a second. Periodic exchange of status messages permits both ends of the link to monitor flows in both directions. Status messages are required to support monitoring by the Network Operations Center (NOC).

The link restart procedure (see Section 8) initializes all internal SIMP counts and statistics for that link to zero. As data and control messages are processed, counts are updated to reflect the total number of messages sent, messages received correctly, and messages received with different classes of errors since the last link restart. Whenever a status message arrives, a snapshot is taken of the local SIMP counts. The local receive counts, in conjunction with a sent count contained in the received status message, permits the computation of traffic statistics in the one second update interval assuming that the set of counts at the time of the previous monitoring report have been saved. By including in the status message sent (in the opposite direction) the receive counts and the received sent count that was used with them, the transmitting end of the access link as well as the receiving end can determine the link performance from sender to receiver. The format of the Status control message is illustrated in Figure 25.

```
            15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0-5      |             DATAGRAM MESSAGE HEADER           |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   6       |           1           |           2           |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   7       |                SETUP CHECKSUM                 |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   8       |                  REQUEST ID                   |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   9       |                GROUP ADDRESS                  |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  10       |                     KEY                       |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  11       |                     KEY                       |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  12       |                     KEY                       |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 23 . DELETE GROUP REQUEST

0-5	Datagram Message Header.
6[8-15]	Setup Type = 1 (Request).
6[0-7]	Request Type = 2 (Delete Group).
7[0-15]	Setup Checksum. Covers words 6-12.
8[0-15]	Request ID.
9[0-15]	Group Address.
10-12	Key.

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0-5     |             DATAGRAM MESSAGE HEADER           |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   6      |          2            |      REPLY CODE       |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   7      |                SETUP CHECKSUM                 |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   8      |                  REQUEST ID                   |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 22 . LEAVE GROUP REPLY

```
0-5         Datagram Message Header.

  6[8-15]   Setup Type = 2 (Reply).

  6[0-7]    Reply Code.

                 3 = Group left
                 9 = Bad key
                10 = Group address nonexistent
                11 = Not member of group
                17 = Insufficient network resources

  7[0-15]   Setup Checksum.  Covers words 6-8.

  8[0-15]   Request ID.
```

RFC 907 Host Access Protocol
July 1984 Specification

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   0      | 1|LB|GOPRI|         XXXXX         |     0     |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   1      |                HEADER CHECKSUM                |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   2      |             MOST RECENT A/R SENT              |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   3      |                STREAM CAPACITY                |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   4      |                   TIMESTAMP                   |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   5      |                      SBU                      |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   6      |                      STU                      |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   7      |                      RNE                      |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   8      |                      RWE                      |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   9      |                      BFC                      |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  10      |                      HEI                      |
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 25 . STATUS MESSAGE

0[15]	Message Class = 1 (Control Message).
0[14]	Loopback Bit.
0[12-13]	Go-Priority.
0[4-11]	Reserved.

0[0-3]	Control Message Type = 0 (Status).
1[0-15]	Header Checksum. Covers words 0-10.
2[0-15]	Most Recent A/R Sent. This field is a duplicate of the most recent acceptance/refusal word. It is included in the periodic status message in case previous transmissions containing A/R information were lost.
3[0-15]	Stream Capacity. When sent by the SIMP, this field indicates how much stream capacity is unused, in units of data bits per frame. Since available capacity depends directly on a variety of parameters that can be selected by the user, the value of this field is the maximum capacity that could be achieved if existing host streams were expanded at low reliability. This field is undefined in messages sent from the host to the SIMP.
4[0-15]	Timestamp. This field indicates the time that the status message was generated. When sent by a SIMP, the time is in units of seconds since the last link restart. The host should also timestamp its messages in units of seconds.
5[0-15]	Sent By Us. Count of messages sent by us since the last link restart (not including this one).
6[0-15]	Sent To Us. Count of messages sent to us since the last link restart. This is the count from word 5 of the last status message received.
7[0-15]	Received, No Errors. This is the count of messages received without errors (since the last link restart) at the time that the last status message was received.
8[0-15]	Received With Errors. This is the count of messages received with errors (since the last link restart) at the time the last status message was received.
9[0-15]	Bad Header Checksums. This is the count of messages

received with bad header checksums (since the last link restart) at the time the last status message was received.

10[0-15] Hardware Error Indication. This is the count of messages received with hardware CRC errors or hardware interface error indications (since the last link restart) at the time the last status message was received.

8 Initialization

The Host Access Protocol uses a number of state variables that must be initialized in order to function properly. These variables are associated with the send and receive message numbers used by the acceptance/refusal mechanism and the statistics maintained to support link monitoring. Link initialization should be carried out when a machine is initially powered up, when it does a system restart, when the ON state (see below) times out, when a loopback condition times out (see Section 9), or whenever the link transitions from non-operational to operational status.

Initialization is accomplished by the exchange of Restart Request (RR) and Restart Complete (RC) messages between a host and a SIMP. The state diagram in Figure 26 shows the sequence of events during initialization. Both SIMP and host must implement this state diagram if deadlocks and oscillations are to be avoided. This particular initialization sequence requires both sides to send and receive the Restart Complete message. Because this message is a reply (to a Restart Request or Restart Complete), its receipt guarantees that the physical link is operating in both directions. Five states are identified in the state diagram:

OFF	Entered upon recognition of a requirement to restart. The device can recognize this requirement itself or be forced to restart by receipt of an RR message from the other end while in the ON state.
INIT	Local state variables have been initialized and local counters have been zeroed but no restart control messages have yet been sent or received.
RR-SNT	A request to reinitialize (RR) has been sent to the other end but no restart control messages have yet been received.
RC-SNT	A reply (RC) has been sent to the other end in response to a received reinitialization request

(RR). The device is waiting for a reply (RC).

ON	Reply (RC) messages have been both sent and received. Data and control messages can now be exchanged between the SIMP and host.

All states have 10-second timeouts (not illustrated) which return the protocol to the OFF state. The occurrence of any events other than those indicated in the diagram are ignored.

The Restart Request control message illustrated in Figure 27 is sent by either a host or a SIMP when it wishes to restart a link. The Restart Request causes all the monitoring statistics to be reset to zero and stops all traffic on the link in both directions. The Restart Complete message illustrated in Figure 28 is sent in response to a received Restart Request or Restart Complete to complete link initialization. The Restart Complete carries a field used by the host to enable or disable the acceptance/refusal mechanism for the link being restarted (see Section 5). After the Restart Complete is processed, traffic may flow on the link.

```
                          -------
Any Timeout or  ----->| OFF  |<-----------------------------
Device Down               -------                          |
                            |                              |
                            |  Device Up                   |
                            |  Initialize Variables        |
                            |                              |
                            V                              |
                        ---------                          |
                        | INIT  |                          |
                        ---------                          |
                          |   |                            |
             Rcv RR       |   |      Snd RR                |
             Snd RC       |   |                            |
                          |   |                            |
              --------------   --------------              |
              |                             |              |
              |                             |              |
              V              Rcv RR         V              |
         ----------          Snd RC     ----------         |
         | RC-SNT |<--------------------| RR-SNT |         |
         ----------                     ----------         |
              |                             |              |
 Rcv RC       |                             |   Rcv RC     |
              |                             |   Snd RC     |
              V                             V              |
              -------------------------------              |
                             |                             |
                             |                             |
                             V                             |
                          -------                          |
  Rcv Any         ------->| ON  |---------------------------
  Other           |       -------        Rcv RR
                  ----------|
```

Figure 26 . HAP LINK RESTART STATE DIAGRAM

```
             15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0         | 1|LB|      XXXXXXX      |  REASON   |     3     |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  1         |                 HEADER CHECKSUM               |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  2         |          HOST ADDRESS / SITE NUMBER           |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  3         |                   LINK NUMBER                 |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 27 . RESTART REQUEST

0[15] Message Type = 1 (Control Message).

0[14] Loopback Bit.

0[8-13] Reserved.

0[4-7] Reason. This field is used by the SIMP or the host to indicate the reason for the restart as follows:

0 = power up
1 = system restart
2 = link restart
3 = link timeout
4 = loopback timeout

0[0-3] Control Message Type = 3 (Restart Request).

1[0-15] Header Checksum. Covers words 0-3.

2[0-15] Host Address / Site Number. The host inserts its satellite network address in this field. The SIMP validates that the host address is correct for the port being used. When sent by the SIMP, this field will contain the SIMP site number.

3[0-15] Link Number. This field contains the sender's identification of the physical link being used. This information is used to identify the link when reporting errors to the Network Operations Center (NOC).

```
         15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
0       | 1|LB|          XXXXXX          |AR|     4     |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
1       |                HEADER CHECKSUM                |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
2       |          HOST ADDRESS / SITE NUMBER           |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
3       |                  LINK NUMBER                  |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 28 . RESTART COMPLETE

0[15]	Message Type = 1 (Control Message).
0[14]	Loopback Bit.
0[5-13]	Reserved.
0[4]	Acceptance/Refusal Control. This bit is used by the host to enable or disable the acceptance/refusal mechanism for all traffic on the link. 0 = Disable acceptance/refusal 1 = Enable acceptance/refusal
0[0-3]	Control Message Type = 4 (Restart Complete).
1[0-15]	Header Checksum. Covers words 0-3.
2[0-15]	Host Address / Site Number.
3[0-15]	Link Number.

9 Loopback Control

The Host Access Protocol provides a Loopback Request control message which can be used by a SIMP or a host to request the remote loopback of its HAP messages. Such requests are usually the result of operator intervention for purposes of system fault diagnosis. For clarity in the following discussion, the unit (SIMP or host) requesting the remote loopback is referred to as the "transmitter" and the unit implementing (or rejecting) the loopback is referred to as the "receiver". The format of a Loopback Request control message is illustrated in Figure 29.

When a transmitter is remotely looped, all of its HAP messages will be returned, unmodified, over the access link by the receiver. The receiver will not send any of its own messages to the transmitter while it is implementing the loop. SIMP-generated messages are distinguished from host-generated messages by means of the Loopback Bit that is in every HAP message header.

Two types of remote loopback may be requested: loopback at the receiver's interface hardware and loopback at the receiver's I/O driver software. HAP does not specify the manner in which the receiver should implement these loops; additionally, some receivers may use interface hardware which is incapable of looping the transmitter's messages, only allowing the receiver to provide software loops. A receiver may not be able to interpret the transmitter's messages as it is looping them back. If such interpretation is possible, however, the receiver will not act on any of the transmitter's messages other than requests to reinitialize the SIMP-host link (Restart Request (RR) control messages; see Section 8.)

When a receiver initiates a loopback condition in response to a loopback request, it makes an implicit promise to maintain the condition for the duration specified in the Loopback Request message. However, if an unanticipated condition such as a system restart occurs in either the transmitter or the receiver, the affected unit will try to reinitialize the SIMP-host link by sending an RR message to the other unit. If the RR message is recognized by the other unit a link initialization sequence can be completed. This will restore the link to an unlooped condition even if the specified loop duration has not yet expired. If a receiver cannot interpret a transmitter's RR messages, and in the absence of operator intervention at the receiver, the loop will remain in place for its duration.

HAP does not specify the characteristics of any loopback conditions that may be locally implemented by a given unit. An example of such a condition is that obtained when a SIMP commands its host interface to loop back its own messages. If such local loop conditions also cause the reflection of messages received from the remote unit, the remote unit will detect the condition via the HAP header Loopback Bit.

A specific sequence must be followed for setting up a remote loopback condition. It begins after the HAP link has been initialized and a decision is made to request a remote loop. The transmitter then sends a Loopback Request message to the receiver and waits for either (1) a 10-second timer to expire, (2) a "Can't implement loop" Unnumbered Response message from the receiver, or (3) one of its own reflected messages. If event (1) or (2) occurs the request has failed and the transmitter may, at its option, try again with a new Loopback Request message. If event (3) occurs, the remote loopback condition has been established. While waiting for one of these events, messages from the receiver are processed normally. Note that RR messages arriving from the receiver during this time will terminate the loopback request.

When a receiver gets a Loopback Request message, it either implements the requested loop for the specified duration, or returns a "Can't implement loop" response without changing the state of the link. The latter response would be returned, for example, if a receiver is incapable of implementing a requested hardware loop. A receiver should initiate reinitialization of the link with an RR message(s) whenever a loopback condition times out.

There is one asymmetry that is required in the above sequence to resolve the (unlikely) case where both SIMP and host request a remote loopback at the same time. If a SIMP receives a Loopback Request message from a host while it is itself waiting for an event of type (1)-(3), it will return a "Can't implement loop" response to the host and will continue to wait. A host in the converse situation, however, will abort its loopback request and will instead act on the SIMP's loopback request.

```
            15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  0        | 1|LB|GOPRI|    XXXXX  | LOOP TYPE |     8     |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  1        |                HEADER CHECKSUM                |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  2        |                 LOOP DURATION                 |
           +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 29 . LOOPBACK REQUEST

0[15]	Message Type = 1 (Control Message).
0[14]	Loopback Bit.
0[12-13]	Go-Priority.
0[8-11]	Reserved.
0[4-7]	Loop Type. This field indicates the type of loop that is being requested as follows: 0 = Undefined; 1 = Loop at interface (hardware loop); 2 = Loop at driver (software loop); 3-15 = Undefined
0[0-3]	Control Message Type = 8 (Loopback Request).
1[0-15]	Header Checksum. Covers words 0-2.
2[0-15]	Loop Duration. The transmitter of a Loopback Request message uses this field to specify the number of seconds that the loop is to be maintained by the receiver.

10 Other Control Messages

Before a SIMP or a host voluntarily disables a SIMP-host link, it should send at least one Link Going Down control message over that link. The format of such a message is illustrated in Figure 30. HAP does not define the action(s) that should be taken by a SIMP or a host when such a message is received; informing the Network Operations Center (NOC) and/or the network users of the impending event is a typical course of action. Note that each Link Going Down message only pertains to the SIMP-host link that it is sent over; if a host and a SIMP are connected by multiple links, these links may be selectively disabled.

A No Operation (NOP) control message may be sent at any time by a SIMP or a host. The format of such a message is illustrated in Figure 31. A NOP message contains up to 32 words of arbitrary data which are undefined by HAP. NOP messages may be required in some cases to clear the state of the SIMP-host link hardware.

```
             15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
0           | 1|LB|GOPRI|   XXXXX   |  REASON   |     7     |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
1           |                HEADER CHECKSUM                |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
2           |                TIME UNTIL DOWN                |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
3           |                 DOWN DURATION                 |
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 30 . LINK GOING DOWN

0[15]	Message Type = 1 (Control Message).
0[14]	Loopback Bit.
0[12-13]	Go-Priority.
0[8-11]	Reserved.
0[4-7]	Reason. This field is used by the SIMP or the host to indicate the reason for disabling this SIMP-host link as follows:

```
0 = NOT going down:  Cancel previous Link
    Going Down message
1 = Unspecified reason
2 = Scheduled PM
3 = Scheduled hardware work
4 = Scheduled software work
5 = Emergency restart
6 = Power outage
7 = Software breakpoint
8 = Hardware failure
```

```
             9 = Not scheduled up
            10 = Last warning:  The SIMP  or host  is disabling
                 the link in 10 seconds
            11-15 = Undefined
```

0[0-3]	Control Message Type = 7 (Link Going Down).
1[0-15]	Header Checksum. Covers words 0-3.
2[0-15]	Time Until Down. This field specifies the amount of time remaining until the SIMP or host disables the link (in minutes). An entry of zero indicates that there is less than a minute remaining.
3[0-15]	Down Duration. This field specifies the amount of time that the SIMP-host link will be down (in minutes). An entry of zero indicates that the down duration will be less than a minute. An entry of -1 (all bits set) indicates an indefinite down duration.

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
0        | 1|LB|            XXXXX           |     6     |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
1        |                 HEADER CHECKSUM               |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
2-N      |                 ARBITRARY DATA                |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 31 . NO OPERATION (NOP)

0[15]	Message Type = 1 (Control Message).
0[14]	Loopback Bit.
0[4-13]	Reserved.
0[0-3]	Control Message Type = 6 (NOP).
1[0-15]	Header Checksum. Covers words 0-N.
2-N	Arbitrary Data. Up to 32 words of data may be sent. The data are undefined by HAP.

Loader Debugger Protocol

RFC-909

Christopher Welles

BBN Communications Corporation

Walter Milliken

BBN Laboratories

July 1984

Status of This Memo

This RFC specifies a proposed protocol for the ARPA Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Table of Contents

FIGURES

CHAPTER 1

Introduction

The Loader-Debugger Protocol (LDP) is an application layer protocol for loading, dumping and debugging target machines from hosts in a network environment. This protocol is designed to accommodate a variety of target cpu types. It provides a powerful set of debugging services. At the same time, it is structured so that a simple subset may be implemented in applications like boot loading where efficiency and space are at a premium.

The authors would like to thank Dan Franklin and Peter Cudhea for providing many of the ideas on which this protocol is based.

1.1 Purpose of This Document

This is a technical specification for the LDP protocol. It is intended to be comprehensive enough to be used by implementors of the protocol. It contains detailed descriptions of the formats and usage of over forty commands. Readers interested in an overview of LDP should read the Summary of Features, below, and skim Sections 2 through 3.1. Also see Appendix B, the Command Summary. The remainder of the document reads best when accompanied by strong coffee or tea.

1.2 Summary of Features

LDP has the following features:

- o commands to perform loading, dumping and debugging
- o support for multiple connections to a single target
- o reliable performance in an internet environment
- o a small protocol subset for target loaders
- o addressing modes and commands to support multiple machine types
- o breakpoints and watchpoints which run in the target machine.

CHAPTER 2

General Description

2.1 Motivation

LDP is an application protocol that provides a set of commands used by application programs for loading, dumping and debugging target machines across a network.

The goals of this protocol are shown in the following list:

o The protocol should support various processor types and operating systems. Overhead and complexity should be minimized for simpler cases.

o The protocol should provide support for applications in which more than one user can debug the same target machine. This implies an underlying transport mechanism that supports multiple connections between a host-target pair.

o LDP should have a minimal subset of commands for boot loading and dumping. Target machine implementations of these applications are often restricted in the amount of code-space they may take. The services needed for loading and dumping should be provided in a small, easily implemented set of commands.

o There should be a means for communicating exceptions and errors from the target LDP process to the host process.

o LDP should allow the application to implement a full set of debugging functions without crippling the performance of the target's application (i.e., PSN, PAD, gateway). For example, a breakpoint mechanism that halts the target machine while breakpoint commands are sent from the host to the target is of limited usefulness, since the target will be unable to service the real-time demands of its application.

2.2 Relation to Other Protocols

LDP is an application protocol that fits into the layered internet protocol environment. Figure 1 illustrates the place of LDP in the protocol hierarchy.

```
+-------------------------------+
|              LDP              |      Application
+-------------------------------+      Layer
      |                  |
      |                  |
      |                  |
+----------+       +----------+
|   RDP    |  or   |   TCP    |        Transport Layer
+----------+       +----------+
  | or |                |
  |    |                |
  |  +--------------------+
  |  |  Internet Protocol |            Internetwork
  |  +--------------------+            Layer
  |             |
+-------------------------------+
|    Network Access Protocol    |      Network Layer
+-------------------------------+
```

Relation to Other Protocols
Figure 1

2.2.1 Transport Service Requirements

LDP requires that the underlying transport layer:

- o allow connections to be opened by specifying a network (or internet) address. Support passive and active opens.
- o for each connection, specify the maximum message size.
- o provide a mechanism for sending and receiving messages over an open connection.
- o deliver messages reliably and in sequence
- o support multiple connections, and distinguish messages associated with different connections. This is only a requirement where LDP is expected to support several users at the same time.
- o explictly return the outcome (success/failure) of each request (open, send, receive), and provide a means of querying the status of a connection (unacknowledged message count, etc.).

Data is passed from the application program to the LDP user process in the form of commands. In the case of an LDP server process, command responses originate in LDP itself. Below LDP is the transport protocol. The Reliable Data Protocol (RDP -- RFC 908) is the recommended transport procotol. Data is passed across the LDP/RDP interface in the form of messages. (TCP may be used in place of RDP, but it will be less efficient and it will require more resources to implement.) An internet layer (IP) normally comes between RDP and the network layer, but RDP may exchange data packets directly with the network layer.

Figure 2 shows the flow of data across the protocol interfaces:

```
              +-------+
              |       |
              |Appli- |
              |cation |
              |       |
              +-------+
                  ^
      Commands    |
                  V
              +-------+
              |       |
              |  LDP  |
              |       |
              +-------+
                  ^
      Messages    |
                  V
              +------+
              |      |
              | RDP  |
              |      |
              +------+
                  ^
      Segments    |
                  V
               +-----+
               |     |
               | IP  |
               |     |
               +-----+
                  ^
      Datagrams   |
                  V
            ?   *      !
         $   =      ^     +
            *
         >     Internet
          ,               ?
                !     )
            *   %      $
```

Form of Data Exchange Between Layers
Figure 2

CHAPTER 3

Protocol Operation

3.1 Overview

An LDP session consists of an exchange of commands and responses between an LDP user process and an LDP server process. Normally, the user process resides on a host machine (a timesharing computer used for network monitoring and control), and the server process resides on a target machine (PSN, PAD, gateway, etc.). Throughout this document, host and target are used as synonyms for user process and server process, respectively, although in some implementations (the Butterfly, for example) this correspondence may be reversed. The host controls the session by sending commands to the target. Some commands elicit responses, and all commands may elicit an error reply.

The protocol contains five classes of commands: protocol, data transfer, management, control and breakpoint. Protocol commands are used to verify the command sequencing mechanism and to handle erroneous commands. Data transfer commands involve the transfer of data from one place to another, such as for memory examine/deposit, or loading. Management commands are used for creating and deleting objects (processes, breakpoints, watchpoints, etc.) in the target machine. Control commands are used to control the execution of target code and breakpoints. Breakpoint commands are used to control the execution of commands inside breakpoints and watchpoints.

3.2 Session Management

An LDP session consists of a series of commands sent from a host LDP to a target LDP, some of which may be followed by responses from the target. A session begins when a host opens a transport connection to a target listening on a well known port. LDP uses RDP port number zzz or TCP port number yyy. When the connection has been established, the host sends a HELLO command, and the target replies with a HELLO_REPLY. The HELLO_REPLY contains parameters that describe the target's implementation of LDP, including protocol version, implementation level, system type, and address format. The session terminates when the host closes the underlying transport connection. When the target detects that the transport connection has been closed, it should deallocate any resources dedicated to the session.

The target process is the passive partner in an LDP session, and it waits for the host process to terminate the session. As an implementation consideration, either LDP or the underlying transport protocol in the target should have a method for detecting if the host process has died. Otherwise, an LDP target that supported only one connection could be rendered useless by a host that crashed in the middle of a session. The problem of detecting half-dead connections can be avoided by taking a different tack: the target could allow new connections to usurp inactive connections. A connection with no activity could be declared 'dead', but would not be usurped until the connection resource was needed. However, this would still require the transport layer to support two connection channels: one to receive connection requests, and another to use for an active connection.

3.3 Command Sequencing

Each command sent from the host to the target has a sequence number. The sequence number is used by the target to refer to the command in normal replies and error replies. To save space, these numbers are not actually included in host commands. Instead, each command sent from the host is assigned an implicit sequence number. The sequence number starts at zero at the beginning of the LDP session and increases by one for each command sent. The host and target each keep track of the current number. The SYNCH <sequence number> command may be used by the host to synchronize the sequence number.

3.4 Data Packing and Transmission

The convention for the order of data packing was chosen for its simplicity: data are packed most significant bit first, in order of increasing target address, into eight-bit octets. The octets of packed data are transmitted in sequential order.

Data are always packed according to the address format of the target machine. For example, in an LDP session between a 20-bit host and a 16-bit target, 16-bit words (packed into octets) are transmitted in both directions. For ease of discussion, targets are treated here as if they have uniform address spaces. In practice, the size of address units may vary within a target -- 16-bit macromemory, 32-bit micromemory, 10-bit dispatch memory, etc. Data packing between host and target is tailored to the units of the current target address space.

Figures showing the packing of data for targets with various address unit sizes are given below. The order of transmission with respect to the diagrams is top to bottom. Bit numbering in the following diagrams refers to significance in the octet: bit zero is the least significant bit in an octet. For an explanation of the bit numbering convention that applies in the rest of this document, please see Appendix A.

The packing of data for targets with word lengths that are multiples of 8 is straightforward. The following diagram illustrates 16-bit packing:

```
                 7                              0
                 --------------------------------
   Octet 0      |      WORD 0 bits 15-08        |
                 --------------------------------
   Octet 1      |      WORD 0 bits 07-00        |
                 --------------------------------
   Octet 2      |      WORD 1 bits 15-08        |
                 --------------------------------
   Octet 3      |      WORD 1 bits 07-00        |
                 --------------------------------
                                *
                                *
                                *
                 --------------------------------
   Octet 2n-1   |      WORD n bits 07-00        |
                 --------------------------------
```

Packing of 16-bit Words
Figure 3

Packing for targets with peculiar word lengths is more complicated. For 20-bit machines, 2 words of data are packed into 5 octets. When an odd number of 20-bit words are transmitted, the partially used octet is included in the length of the command, and the octet is padded to the right with zeroes.

```
                 7                                 0
                 -----------------------------------
     Octet 0     |        WORD 0 bits 19-12        |
                 -----------------------------------
     Octet 1     |        WORD 0 bits 11-04        |
                 -----------------------------------
     Octet 2     | WORD 0 03-00 |  WORD 1 19-16    |
                 -----------------------------------
     Octet 3     |        WORD 1 bits 15-08        |
                 -----------------------------------
     Octet 4     |        WORD 1 bits 07-00        |
                 -----------------------------------
```

Packing of 20-bit Words
Figure 4

3.5 Implementations

A subset of LDP commands may be implemented in targets where machine resources are limited and the full capabilities of LDP are not needed. There are three basic levels of target implementations: LOADER_DUMPER, BASIC_DEBUGGER and FULL_DEBUGGER. The target communicates its LDP implementation level to the host during session initiation. The implementation levels are described below:

LOADER_DUMPER

Used for loading/dumping of the target machine. Includes all protocol class commands and replies; data transfer commands READ, WRITE, MOVE and their responses; control command START and control reply EXCEPTION. Understands at least PHYS_MACRO and HOST addressing modes; others if desired.

BASIC_DEBUGGER

Implements LOADER_DUMPER commands, all control commands, all addressing modes appropriate to the target machine, but does not have finite state machine (FSM) breakpoints or watchpoints. Default breakpoints are implemented. The target understands long addressing mode.

FULL_DEBUGGER

Implements all commands and addressing modes appropriate to the target machine, and includes breakpoint commands, conditional commands and BREAKPOINT_DATA. Watchpoints are optional.

CHAPTER 4

Commands and Formats

4.1 Packet Format

LDP commands are enclosed in RDP transport messages. An RDP message may contain more than one command, but each command must fit entirely within a single message. Network packets containing LDP commands have the format shown in Figure 5.

```
+----------------+
| Local Network  |
|   Header(s)    |
+----------------+
|   IP Header    |
+----------------+
|   RDP Header   |
+----------------+     +-+
|  LDP Command   |       |
|  Header        |       |
+----------------+       |
|  Optional      |       |
.  LDP           .       | LDP Command
.  Data          .       | Format
|                |       |
+----------------+       |
|  LDP Padding   |       |
+----------------+     +-+
|  Additional    |
.  LDP           .
.  Commands      .
.                .
+----------------+
```

Network Packet Format
Figure 5

4.2 Command Format

LDP commands consist of a standard two-word header followed optionally by additional data. To facilitate parsing of multi-command messages, all commands contain an even number of octets. Commands that contain an odd number of data octets must be padded with a null octet.

The commands defined by the LDP specification are intended to be of universal application to provide a common basis for all implementations. Command class and type codes from 0 to 63. are reserved by the protocol. Codes above 63. are available for the implementation of target-specific commands.

4.2.1 Command Header

LDP commands begin with a fixed length header. The header specifies the type of command and its length in octets.

```
      0               0 0   1           1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |     Command Length (octets)   |
     +---------------+---------------+
   1 | Command Class | Command Type  |
     +---------------+---------------+
```

LDP Command Header Format
Figure 6

HEADER FIELDS:

Command Length

The command length gives the total number of octets in the command, including the length field and data, and excluding padding.

Command Class
Command Type

The command class and type together specify a particular command. The class selects one of six command categories, and the type gives the command within that category. All codes are decimal. The symbols given in Figures 7 and 8 for command classes and types are used in the remainder of this document for reference.

The command classes that have been defined are:

Command Class	Symbol
1	PROTOCOL
2	DATA_TRANSFER
3	CONTROL
4	MANAGEMENT
5	BREAKPOINT
6	CONDITION
7 - 63	<reserved>

Command Classes
Figure 7

Command type codes are assigned in order of expected frequency of use. Commands and their responses/replies are numbered sequentially. The command types, ordered by command class, are:

Command Class	Command Type	Symbol
PROTOCOL	1	HELLO
	2	HELLO_REPLY
	3	SYNCH
	4	SYNCH_REPLY
	5	ERROR
	6	ERRACK
	7	ABORT
	8	ABORT_DONE
	9 - 63	<reserved>
DATA_TRANSFER	1	WRITE
	2	READ
	3	READ_DONE
	4	READ_DATA
	5	MOVE
	6	MOVE_DONE
	7	MOVE_DATA
	8	REPEAT_DATA
	9	BREAKPOINT_DATA
	10	WRITE_MASK
	11 - 63	<reserved>
CONTROL	1	START
	2	STOP
	3	CONTINUE
	4	STEP
	5	REPORT
	6	STATUS
	7	EXCEPTION
	8 - 63	<reserved>
MANAGEMENT	1	CREATE
	2	CREATE_DONE
	3	DELETE
	4	DELETE_DONE
	5	LIST_ADDRESSES
	6	ADDRESS_LIST
	7	GET_PHYS_ADDRESS
	8	GOT_PHYS_ADDRESS
	9	GET_OBJECT
	10	GOT_OBJECT
	11	LIST_BREAKPOINTS
	12	BREAKPOINT_LIST

	13	LIST_NAMES
	14	NAME_LIST
	15	LIST_PROCESSES
	16	PROCESS_LIST
	17 - 63	<reserved>
BREAKPOINT	1	INCREMENT
	2	INC_COUNT
	3	OR
	4	SET_PTR
	5	SET_STATE
	6 - 63	<reserved>
CONDITION	1	CHANGED
	2	COMPARE
	3	COUNT_EQ
	4	COUNT_GT
	5	COUNT_LT
	6	TEST
	7 - 63	<reserved>

Command Types
Figure 8

4.3 Addressing

Addresses are used in LDP commands to refer to memory locations, processes, buffers, breakpoints and other entities. Many of these entities are machine-dependent; some machines have named objects, some machines have multiple address spaces, the size of address spaces varies, etc. The format for specifying addresses needs to be general enough to handle all of these cases. This speaks for a large, hierarchically structured address format. However, the disadvantage of a large format is that it imposes extra overhead on communication with targets that have simpler address schemes.

LDP resolves this conflict by employing two address formats: a short three-word format for addressing simpler targets, and a long five-word format for others. Each target LDP is required to implement at least one of these formats. At the start of an LDP session, the target specifies the address format(s) it uses in the Flag field of the HELLO_REPLY message. In each address, the first bit of the mode octet is a format flag: 0 indicates LONG address format, and 1 indicates SHORT format.

4.3.1 Long Address Format

The long address format is five words long and consists of a three-word address descriptor and a two-word offset (see Figure 9). The descriptor specifies an address space to which the offset is applied. The descriptor is subdivided into several fields, as described below. The structuring of the descriptor is designed to support complex addressing modes. For example, on targets with multiple processes, descriptors may reference virtual addresses, registers, and other entities within a particular process.

The addressing modes defined below are intended as a base to which target-specific modes may be added. Modes up to 63. are reserved by the protocol. The range 64. to 127. may be used for target-specific address modes.

```
          Long Format - Format bit is LONG=0

           0                   0 0 1           1
           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
          +-------------------------------+   +-+
          |0|      Mode      | Mode Arg   |   |
          +-------------------------------+   |
          |                     (31-16)   |   | Descriptor
          +----          ID           ----+   |
          |                     (15-0)    |   |
          +-------------------------------+   +-+
          |                     (31-16)   |   |
          +----        Offset         ----+   | Offset
          |                     (15-0)    |   |
          +-------------------------------+   +-+
```

Long Address Format
Figure 9

LONG ADDRESS FIELDS:

Mode

The address mode identifies the type of address space being referenced. The mode is qualified by the mode argument and the ID field. Implementation of modes other than physical and host is machine-dependent. Currently defined modes and the address space they reference are shown in Figure 10.

Mode	Symbol	Address space
0	HOST	Host
1	PHYS_MACRO	Macromemory
2	PHYS_MICRO	Micromemory
3	PHYS_I/O	I/O space
4	PHYS_MACRO_PTR	Macro contains a pointer
5	PHYS_REG	Register
6	PHYS_REG_OFFSET	Register plus offset
7	PHYS_REG_INDIRECT	Register contains address of a pointer
8	PROCESS_CODE	Process code space
9	PROCESS_DATA	Process data space
10	PROCESS_DATA_PTR	Process data contains a ptr
11	PROCESS_REG	Process virtual register
12	PROCESS_REG_OFFSET	Process register plus offset
13	PROCESS_REG_INDIRECT	Process register contains address of a pointer
14	OBJECT_OFFSET	Memory object (queue, pool)
15	OBJECT_HEADER	System header for an object
16	BREAKPOINT	Breakpoint
17	WATCHPOINT	Watchpoint
18	BPT_PTR_OFFSET	Breakpoint ptr plus offset
19	BPT_PTR_INDIRECT	Breakpoint ptr plus offset gives address of a pointer
20 - 63	<reserved>	

Long Address Modes
Figure 10

Mode Argument

Provides a numeric argument to the mode field. Specifies the register in physical and process REG and REG_OFFSET modes.

ID Field

Identifies a particular process, buffer or object.

Offset

The offset into the linear address space defined by the mode. The size of the machine word determines the number of significant bits in the offset. Likewise, the addressing units of the target are the units of the offset.

The interpretation of the mode argument, ID field and offset for each address mode is given below:

HOST

The ID and offset fields are numbers assigned arbitrarily by the host side of the debugger. These numbers are used in MOVE and MOVE_DATA messages. MOVE_DATA responses containing this mode as the destination are sent by the target to the host. This may occur in debugging when data is sent to the host from the target breakpoint.

PHYS_MACRO

The offset contains the 32-bit physical address of a location in macromemory. The mode argument and ID field are not used. For example, mode=PHYS_MACRO and offset=1000 specifies location 1000 in physical memory.

PHYS_MICRO

Like PHYS_MACRO, but the location is in micromemory.

PHYS_I/O

Like PHYS_MACRO, but the location is in I/O space.

PHYS_MACRO_PTR

The offset contains the address of a pointer in macromemory. The location pointed to (the effective address) is also in macromemory. The mode argument and ID field are unused.

PHYS_REG

The mode argument gives the physical register. If the register is used by the LDP target process, then the saved copy from the previous context is used. This comment applies to PHYS_REG_OFFSET mode as well. The ID field is not used.

PHYS_REG_OFFSET

The offset is added to the contents of a register given as the mode argument. The result is used as a physical address in macromemory. ID is unused.

PHYS_REG_INDIRECT

The register specified in the mode arg contains the address of a pointer in macromemory. The effective address is the macromemory location specified in the pointer, plus the offset. The ID field is unused.

PROCESS_CODE

The ID is a process ID, the offset is into the code space for this process. Mode argument is not used.

PROCESS_DATA

The ID is a process ID, the offset is into the data space for this process. Mode argument is not used. On systems that do not distinguish between code and data space, these two modes are equivalent, and reference the virtual address space of the process.

PROCESS_DATA_PTR

The offset contains the address of a pointer in the data space of the process specified by the ID. The location pointed to (the effective address) is also in the data space. The mode argument is not used.

PROCESS_REG

Accesses the registers (and other system data) of the process given by the ID field. Mode argument 0 starts the registers. After the registers, the mode argument is an offset into the system area for the process.

PROCESS_REG_OFFSET

The offset plus the contents of the register given in the mode argument specifies a location in the data space of the process specified by the ID.

PROCESS_REG_INDIRECT

The register specified in the mode arg contains the address of a pointer in the data space of the process given by the ID. The effective address is the location in process data space specified in the pointer, plus the offset.

OBJECT_OFFSET (optional)

The offset is into the memory space defined by the object ID in ID. Recommended for remote control of parameter segments.

OBJECT_HEADER (optional)

The offset is into the system header for the object specified by the ID. Intended for use with the Butterfly.

BREAKPOINT

The descriptor specifies a breakpoint. The offset is never used, this type is only used in descriptors referring to breakpoints. (See Breakpoints and Watchpoints, below, for an explanation of breakpoint descriptors.)

WATCHPOINT

The descriptor specifies a watchpoint. The offset is never used, this type is only used in descriptors referring to watchpoints. (See Breakpoints and Watchpoints, below, for an explanation of watchpoint descriptors).

BPT_PTR_OFFSET

For this mode and BPT_PTR_INDIRECT, the mode argument specifies one of two breakpoint pointer variables local to the breakpoint in which this address occurs. These pointers and the SET_PTR command which manipulates them provide for an arbitrary amount of address indirection. They are intended for use in traversing data structures: for example, chasing queues. In BPT_PTR_OFFSET, the offset is added to the pointer variable to give the effective address. In targets which support multiple processes, the location is in the data space of the process given by the ID. Otherwise, the location is a physical address in macro-memory. BPT_PTR.* modes are valid only in breakpoints and watchpoints.

BPT_PTR_INDIRECT

Like BPT_PTR_OFFSET, except that it uses one more level of indirection. The pointer variable given by the mode argument plus the offset specify an address which points to the effective address. See the description of BPT_PTR_OFFSET for a discussion of usage, limitations and address space.

4.3.2 Short Address Format

The short address format is intended for use in implementations where protocol overhead must be minimized. This format is a subset of the long address format: it contains the same fields except for the ID field. Therefore, the short addressing format supports only HOST and PHYS_* address modes. Only the LOADER_DUMPER implementation level commands may be used with the short addressing format. The short address format is three words long, consisting of a 16-bit word describing the address space, and a 32-bit offset.

```
                Short Format - Format bit is SHORT=1

                 0             0 0   1         1
                 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
                +-------------------------------+
                |1|       Mode   | Mode Argument |
                +-------------------------------+   +-+
                |                   (31-16)     |   |
                +----        Offset          ---+   | Offset
                |                   (15-0)      |   |
                +-------------------------------+   +-+

                       Short Address Format
                            Figure 11
```

SHORT ADDRESS FIELDS:
Mode

The high-order bit is 1, indicating the short address format. A list of the address modes supported is given belov. The interpretation of the remaining fields is as described above for the long addressing format.

Mode	Symbol	Address space
0	HOST	Host
1	PHYS_MACRO	Macro-memory
2	PHYS_MICRO	Micro-memory
3	PHYS_I/O	I/O space
4	PHYS_MACRO_PTR	Macro contains a pointer
5	PHYS_REG	Register
6	PHYS_REG_OFFSET	Register plus offset
7	PHYS_REG_INDIRECT	Register contains address of a pointer
8 - 32	<reserved>	

Short Address Modes
Figure 12

CHAPTER 5

Protocol Commands

Protocol commands are used for error handling, for synchronizing the command sequence number, and for communicating protocol implementation parameters. Every protocol command has a corresponding reply. All protocol commands are sent from the host to the target, with replies flowing in the opposite direction.

5.1 HELLO Command

The HELLO command is sent by the host to signal the start of an LDP session. The target responds with HELLO_REPLY.

```
        0             0 0   1         1
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
       +---------------+---------------+
     0 |               4               |
       +---------------+---------------+
     1 |   PROTOCOL    |     HELLO     |
       +---------------+---------------+
```

HELLO Command Format
Figure 13

5.2 HELLO_REPLY

A HELLO_REPLY is sent by the target in response to the HELLO command at the start of an LDP session. This reply is used to inform the host about the target's implementation of LDP.

```
           0             0 0   1         1
           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
          +---------------+---------------+
        0 |              10               |
          +---------------+---------------+
        1 |   PROTOCOL    |  HELLO_REPLY  |
          +---------------+---------------+
        2 |  LDP Version  |  System Type  |
          +---------------+---------------+
        3 |  Options |W|S| Implementation|
          +---------------+---------------+
        4 | Address Code  |   Reserved    |
          +---------------+---------------+
```

HELLO_REPLY Format
Figure 14

HELLO_REPLY FIELDS:

LDP Version

The target's LDP protocol version. If the current host protocol version does not agree with the target's protocol version, the host may terminate the session, or may continue it, at the discretion of the implementor. The current version number is 2.

System Type

The type of system running on the target. This is used as a check against what the host thinks the target is. The host is expected to have a table of target system types with information about target address spaces, target-specific commands and addressing modes, and so forth.

Currently defined system types are shown in Figure 15. This list includes some systems normally thought of as 'hosts' (e.g. C70, VAX), for implementations where targets actively initiate and direct a load of themselves.

Code	System	Description
1	C30_16_BIT	BBN 16-bit C30
2	C30_20_BIT	BBN 20-bit C30
3	H316	Honeywell-316
4	BUTTERFLY	BBN Butterfly
5	PDP-11	DEC PDP-11
6	C10	BBN C10
7	C50	BBN C50
8	PLURIBUS	BBN Pluribus
9	C70	BBN C70
10	VAX	DEC VAX
11	MACINTOSH	Apple MacIntosh

System Types
Figure 15

Address Code

The address code indicates which LDP address format(s) the target is prepared to use. Address codes are show in Figure 16.

Address Code	Symbol	Description
1	LONG_ADDRESS	Five word address format. Supports all address modes and commands.
2	SHORT_ADDRESS	Three word address format. Supports only physical and host address modes. Only the LOADER_DUMPER set of commands are supported.

Target Address Codes
Figure 16

The implementation level specifies which features of the protocol are implemented in the target. There are three levels of protocol implementation. These levels are intended to correspond to the three most likely applications of LDP: simple loading and dumping, basic debugging, and full debugging. (Please see Implementations, above, for a detailed description of implementation levels.) There are are also several optional features that are not included in any particular level.

Implementation levels are cumulative, that is, each higher level includes the features of all previous levels. The levels are shown in Figure 17.

Feature Level	Symbol	Description
1	LOADER_DUMPER	Loader/dumper subset of LDP
2	BASIC_DEBUGGER	Control commands, CREATE
3	FULL_DEBUGGER	FSM breakpoints

Feature Levels
Figure 17

Options

The options field (see Figure 18) is an eight-bit flag field. Bit flags are used to indicate if the target has implemented particular optional commands. Not all optional commands are referenced in this field. Commands whose implementation depends on target machine features are omitted. The LDP application is expected to 'know' about target features that are not intrinsic to the protocol. Examples of target-dependent commands are commands that refer to named objects (CREATE, LIST_NAMES).

Mask	Symbol	Description
1	STEP	The STEP command is implemented
2	WATCHPOINTS	Watchpoints are implemented

Options
Figure 18

5.3 SYNCH Command

The SYNCH command is sent by the host to the target. The target responds with a SYNCH_REPLY. The SYNCH - SYNCH_REPLY exchange serves two functions: it synchronizes the host-to-target implicit sequence number and acts as a cumulative acknowledgement of the receipt and execution of all host commands up to the SYNCH.

```
      0             0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |               6               |
     +---------------+---------------+
   1 |   PROTOCOL    |     SYNCH     |
     +---------------+---------------+
   2 |        Sequence Number        |
     +---------------+---------------+
```

SYNCH Command Format
Figure 19

SYNCH FIELDS:

Sequence Number

The sequence number of this command. If this is not what the target is expecting, the target will reset to it and respond with an ERROR reply.

5.4 SYNCH_REPLY

A SYNCH_REPLY is sent by the target in reponse to a valid SYNCH command. A SYNCH command is valid if its sequence number agrees with the sequence number the target is expecting. Otherwise, the target will reset its sequence number to the SYNCH command and send an ERROR reply.

```
     0             0 0   1         1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +---------------+---------------+
  0 |               6               |
    +---------------+---------------+
  1 |   PROTOCOL    |  SYNCH_REPLY  |
    +---------------+---------------+
  2 |        Sequence Number        |
    +---------------+---------------+
```

SYNCH_REPLY Format
Figure 20

SYNCH_REPLY FIELDS:

Sequence Number

The sequence number of the SYNCH command to which this SYNCH_REPLY is the response.

5.5 ABORT Command

The ABORT command is sent from the host to abort all pending operations at the target. The target responds with ABORT_DONE. This is primarily intended to stop large data transfers from the target. A likely application would be during a debugging session when the user types an interrupt to abort a large printout of data from the target. The ABORT command has no effect on any breakpoints or watchpoints that may be enabled in the target.

As a practical matter, the ABORT command may be difficult to implement on some targets. Its ability to interrupt command processing on the target depends on the target being able to look ahead at incoming commands and receive an out-of-band signal from the host. However, the effect of an ABORT may be achieved by simply closing and reopening the transport connection.

```
      0               0 0     1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |               4               |
     +---------------+---------------+
   1 |   PROTOCOL    |     ABORT     |
     +---------------+---------------+
```

ABORT Command Format
Figure 21

5.6 ABORT_DONE Reply

The ABORT_DONE reply is sent from the target to the host in response to an ABORT command. This indicates that the target has terminated all operations that were pending when the ABORT command was received. The sequence number of the ABORT command is included in the reply.

```
             0                   0 0     1           1
             0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
            +---------------+---------------+
         0  |               4               |
            +---------------+---------------+
         1  |   PROTOCOL    |  ABORT_DONE   |
            +---------------+---------------+
         2  |        Sequence Number        |
            +---------------+---------------+
```

ABORT_DONE Reply Format
Figure 22

ABORT_DONE FIELDS:

Sequence Number

The sequence number of the ABORT command that elicited this reply. This enables the host to distinguish between replies to multiple aborts.

5.7 ERROR Reply

The ERROR reply is sent by the target in response to a bad command. The ERROR reply gives the sequence number of the offending command and a reason code. The target ignores further commands until an ERRACK command is received. The reason for ignoring commands is that the proper operation of outstanding commands may be predicated on the execution of the erroneous command.

```
          0             0 0   1         1
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
         +---------------+---------------+
       0 |        Command Length         |
         +---------------+---------------+
       1 |   PROTOCOL    |     ERROR     |
         +---------------+---------------+
       2 |   Command Sequence Number     |
         +---------------+---------------+
       3 |          Error code           |
         +---------------+---------------+
       4 |        Optional Data          |
         +---------------+---------------+
                         *
                         *
                         *
         +---------------+---------------+
       n |        Optional Data          |
         +---------------+---------------+
```

ERROR Reply Format
Figure 23

ERROR Reply FIELDS:

Command Sequence Number

The implicit sequence number of the erroneous command.

Error Code

A code specifying what error has taken place. The currently defined codes are shown in Figure 24.

Error Code	Symbol
1	BAD_COMMAND
2	BAD_ADDRESS_MODE
3	BAD_ADDRESS_ID
4	BAD_ADDRESS_OFFSET
5	BAD_CREATE_TYPE
6	NO_RESOURCES
7	NO_OBJECT
8	OUT_OF_SYNCH
9	IN_BREAKPOINT

ERROR Codes
Figure 24

An explanation of each of these error codes follows:
BAD_COMMAND

The command was not meaningful to the target machine. This includes commands that are valid but unimplemented in this target. Also, the command was not valid in this context. For example, a command given by the host that is only legal in a breakpoint (e.g. IF, SET_STATE).

BAD_ADDRESS_MODE <offending-address>

The mode of an address given in the command is not meaningful to this target system. For example, a PROCESS address mode on a target that does not support multi-processing.

BAD_ADDRESS_ID <offending-address>

The ID field of an address didn't correspond to an appropriate thing. For example, for a PROCESS address mode, the ID of a non-existent process.

BAD_ADDRESS_OFFSET <offending-address>

The offset field of the address was outside the legal range for the thing addressed. For example, an offset of 200,000 in PHYS_MACRO mode on a target with 64K of macro-memory.

BAD_CREATE_TYPE

The object type in a CREATE command was unknown.

NO_RESOURCES

A CREATE command failed due to lack of necessary resources.

NO_OBJECT

A GET_OBJECT command failed to find the named object.

OUT_OF_SYNCH

The sequence number of the SYNCH command was not expected by the target. The target has resynchronized to it.

IN_BREAKPOINT <breakpoint-descriptor> <breakpoint-sequence#> <reason-code> [<optional-info>]

An error occurred within a breakpoint command list. The given 16-bit sequence-number refers to the sequence number of the CREATE command that created the breakpoint, while breakpoint-sequence# refers to the sequence number of the command within the breakpoint given by <breakpoint-descriptor>.

5.8 ERRACK Acknowledgement

An ERRACK is sent by the host in response to an ERROR reply from the target. The ERRACK is used to acknowledge that the host has received the ERROR reply.

```
          0                   0 0     1               1
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
         +---------------+---------------+
       0 |               4               |
         +---------------+---------------+
       1 |   PROTOCOL    |    ERRACK     |
         +---------------+---------------+
```

ERRACK Command Format
Figure 25

CHAPTER 6

Data Transfer Commands

Data transfer commands transfer data between the host and the target. These commands are used for loading and dumping the target, and examining and depositing locations on the target. The READ command reads data from the target, the MOVE command moves data within the target or from the target to another entity, and the WRITE command writes data to the target. REPEAT_DATA makes copies of a pattern to the target -- it is useful for zeroing memory. WRITE_MASK writes data with a mask, and is intended for modifying target parameter tables.

Data transmitted to and from the target always contains a target address. In writes to the target, this is used as the destination of the data. In reads from the target, the target address is used by the host to identify where in the target the data came from. In addition, the MOVE command may contain a 'host' address as its destination; this permits the host to further discriminate between possible sources of data from the target -- from different breakpoints, debugging windows, etc.

A read request to the target may generate one or more response messages. In particular, responses to requests for large amounts of data -- core dumps, for example -- must be broken up into multiple messages, if the block of data requested plus the LDP header exceeds the transport layer message size.

In commands which contain data (WRITE, READ_DATA, MOVE_DATA and REPEAT_DATA), if there are an odd number of data octets, then a null octet is appended. This is so that the next command in the message, if any, will begin on an even octet. The command length is the sum of the number of octets in the command header and the number of octets of data, excluding the null octet, if any.

The addressing formats which may be used with data transfer commands are specified for each LDP session at the start of the session by the target in the HELLO_REPLY response. See the section entitled 'Addressing', above, for a description of LDP addressing formats and modes. In the command diagrams given below, the short addressing format is illustrated. For LDP sessions using long addressing, addresses are five words long, instead of three words, as shown here. In both addressing modes, descriptors are three words and offsets are two words.

6.1 WRITE Command

The WRITE command is used to send octets of data from the host to the target. This command specifies the address in the target where the data is to be stored, followed by a stream of data octets. If the data stream contains an odd number of octets, then a null octet is appended so that the next command, if any, will begin on an even octet. Since LDP must observe message size limitations imposed by the underlying transport layer, a single logical write may need to be broken up into multiple WRITEs in separate transport messages.

```
     0             0 0   1         1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +---------------+---------------+
  0 |        Command Length         |
    +---------------+---------------+
  1 | DATA_TRANSFER |     WRITE     |
    +---------------+---------------+
  2 |                               |
    +--          Target           --+
  3 |            Start              |
    +--          Address          --+
  4 |                               |
    +---------------+---------------+
  5 |  Data Octet   |  Data Octet   |
    +---------------+---------------+
                    *
                    *
                    *
    +---------------+---------------+
  n |  Data Octet   | Data or Null  |
    +---------------+---------------+
```

WRITE Command Format
Figure 26

WRITE FIELDS:

Command Length

The command length gives the number of octets in the command, including data octets, but excluding the padding octet, if any.

Target Start Address

This is the address to begin storing data in the target. The length of the data to be stored may be inferred by the target from the command length. An illegal address or range will generate an ERROR reply.

Data Octets

Octets of data to be stored in the target. Data are packed according to the packing convention described above. Ends with a null octet if there are an odd number of data octets.

6.2 READ Command

The host uses the READ command to ask the target to send back a contiguous block of data. The data is specified by a target starting address and a count. The target returns the data in one or more READ_DATA commands, which give the starting address (in the target) of each segment of returned data. When the transfer is completed, the target sends a READ_DONE command to the host.

```
          0             0 0   1         1
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
         +---------------+---------------+
       0 |              14               |
         +---------------+---------------+
       1 | DATA_TRANSFER |     READ      |
         +---------------+---------------+
       2 |                               |
         +--          Target           --+
       3 |            Start              |
         +--          Address          --+
       4 |                               |
         +---------------+---------------+
       5 |            Address            |
         +--          Unit             --+
       6 |            Count              |
         +---------------+---------------+
```

READ Command Format
Figure 27

READ FIELDS:

Target Start Address

The starting address of the requested block of target data. The target sends an ERROR reply if the starting address is illegal, if the ending address computed from the sum of the start and the count is illegal, or if holes are encountered in the middle of the range.

Address Unit Count

The count of the number of target indivisibly-addressable units to be transferred. For example, if the address space is PHYS_MACRO, a count of two and a start address of 1000 selects the contents of locations 1000 and 1001. 'Count' is used instead of 'length' to avoid the problem of determining units the length should be denominated in (octets, words, etc.). The size and type of the unit will vary depending on the address space selected by the target start address. The target should reply with an error (if it is able to determine in advance of a transfer) if the inclusive range of addresses specified by the start address and the count contains an illegal or nonexistent address.

6.3 READ_DATA Response

The target uses the READ_DATA response to transmit data requested by a host READ command. One or more READ_DATA responses may be needed to fulfill a given READ command, depending on the size of the data block requested and the transport layer message size limits. Each READ_DATA response gives the target starting address of its segment of data. If the response contains an odd number of data octets, the target ends the response with a null octet.

```
         0                             0   0       1               1
         0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5
        +-------------------------------+-------------------------------+
      0 |                        Command Length                         |
        +-------------------------------+-------------------------------+
      1 |         DATA_TRANSFER         |           READ_DATA           |
        +-------------------------------+-------------------------------+
      2 |                                                               |
        +--                           Target                          --+
      3 |                              Start                            |
        +--                           Address                         --+
      4 |                                                               |
        +-------------------------------+-------------------------------+  +-+
      5 |           Data Octet          |           Data Octet          |    |
        +-------------------------------+-------------------------------+    |
                                        *                                    |
                                        *                                    | Data
                                        *                                    |
        +-------------------------------+-------------------------------+    |
      n |           Data Octet          |          Data or Null         |    |
        +-------------------------------+-------------------------------+  +-+
```

DATA Response Format
Figure 28

READ_DATA FIELDS:

Command Length

The command length gives the number of octets in the command, including data octets, but excluding the padding octet, if any. The host can calculate the length of the data by subtracting the header length from the command length. Since the target address may be either three words (short format) or five words (long format), the address mode must be checked to determine which is being used.

Target Start Address

This is the starting address of the data segment in this message. The host may infer the length of the data from the command length. The address format (short or long) is the same as on the initial READ command.

Data Octets

Octets of data from the target. Data are packed according to the packing convention described above. Ends with a null octet if there are an odd number of data octets.

6.4 READ_DONE Reply

The target sends a READ_DONE reply to the host after it has finished transferring the data requested by a READ command. READ_DONE specifies the sequence number of the READ command.

```
      0             0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |               6               |
     +---------------+---------------+
   1 | DATA_TRANSFER |   READ_DONE   |
     +---------------+---------------+
   2 |      READ Sequence Number     |
     +---------------+---------------+
```

READ_DONE Reply Format
Figure 29

READ_DONE FIELDS:

READ Sequence Number

The sequence number of the READ command this is a reply to.

6.5 MOVE Command

The MOVE command is sent by the host to move a block of data from the target to a specified destination. The destination address may specify a location in the target, in the host, or in another target (for loading one target from another). The data is specified by a target starting address and an address unit count. The target sends an ERROR reply if the starting address is illegal, if the ending address computed from the sum of the start and the count is illegal, or if holes are encountered in the middle of the range. If the MOVE destination is off-target, the target moves the data in one or MOVE_DATAs. Other commands arriving at the target during the transfer should be processed in a timely fashion, particularly the ABORT command. When the data has been moved, the target sends a MOVE_DONE to the host. However, a MOVE within a breakpoint will not generate a MOVE_DONE.

A MOVE with a host destination differs from a READ in that it contains a host address. This field is specified by the host in the MOVE command and copied by the target into the responding MOVE_DATA(s). The address may be used by the host to differentiate data returned from multiple MOVE requests. This information may be useful in breakpoints, in multi-window debugging and in communication with targets with multiple processors. For example, the host sends the MOVE command to the target to be executed during a breakpoint. The ID field in the host address might be an index into a host breakpoint table. When the breakpoint executes, the host would use the ID to associate the returning MOVE_DATA with this breakpoint.

```
         0                   0 0     1           1
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
        +---------------+---------------+
     0  |         Command Length        |
        +---------------+---------------+
     1  | DATA_TRANSFER |     MOVE      |
        +---------------+---------------+
     2  |                               |
        +--          Source           --+
     3  |            Start              |
        +--          Address          --+
     4  |                               |
        +---------------+---------------+
     5  |            Address            |
        +--          Unit             --+
     6  |            Count              |
        +---------------+---------------+
     7  |                               |
        +--          Destination      --+
     8  |            Start              |
        +--          Address          --+
     9  |                               |
        +---------------+---------------+
```

MOVE Command Format
Figure 30

MOVE FIELDS:

Source Start Address

The starting address of the requested block of target data. An illegal address type will generate an error reply.

Address Unit Count

The count of the number of target indivisibly-addressable units to be transferred. For example, if the address space is PHYS_MACRO, a count of two and a start address of 1000 selects the contents of locations 1000 and 1001. 'Count' is used instead of 'length' to avoid the problem of determining units the length should be denominated in (octets, words, etc.). The size and type of the unit will vary depending on the address space selected by the target start address. The target should reply with an error (if it is able to determine in advance of a transfer) if the inclusive range of addresses specified by the start address and the count contains an illegal or nonexistent address.

Destination Address

The destination of the MOVE. If the address space is on the target, the address unit size should agree with that of the source address space. If the address mode is HOST, the values and interpretations of the remaining address fields are arbitrary, and are determined by the host implementation. For example, the mode argument might specify a table (breakpoint, debugging window, etc.) and the ID field an index into the table.

6.6 MOVE_DATA Response

The target uses the MOVE_DATA responses to transmit data requested by a host MOVE command. One or more MOVE_DATA responses may be needed to fulfill a given MOVE command, depending on the size of the data block requested and the transport layer message size limits. Each MOVE_DATA response gives the target starting address of its segment of data. If the response contains an odd number of data octets, the target should end the response with a null octet.

```
           0             0 0     1         1
           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
          +---------------+---------------+
        0 |        Command Length         |
          +---------------+---------------+
        1 | DATA_TRANSFER |   MOVE_DATA   |
          +---------------+---------------+
        2 |                               |
          +--          Source           --+
        3 |            Start              |
          +--          Address          --+
        4 |                               |
          +---------------+---------------+
        5 |                               |
          +--          Destination      --+
        6 |            Start              |
          +--          Address          --+
        7 |                               |
          +---------------+---------------+    +-+
        8 |  Data Octet   |  Data Octet   |     |
          +---------------+---------------+     |
                          *                     |
                          *                     | Data
                          *                     |
          +---------------+---------------+     |
        n |  Data Octet   | Data or Null  |     |
          +---------------+---------------+    +-+
```

MOVE_DATA Response Format
Figure 31

MOVE_DATA FIELDS:

Command Length

The command length gives the number of octets in the command, including data octets, but excluding the padding octet, if any.

Source Start Address

This is the starting address of the data segment in this message. The host may infer length of the data from the command length.

Destination Address

The destination address copied from the MOVE command that initiated this transfer. In the case of HOST MOVEs, this is used by the host to identify the source of the data.

Data Octets

Octets of data from the target. Data are packed according to the packing convention described above. Ends with a null octet if there are an odd number of data octets.

6.7 MOVE_DONE Reply

The target sends a MOVE_DONE reply to the host after it has finished transferring the data requested by a MOVE command. MOVE_DONE specifies the sequence number of the MOVE command.

```
      0             0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |               6               |
     +---------------+---------------+
   1 | DATA_TRANSFER |   MOVE_DONE   |
     +---------------+---------------+
   2 |      MOVE Sequence Number     |
     +---------------+---------------+
```

MOVE_DONE Reply Format
Figure 32

MOVE_DONE FIELDS:

MOVE Sequence Number

The sequence number of the MOVE command this is a reply to.

6.8 REPEAT_DATA

The REPEAT_DATA command is sent by the host to write copies of a specified pattern into the target. This provides an efficient way of zeroing target memory and initializing target data structures. The command specifies the target starting address, the number of copies of the pattern to be made, and a stream of octets that constitutes the pattern.

This command differs from the other data transfer commands in that the effect of a REPEAT_DATA with a large pattern cannot be duplicated by sending the data in smaller chunks over several commands. Therefore, the maximum size of a pattern that can be copied with REPEAT_DATA will depend on the message size limits of the transport layer.

```
         0             0 0   1         1
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
        +---------------+---------------+
      0 |        Command Length         |
        +---------------+---------------+
      1 | DATA_TRANSFER | REPEAT_DATA   |
        +---------------+---------------+
      2 |                               |
        +--           Target          --+
      3 |             Start             |
        +--           Address         --+
      4 |                               |
        +---------------+---------------+
      6 |          Repeat Count         |
        +---------------+---------------+  +-+
      7 |  Data Octet   |  Data Octet   |   |
        +---------------+---------------+   |
                        *                   |
                        *                   | Pattern
                        *                   |
        +---------------+---------------+   |
      n |  Data Octet   | Data or Null  |   |
        +---------------+---------------+  +-+
```

REPEAT_DATA Command Format
Figure 33

REPEAT_DATA FIELDS:

Command Length

The command length gives the number of octets in the command, including data octets in the pattern, but excluding the padding octet, if any.

Target Start Address

This is the starting address where the first copy of the pattern should be written in the target. Successive copies of the pattern are made contiguously starting at this address.

Repeat Count

The repeat count specifies the number of copies of the pattern that should be made in the target. The repeat count should be greater than zero.

Pattern

The pattern to be copied into the target, packed into a stream of octets. Data are packed according to the packing convention described above. Ends with a null octet if there are an odd number of data octets.

6.9 WRITE_MASK Command (Optional)

The host sends a WRITE_MASK command to the target to write one or more masked values. The command uses an address to specify a target base location, followed by one or more offset-mask-value triplets. Each triplet gives an offset from the base, a value, and a mask indicating which bits in the location at the offset are to be changed.

This optional command is intended for use in controlling the target by changing locations in a table. For example, it may be used to change entries in a target parameter table. The operation of modifying a specified location with a masked value is intended to be atomic. In other words, another target process should not be able to access the location to be modified between the start and the end of the modification.

```
     0             0 0   1         1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +-------------------------------+
  0 |        Command Length         |
    +---------------+---------------+
  1 | DATA_TRANSFER |  WRITE_MASK   |
    +---------------+---------------+
  2 |                               |
    +--           Target          --+
  3 |             Base              |
    +--           Address         --+
  4 |                               |
    +---------------+---------------+    +-+
  5 |                               |     |
    +--           Offset          --+     |
  6 |                               |     |
    +---------------+---------------+     | Offset-Mask-Valu
  7 |                               |     | Triplet
    +--           Mask            --+     |
  8 |                               |     |
    +---------------+---------------+     |
  9 |                               |     |
    +--           Value           --+     |
 10 |                               |     |
    +---------------+---------------+    +-+
                    .
                    .
                    .
    +---------------+---------------+    +-+
    |                               |     |
    +--           Offset          --+     |
    |                               |     |
    +---------------+---------------+     | Offset-Mask-Valu
    |                               |     | Triplet
    +--           Mask            --+     |
    |                               |     |
    +---------------+---------------+     |
    |                               |     |
    +--           Value           --+     |
    |                               |     |
    +---------------+---------------+    +-+
```

WRITE_MASK Format
Figure 34

WRITE_MASK FIELDS:

Command Length

The command length gives the number of octets in the command. The number of offset-value pairs may be calculated from this, since the command header is either 10 or 12 octets long (short or long address format), and each offset-mask-value triplet is 12 octets long.

Target Base Address

Specifies the target location to which the offset is added to yield the location to be modified.

Offset

An offset to be added to the base to select a location to be modified.

Mask

Specifies which bits in the value are to be copied into the location.

Value

A value to be stored at the specified offset from the base. The set bits in the mask determine which bits in the value are applied to the location. The following algorithm will achieve the intended result: take the one's complement of the mask and AND it with the location, leaving the result in the location. Then AND the mask and the value, and OR the result into the location.

CHAPTER 7

Control Commands

Control commands are used to control the execution of target code, breakpoints and watchpoints. They are also used to read and report the state of these objects. The object to be controlled or reported on is specified with a descriptor. Valid descriptor modes include PHYS_* (for some commands) PROCESS_CODE, BREAKPOINT and WATCHPOINT. Control commands which change the state of the target are START, STOP, CONTINUE and STEP. REPORT requests a STATUS report on a target object. EXCEPTION is a spontaneous report on an object, used to report asynchronous events such as hardware traps. The host may verify the action of a START, STOP, STEP or CONTINUE command by following it with a REPORT command.

7.1 START Command

The START command is sent by the host to start execution of a specified object in the target. For targets which support multiple processes, a PROCESS_CODE address specifies the process to be started. Otherwise, one of the PHYS_* modes may specify a location in macro-memory where execution is to continue. Applied to a breakpoint or watchpoint, START sets the value of the object's state variable, and activates the breakpoint. The breakpoint counter and pointer variables are initialized to zero.

```
               0               0 0   1           1
               0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
              +---------------+---------------+
            0 |              14               |
              +---------------+---------------+
            1 |    CONTROL    |     START     |
              +---------------+---------------+  +-+
            2 |     Mode      |       0       |  |
              +---------------+---------------+  |
            3 |                               |  |
              +--            ID             --+  |
            4 |             Field             |  |   Address
              +-------------------------------+  |
            5 |                               |  |
              +--           Offset          --+  |
            6 |                               |  |
              +-------------------------------+  +-+
```

START Command Format
Figure 35

START FIELDS:

Address

The descriptor specifies the object to be started. If the mode is PROCESS_CODE, ID specifies the process to be started, and offset gives the process virtual address to start at. If the mode is PHYS_*, execution of the target is continued at the specified address.

For modes of BREAKPOINT and WATCHPOINT, the offset specifies the new value of the FSM state variable. This is for FSM breakpoints and watchpoints.

7.2 STOP Command

The STOP command is sent by the host to stop execution of a specified object in the target. A descriptor specifies the object. Applied to a breakpoint or watchpoint, STOP deactivates it. The breakpoint/watchpoint may be re-activated by issuing a START or a CONTINUE command for it.

```
        0             0 0   1         1
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
       +---------------+---------------+
     0 |              10               |
       +---------------+---------------+
     1 |   CONTROL     |     STOP      |
       +---------------+---------------+  +-+
     2 |    Mode       |    0          |   |
       +---------------+---------------+   |
     3 |                               |   |  Descriptor
       +--             ID            --+   |
     4 |              Field            |   |
       +-------------------------------+  +-+
```

STOP Command Format
Figure 36

STOP FIELDS:

Descriptor

The descriptor specifies the object to be stopped or disarmed. If the mode is PROCESS_CODE, the ID specifies the process to be stopped.

For modes of BREAKPOINT and WATCHPOINT, the specified breakpoint or watchpoint is deactivated. It may be re-activated by a CONTINUE or START command.

7.3 CONTINUE Command

The CONTINUE command is sent by the host to resume execution of a specified object in the target. A descriptor specifies the object. Applied to a breakpoint or watchpoint, CONTINUE activates it.

```
             0             0 0   1           1
             0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
            +---------------+---------------+
          0 |              10               |
            +---------------+---------------+
          1 |   CONTROL     |   CONTINUE    |
            +---------------+---------------+  +-+
          2 |    Mode       |      0        |  |
            +---------------+---------------+  |
          3 |                               |  |  Descriptor
            +--             ID            --+  |
          4 |              Field            |  |
            +-------------------------------+  +-+
```

CONTINUE Command Format
Figure 37

CONTINUE FIELDS:

Descriptor

The descriptor specifies the object to be resumed or armed. If the mode is PROCESS_CODE, the ID specifies the process to be resumed.

For modes of BREAKPOINT and WATCHPOINT, the specified breakpoint or watchpoint is armed.

7.4 STEP Command

The STEP command is sent by the host to the target. It requests the execution of one instruction (or appropriate operation) in the object specified by the descriptor.

```
             0             0 0   1         1
             0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
            +---------------+---------------+
          0 |              10               |
            +---------------+---------------+
          1 |    CONTROL    |     STEP      |
            +---------------+---------------+  +-+
          2 |     Mode      |       0       |   |
            +---------------+---------------+   |
          3 |                               |   |  Descriptor
            +--             ID            --+   |
          4 |              Field            |   |
            +-------------------------------+  +-+
```

STEP Command Format
Figure 38

STEP FIELDS:

Descriptor

The descriptor specifies the object to be stepped. If the mode is PROCESS_CODE, the ID specifies a process.

7.5 REPORT Command

The REPORT command is sent by the host to request a status report on a specified target object. The status is returned in a STATUS reply.

```
                0               0 0   1         1
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
               +---------------+---------------+
             0 |              10               |
               +---------------+---------------+
             1 |    CONTROL    |    REPORT     |
               +---------------+---------------+  +-+
             2 |     Mode      |       0       |  |
               +---------------+---------------+  |
             3 |                               |  |  Descriptor
               +--            ID             --+  |
             4 |             Field             |  |
               +-------------------------------+  +-+
```

REPORT Command Format
Figure 39

REPORT FIELDS:

Descriptor

The descriptor specifies the object for which a STATUS report is requested. For a mode of PROCESS_CODE, the ID specifies a process. Other valid modes are PHYS_MACRO, to query the status of the target application, and BREAKPOINT and WATCHPOINT, to get the status of a breakpoint or watchpoint.

7.6 STATUS Reply

The target sends a STATUS reply in response to a REPORT command from the host. STATUS gives the state of a specified object. For example, it may tell whether a particular target process is running or stopped.

```
         0             0 0   1         1
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
        +---------------+---------------+
     0  |         Command Length        |
        +---------------+---------------+
     1  |    CONTROL    |     STATUS    |
        +---------------+---------------+  +-+
     2  |     Mode      |       0       |  |
        +---------------+---------------+  |
     3  |                               |  |  Descriptor
        +--             ID            --+  |
     4  |              Field            |  |
        +-------------------------------+  +-+
     5  |             Status            |
        +-------------------------------+  +-+
                        *                  |
                        *                  |
                        *                  |  Other Data
        +-------------------------------+  |
     n  |          Other Data           |  |
        +-------------------------------+  +-+
```

STATUS Reply Format
Figure 40

STATUS FIELDS:

Descriptor

The descriptor specifies the object whose status is being given. If the mode is PROCESS_CODE, then the ID specifies a process. If the mode is PHYS_MACRO, then the status is that of the target application.

Status

The status code describes the status of the object. Status codes are 0=STOPPED and 1=RUNNING. For breakpoints and watchpoints, STOPPED means disarmed and RUNNING means armed.

Other Data

For breakpoints and watchpoints, Other Data consists of a

16-bit word giving the current value of the FSM state variable.

7.7 EXCEPTION Trap

An EXCEPTION is a spontaneous message sent from the target indicating a target-machine exception associated with a particular object. The object is specified by an address.

```
     0             0 0   1           1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +---------------+---------------+
  0 |        Command Length         |
    +---------------+---------------+
  1 |   CONTROL     |   EXCEPTION   |
    +---------------+---------------+  +-+
  2 |    Mode       |       0       |  |
    +---------------+---------------+  |
  3 |                               |  |
    +--            ID             --+  |
  4 |             Field             |  |  Address
    +-------------------------------+  |
  5 |                               |  |
    +--          Offset           --+  |
  6 |                               |  |
    +-------------------------------+  +-+
  7 |             Type              |
    +-------------------------------+  +-+
                    *                  |
                    *                  |
                    *                  |  Other Data
    +-------------------------------+  |
  n |          Other Data           |  |
    +-------------------------------+  +-+
```

EXCEPTION Format
Figure 41

EXCEPTION FIELDS:

Address

The address specifies the object the exception is for.

Type

The type of exception. Values are target-dependent.

Other Data

Values are target-dependent.

CHAPTER 8

Management Commands

Management commands are used to control resources in the target machine. There are two kinds of commands: those that interrogate the remote machine about resources, and those that allocate and free resources. There are management commands to create, list and delete breakpoints. All commands have corresponding replies which include the sequence number of the request command. Failing requests produce ERROR replies.

There are two resource allocation commands, CREATE and DELETE, which create and delete objects in the remote machine. There are a number of listing commands for listing a variety of target objects -- breakpoints, watchpoints, processes, and names. The amount of data returned by listing commands may vary in length, depending on the state of the target. If a list is too large to fit in a single message, the target will send it in several list replies. A flag in each reply specifies whether more messages are to follow.

8.1 CREATE Command

The CREATE command is sent from the host to the target to create a target object. If the CREATE is successful, the target returns a CREATE_DONE reply, which contains a descriptor associated with the CREATEd object. The types of objects that may be specified in a CREATE include breakpoints, processes, memory objects and descriptors. All are optional except for breakpoints.

```
            0               0 0             1
            0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
           +-------------------------------+
         0 |         Command Length        |
           +-------------------------------+
         1 | MANAGEMENT    |    CREATE     |
           +-------------------------------+
         2 |          Create Type          |
           +-------------------------------+   +-+
                           *                   |
                           *                   |  Create
                           *                   |  Arguments
           +-------------------------------+   |
         n |        Create Arguments       |   |
           +-------------------------------+   +-+
```

CREATE Command Format
Figure 42

CREATE FIELDS:

Create Type

The type of object to be created. Arguments vary with the type. Currently defined types are shown in Figure 43. All are optional except for BREAKPOINT.

Create Type	Symbol
0	BREAKPOINT
1	WATCHPOINT
2	PROCESS
3	MEMORY_OBJECT
4	DESCRIPTOR

Create Types
Figure 43

Create Arguments

Create arguments depend on the type of object being created. The formats for each type of object are described below.

```
           0             0 0   1         1
           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
          +---------------+---------------+
        0 |              22               |
          +---------------+---------------+
        1 |  MANAGEMENT   |    CREATE     |
          +---------------+---------------+
        2 |          BREAKPOINT           |
          +---------------+---------------+  +-+
        3 |     Mode      | Mode Argument |   |
          +---------------+---------------+   |
        4 |                               |   |
          +--             ID            --+   |  Create
        5 |            Field              |   |  BREAKPOINT
          +-------------------------------+   |  Arguments
        6 |                               |   |
          +--          Offset           --+   |
        7 |                               |   |
          +-------------------------------+   |
        8 |        Maximum States         |   |
          +---------------+---------------+   |
        9 |         Maximum Size          |   |
          +---------------+---------------+   |
       10 |    Maximum Local Variables    |   |
          +---------------+---------------+  +-+
```

CREATE BREAKPOINT Format
Figure 44

BREAKPOINT and WATCHPOINT

The format is the same for CREATE BREAKPOINT and CREATE WATCHPOINT. In the following discussion, 'breakpoint' may be taken to mean either breakpoint or watchpoint.

The address is the location where the breakpoint is to be set. In the case of watchpoints it is the location to be watched. Valid modes are any PHYS_* mode that addresses macro-memory, PROCESS_CODE for breakpoints and PROCESS_DATA for watchpoints.

'Maximum states' is the number of states the finite state machine for this breakpoint will have. A value of zero indicates a default breakpoint, for targets which do not implement finite state machine (FSM) breakpoints. A default breakpoint is the same as an FSM with one state consisting of a STOP and a REPORT command for the process containing the breakpoint.

'Maximum size' is the total size, in octets, of the breakpoint data to be sent via subsequent BREAKPOINT_DATA commands. This is the size of the data only, and does not include the LDP command headers and breakpoint descriptors.

'Maximum local variables' is the number of 32-bit longs to reserve for local variables for this breakpoint. Normally this value will be zero.

PROCESS

Creates a new process. Arguments are target-dependent.

```
    0             0 0   1         1
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
   +---------------+---------------+
 0 |        Command Length         |
   +---------------+---------------+
 1 |   MANAGEMENT  |    CREATE     |
   +---------------+---------------+
 2 |         MEMORY_OBJECT         |
   +---------------+---------------+
 3 |          Object Size          |
   +---------------+---------------+
 4 |           Name Size           |
   +-------------------------------+  +-+
 5 |   Name char   |   Name char   |   |
   +-------------------------------+   |
                   *                   |  Object
                   *                   |  Name
                   *                   |
   +---------------+---------------+   |
 n | 0 or Name char|       0       |   |
   +---------------+---------------+  +-+
```

CREATE MEMORY_OBJECT Format
Figure 45

MEMORY_OBJECT

Creates an object of size Object Size, with the given name. Object Size is in target dependent units. The name may be the null string for unnamed objects. Name Size gives the number of characters in Object Name, and must be even. Always ends with a null octect.

DESCRIPTOR

Used for obtaining descriptors from IDs on target systems where IDs are longer than 32 bits. There is a single argument, Long ID, whose length is target dependent.

8.2 CREATE_DONE Reply

The target sends a CREATE_DONE reply to the host in response to a successful CREATE command. The reply contains the sequence number of the CREATE request, and a descriptor for the object created. This descriptor is used by the host to specify the object in subsequent commands referring to it. Commands which refer to created objects include LIST_* commands, DELETE and BREAKPOINT_DATA. For example, to delete a CREATEd object, the host sends a DELETE command that specifies the descriptor returned by the CREATE_DONE reply.

```
          0             0 0   1         1
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
         +---------------+---------------+
       0 |              12               |
         +---------------+---------------+
       1 |  MANAGEMENT   |  CREATE_DONE  |
         +---------------+---------------+
       2 |    Create Sequence Number     |
         +---------------+---------------+  +-+
       3 |     Mode      | Mode Argmuent |   |
         +---------------+---------------+   |  Created
       4 |                               |   |  Object
         +--            ID             --+   |  Descriptor
       5 |             Field             |   |
         +---------------+---------------+  +-+
```

CREATE_DONE Reply Format
Figure 46

CREATE_DONE FIELDS:

Create Sequence Number

The sequence number of the CREATE command to which this is the reply.

Created Object Descriptor

A descriptor assigned by the target to the created object. The contents of the descriptor fields are arbitrarily assigned by the target at its convenience. The host treats the descriptor as a unitary object, used for referring to the created object in subsequent commands.

8.3 DELETE Command

The host sends a DELETE command to remove an object created by an earlier CREATE command. The object to be deleted is specified with a descriptor. The descriptor is from the CREATE_DONE reply to the original CREATE command.

```
     0             0 0   1         1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +---------------+---------------+
  0 |              10               |
    +---------------+---------------+
  1 |  MANAGEMENT   |    DELETE     |
    +---------------+---------------+  +-+
  2 |     Mode      | Mode Argument |  |
    +---------------+---------------+  |
  3 |                               |  | Created
    +--            ID            --+  | Object
  4 |           Field               |  | Descriptor
    +---------------+---------------+  +-+
```

DELETE Command Format
Figure 47

DELETE FIELDS:

Created Object Descriptor

Specifies the object to be deleted. This is the descriptor that was returned by the target in the CREATE_DONE reply to the original CREATE command.

8.4 DELETE_DONE Reply

The target sends a DELETE_DONE reply to the host in response to a successful DELETE command. The reply contains the sequence number of the DELETE request.

```
       0               0 0   1         1
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
      +---------------+---------------+
    0 |               6               |
      +---------------+---------------+
    1 |  MANAGEMENT   |  DELETE_DONE  |
      +---------------+---------------+
    2 |    Delete Sequence Number     |
      +---------------+---------------+
```

DELETE_DONE Reply Format
Figure 48

DELETE_DONE FIELDS:

Request Sequence Number

The sequence number of the DELETE command to which this is the reply.

8.5 LIST_ADDRESSES Command

The host sends a LIST_ADDRESSES command to request a list of valid address ranges for a specified object. The object is given by a descriptor. Typical objects are a target process, or the target physical machine. The target responds with an ADDRESS_LIST reply. This command is used for obtaining the size of dynamic address spaces and for determining dump ranges.

```
                0             0 0   1         1
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
               +---------------+---------------+
             0 |              10               |
               +---------------+---------------+
             1 |   MANAGEMENT  | LIST_ADDRESSES|
               +---------------+---------------+  +-+
             2 |     Mode      | Mode Argument |   |
               +---------------+---------------+   |  Object
             3 |                               |   |  Descriptor
               +--             ID            --+   |
             4 |              Field            |   |
               +---------------+---------------+  +-+
```

LIST_ADDRESSES Command Format
Figure 49

LIST_ADDRESSES FIELDS:

Object Descriptor

Specifies the object whose address ranges are to be listed. Valid modes include PHYS_MACRO, PHYS_MICRO, PROCESS_CODE, and PROCESS_DATA.

8.6 ADDRESS_LIST Reply

The target sends an ADDRESS_LIST reply to the host in response to a successful LIST_ADDRESSES command. The reply contains the sequence number of the LIST_ADDRESSES request, the descriptor of the object being listed, and a list of the valid address ranges within the object.

```
       0                   0 0   1           1
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
      +---------------+---------------+
   0  |         Command Length        |
      +---------------+---------------+
   1  |   MANAGEMENT  | ADDRESS_LIST  |
      +---------------+---------------+
   2  |      List Sequence Number     |
      +---------------+---------------+
   3  |    Flags    |M|  Item Count   |
      +---------------+---------------+
   4  |                               |
      +--                           --+
   5  |           Descriptor          |
      +--                           --+
   6  |                               |
      +---------------+---------------+  +-+
   7  |                               |  |
      +--        First Address      --+  |  First
   8  |                               |  |  Address
      +-------------------------------+  |  Range
   9  |                               |  |
      +--         Last Address      --+  |
   10 |                               |  |
      +-------------------------------+  +-+
                      *
                      *
                      *
      +---------------+---------------+  +-+
      |                               |  |
      +--        First Address      --+  |  Last
      |                               |  |  Address
      +-------------------------------+  |  Range
      |                               |  |
      +--         Last Address      --+  |
      |                               |  |
      +-------------------------------+  +-+
```

ADDRESS_LIST Reply Format
Figure 50

ADDRESS_LIST FIELDS:

List Sequence Number

The sequence number of the LIST_ADDRESSES command to which this is the reply.

Flags

If M=1, the address list is continued in one or more subsequent ADDRESS_LIST replies. If M=0, this is the final ADDRESS_LIST.

Item Count

The number of address ranges described in this command.

Descriptor

The descriptor of the object being listed.

Address Range

Each address range is composed of a pair of 32-bit addresses which give the first and last addresses of the range. If there are 'holes' in the address space of the object, then multiple address ranges will be used to describe the valid address space.

8.7 LIST_BREAKPOINTS Command

The host sends a LIST_BREAKPOINTS command to request a list of all breakpoints associated with the current connection. The target replies with BREAKPOINT_LIST.

```
     0             0 0   1         1
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
    +---------------+---------------+
 0  |               4               |
    +---------------+---------------+
 1  |   MANAGEMENT  |LIST_BREAKPOINTS
    +---------------+---------------+
```

LIST_BREAKPOINTS Command Format
Figure 51

8.8 BREAKPOINT_LIST Reply

The target sends a BREAKPOINT_LIST reply to the host in response to a LIST_BREAKPOINTS command. The reply contains the sequence number of the LIST_BREAKPOINTS request, and a list of all breakpoints associated with the current connection. The descriptor and address of each breakpoint are listed.

```
                0             0 0   1         1
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
               +---------------+---------------+
             0 |         Command Length        |
               +---------------+---------------+
             1 |   MANAGEMENT  |BREAKPOINT_LIST|
               +---------------+---------------+
             2 |      List Sequence Number     |
               +---------------+---------------+
             3 |   Flags     |M|  Item Count   |
               +---------------+---------------+   +-+
             4 |     Mode      |       0       |   |
               +---------------+---------------+   |
             5 |                               |   |  Breakpoint
               +--             ID            --+   |  Descriptor
             6 |              Field            |   |
               +---------------+---------------+   +-+
             7 |     Mode      | Mode Argument |   |
               +---------------+---------------+   |
             8 |                               |   |
               +--             ID            --+   |  Breakpoint
             9 |              Field            |   |  Address
               +-------------------------------+   |
            10 |                               |   |
               +--            Offset         --+   |
            11 |                               |   |
               +-------------------------------+   +-+
                               *                     |  Additional
                               *                     |  Descriptor-Addre
                               *                     |  Pairs
                                                   +-+
```

BREAKPOINT_LIST Reply Format
Figure 52

BREAKPOINT_LIST FIELDS:

List Sequence Number

The sequence number of the LIST_BREAKPOINTS command to which this is the reply.

Flags

If M=1, the breakpoint list is continued in one or more subsequent BREAKPOINT_LIST replies. If M=0, this is the final BREAKPOINT_LIST.

Item Count

The number of breakpoints described in this list.

Breakpoint Descriptor

A descriptor assigned by the target to this breakpoint. Used by the host to specify this breakpoint in BREAKPOINT_DATA and DELETE commands.

Breakpoint Address

The address at which this breakpoint is set.

8.9 LIST_PROCESSES Command

The host sends a LIST_PROCESSES command to request a list of descriptors for all processes on the target. The target replies with PROCESS_LIST.

```
           0             0 0   1         1
           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
          +-------------------------------+
        0 |               4               |
          +---------------+---------------+
        1 |   MANAGEMENT  |LIST_PROCESSES |
          +---------------+---------------+
```

LIST_PROCESSES Command Format
Figure 53

8.10 PROCESS_LIST Reply

The target sends a PROCESS_LIST reply to the host in response to a LIST_PROCESSES command. The reply contains the sequence number of the LIST_PROCESSES request, and a list of all processes in the target. For each process, a descriptor and a target-dependent amount of process data are given.

```
                  0             0 0   1         1
                  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
                 +---------------+---------------+
              0  |         Command Length        |
                 +---------------+---------------+
              1  |   MANAGEMENT  | PROCESS_LIST  |
                 +---------------+---------------+
              2  |      List Sequence Number     |
                 +---------------+---------------+
              3  |    Flags    |M| Item Count    |
                 +---------------+---------------+   +-+
              4  | PROCESS_CODE  |       0       |    |
                 +---------------+---------------+    |
              5  |                               |    |  Process
                 +--            ID             --+    |  Descriptor
              6  |             Field             |    |
                 +---------------+---------------+   +-+
              7  |      Process data count       |    |
                 +---------------+---------------+    |
              8  | Process data  | Process data  |    |
                 +-------------------------------+    |  Process
                                 *                    |  Data
                                 *                    |
                                 *                    |
                 +---------------+---------------+    |
              n  | Process data  | Process data  |    |
                 +-------------------------------+   +-+
                                 *                    |  Additional
                                 *                    |  Descriptor-Data
                                 *                    |  Pairs
                                                     +-+
```

PROCESS_LIST Reply Format
Figure 54

PROCESS_LIST FIELDS:

List Sequence Number

The sequence number of the LIST_PROCESSES command to which this is the reply.

Flags

If M=1, the process list is continued in one or more subsequent PROCESS_LIST replies. If M=0, this is the final PROCESS_LIST.

Item Count

The number of processes described in this list. For each process there is a descriptor and a variable number of octets of process data.

Process Descriptor

A descriptor assigned by the target to this process. Used by the host to specify this PROCESS in a DELETE command.

Process Data Count

Number of octets of process data for this process. Must be even.

Process Data

Target-dependent information about this process. Number of octets is given by the process data count.

8.11 LIST_NAMES Command

The host sends a LIST_NAMES command to request a list of available names as strings. The target replies with NAME_LIST.

```
           0             0 0   1         1
           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
          +-------------+-+-------------+
       0  |                4                |
          +-------------+-+-------------+
       1  |   MANAGEMENT   |   LIST_NAMES   |
          +-------------+-+-------------+
```

LIST_NAMES Command Format
Figure 55

8.12 NAME_LIST Reply

The target sends a NAME_LIST reply to the host in response to a LIST_NAMES command. The reply contains the sequence number of the LIST_NAMES request, and a list of all target names, as strings.

```
                0             0 0   1         1
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
               +---------------+---------------+
             0 |         Command Length        |
               +---------------+---------------+
             1 |  MANAGEMENT   |  NAME_LIST    |
               +---------------+---------------+
             2 |      List Sequence Number     |
               +---------------+---------------+
             3 |  Flags      |M|  Item Count   |
               +---------------+---------------+   +-+
             4 |           Name Size           |   |
               +---------------+---------------+   |
             5 |  Name Char    |   Name Char   |   |  Name
               +---------------+---------------+   |  String
                               *                   |
                               *                   |
                               *                   |
               +---------------+---------------+   |
             n | 0 or Name Char|       0       |   |
               +---------------+---------------+   +-+
                               *                   |  Additional
                               *                   |  Name
                               *                   |  Strings
                                                   +-+
```

NAME_LIST Reply Format
Figure 56

NAME_LIST FIELDS:

List Sequence Number

The sequence number of the LIST_NAMES command to which this is the reply.

Flags

If M=1, the name list is continued in one or more subsequent NAME_LIST replies. If M=0, this is the final NAME_LIST.

Item Count

The number of name strings in this list. Each name string consists of a character count and a null-terminated string of characters.

Name Size

The number of octets in this name string. Must be even.

Name Characters

A string of octets composing the name. Ends with a null octet. The number of characters must be even, so if the terminating null comes on an odd octet, another null is appended.

8.13 GET_PHYS_ADDR Command

The host sends a GET_PHYS_ADDR command to convert an address into physical form. The target returns the physical address in a GOT_PHYS_ADDR reply. For example, the host could send a GET_PHYS_ADDR command containing a register-offset address, and the target would return the physical address derived from this in a GOT_PHYS_ADDR reply.

```
      0             0 0   1           1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |              14               |
     +---------------+---------------+
   1 |  MANAGEMENT   | GET_PHYS_ADDR |
     +---------------+---------------+   +-+
   2 |     Mode      | Mode Argument |   |
     +---------------+---------------+   |
   3 |              ID               |   |
     +--           Field           --+   |
   4 |                               |   |  Address
     +---------------+---------------+   |
   5 |                               |   |
     +--           Offset          --+   |
   6 |                               |   |
     +---------------+---------------+   +-+
```

GET_PHYS_ADDR Command Format
Figure 57

GET_PHYS_ADDR FIELDS:

Address

The address to be converted to a physical address. The mode may be one of PHYS_REG_OFFSET, PHYS_REG_INDIRECT, PHYS_MACRO_PTR, any OBJECT_* mode, and any PROCESS_* mode except for PROCESS_REG.

8.14 GOT_PHYS_ADDR Reply

The target sends a GOT_PHYS_ADDR reply to the host in response to a successful GET_PHYS_ADDR command. The reply contains the sequence number of the GET_PHYS_ADDR request, and the specified address converted into a physical address.

```
      0             0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |              16               |
     +---------------+---------------+
   1 |  MANAGEMENT   | GOT_PHYS_ADDR |
     +---------------+---------------+
   2 |     Get  Sequence Number      |
     +---------------+---------------+  +-+
   3 | PHYS_MACRO    |       0       |   |
     +---------------+---------------+   |
   4 |                               |   |
     +--             0             --+   |
   5 |                               |   |  Address
     +---------------+---------------+   |
   6 |                               |   |
     +--           Offset          --+   |
   7 |                               |   |
     +---------------+---------------+  +-+
```

GOT_PHYS_ADDR Reply Format
Figure 58

GOT_PHYS_ADDR FIELDS:

Get Sequence Number

The sequence number of the GET_PHYS_ADDR command to which this is the reply.

Address

The address resulting from translating the address given in the GET_PHYS_ADDR command into a physical address. Mode is always PHYS_MACRO and ID and mode argument are always zero. Offset gives the 32-bit physical address.

8.15 GET_OBJECT Command

The host sends a GET_OBJECT command to convert a name string into a descriptor. The target returns the descriptor in a GOT_OBJECT reply. Intended for use in finding control parameter objects.

```
                 0             0 0   1           1
                 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
                +---------------+---------------+
              0 |        Command Length         |
                +---------------+---------------+
              1 |  MANAGEMENT   |  GET_OBJECT   |
                +---------------+---------------+  +-+
              2 |          Name Size            |  |
                +---------------+---------------+  |
              3 |  Name Char    |   Name Char   |  |  Name
                +---------------+---------------+  |  String
                                *                  |
                                *                  |
                                *                  |
                +---------------+---------------+  |
              n | 0 or Name Char|       0       |  |
                +---------------+---------------+  +-+
```

GET_OBJECT Command Format
Figure 59

GET_OBJECT FIELDS:

Name String

The name of an object.

Name Size

The number of octets in this name string. Must be even.

Name Characters

A string of octets composing the name. Ends with a null octet. The number of characters must be even, so if the terminating null comes on an odd octet, another null is appended.

8.16 GOT_OBJECT Reply

The target sends a GOT_OBJECT reply to the host in response to a successful GET_OBJECT command. The reply contains the sequence number of the GET_OBJECT request, and the specified object name converted into a descriptor.

```
      0               0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |              12               |
     +---------------+---------------+
   1 |  MANAGEMENT   |  GOT_OBJECT   |
     +---------------+---------------+
   2 |      Get  Sequence Number     |
     +---------------+---------------+  +-+
   3 | Mode          | Mode Argument |  |
     +---------------+---------------+  |
   4 |                               |  |
     +--             ID            --+  |  Object
   5 |                               |  |  Descriptor
     +---------------+---------------+  +-+
```

GOT_OBJECT Reply Format
Figure 60

GOT_OBJECT FIELDS:

Get Sequence Number

The sequence number of the GET_OBJECT command to which this is the reply.

Descriptor

The descriptor of the object named in the GET_OBJECT command.

CHAPTER 9

Breakpoints and Watchpoints

Breakpoints and watchpoints are used in debugging applications. Each breakpoint or watchpoint is associated with one debugger connection and one address. When a breakpoint or watchpoint is triggered, the target executes one or more commands associated with it. A breakpoint is triggered when its address is executed. A watchpoint is triggered when its address is modified. The same mechanism is used for structuring breakpoint and watchpoint commands. For brevity's sake, 'breakpoint' will be used in the remainder of this document to refer to either a breakpoint or a watchpoint.

The commands used by the host to manipulate breakpoints are given in Figure 61, in the order in which they are normally used. All commands are sent from the host to the target, and each specifies the descriptor of a breakpoint.

Command	Description
CREATE	Create a breakpoint
BREAKPOINT_DATA	Send commands to be executed in an FSM breakpoint
START	Activate a breakpoint, set state and initialize breakpoint variables
STOP	Deactivate a breakpoint
CONTINUE	Activate a breakpoint
LIST_BREAKPOINTS	List all breakpoints
REPORT	Report the status of a breakpoint
DELETE	Delete a breakpoint

Commands to Manipulate Breakpoints
Figure 61

There are two kinds of breakpoints: default breakpoints and finite state machine (FSM) breakpoints. They differ in their use of commands.

Default breakpoints do not contain any commands. When triggered, a default breakpoint stops the target object (i.e., target process or application) it is located in. A STATUS report on the stopped object is sent to the host. At this point, the host may send further commands to debug the target.

An FSM breakpoint has one or more conditional command lists, organized into a finite state machine. When an FSM breakpoint is created, the total number of states is specified. The host then sends commands (using BREAKPOINT_DATA) to be associated with each state. The target maintains a state variable for the breakpoint, which determines which command list will be executed if the breakpoint is triggered. When the breakpoint is created its state variable is initialized to zero (zero is the first state). A breakpoint command, SET_STATE, may be used within a breakpoint to change the value of the state variable. A REPORT command applied to a breakpoint descriptor returns its address, whether it is armed or disarmed, and the value of its state variable.

Commands valid in breakpoints include all implemented data transfer and control commands, a set of conditional commands, and a set of breakpoint commands. The conditional commands and the breakpoint commands act on a set of local breakpoint variables. The breakpoint variables consist of the state variable, a counter, and two pointer variables. The conditional commands control the execution of breakpoint command lists based on the contents of one of the breakpoint variables. The breakpoint commands are used to set the value of the breakpoint variables: SET_STATE sets the state variable, SET_PTR sets one of the pointer variables, and INC_COUNT increments the breakpoint counter. There may be implementation restrictions on the number of breakpoints, the number of states, the number of conditions, and the size of the command lists. Management commands and protocol commands are forbidden in breakpoints.

In FSM breakpoints, the execution of commands is controlled as follows. When a breakpoint is triggered, the breakpoint's state variable selects a particular state. One or more conditional command lists is associated with this state. A conditional command list consists of a list of conditions followed by a list of commands which are executed if the condition list is satisfied. The debugger starts a breakpoint by executing the first of these lists. If the condition list is satisfied, the debugger executes the associated command list and leaves the breakpoint. If the condition list fails, the debugger skips to the next conditional command list. This process continues until the debugger either encounters a successful condition list, or exhausts all the conditional command lists for the state. The relationship of commands, lists and states is shown in Figure 62 (IFs, THENs and ELSEs are used below to clarify the logical structure within a state; they are not part of the protocol).

```
State 0
        IF <condition list 0>
           THEN <command list 0>

        ELSE IF <condition list 1>
           THEN <command list 1>

        *
        *
        *

        ELSE IF <condition list n>
            THEN <command list n>

        ELSE <exit>
  *
  *
  *
State n
```

Breakpoint Conditional Command Lists
Figure 62

9.1 BREAKPOINT_DATA Command

BREAKPOINT_DATA is a data transfer command used by the host to send commands to be executed in breakpoints and watchpoints. The command specifies the descriptor of the breakpoint or watchpoint, and a stream of commands to be appended to the end of the breakpoint's command list. BREAKPOINT_DATA is applied sequentially to successive breakpoint states, and successive command lists within each state. Multiple BREAKPOINT_DATAs may be sent for a given breakpoint. Breaks between BREAKPOINT_DATA commands may occur anywhere within the data stream, even within individual commands in the data. Sufficient space to store the data must have been allocated by the maximum size field in the CREATE BREAKPOINT/WATCHPOINT command.

```
          0             0 0   1         1
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
         +---------------+---------------+
       0 |        Command Length         |
         +---------------+---------------+
       1 | DATA_TRANSFER |BREAKPOINT_DATA|
         +---------------+---------------+  +-+
       2 |     Mode      | Mode Argument |  |
         +---------------+---------------+  |  Breakpoint or
       3 |                               |  |  Watchpoint
         +--            ID            --+  |  Descriptor
       4 |             Field             |  |
         +-------------------------------+  +-+
       5 |      Data     |     Data      |  |
         +-------------------------------+  |
                         *                  |
                         *                  |  Data
                         *                  |
         +---------------+---------------+  |
       n |      Data     |   Data or 0   |  |
         +---------------+---------------+  +-+
```

BREAKPOINT_DATA Command Format
Figure 63

BREAKPOINT_DATA FIELDS:

Command Length

Total length of this command in octets, including data, excluding the final padding octet, if any.

Data

A stream of data to be appended to the data for this breakpoint or watchpoint. This stream has the form of one or more states, each containing one or more conditional command lists. The first BREAKPOINT_DATA command sent for a breakpoint contains data starting with state zero. The data for each state starts with the state size. A conditional command list is composed of two parts: a condition list, and a command list. Each list begins with a word that gives its size in octets.

```
<state 0 size>
          <condition list 0 size> <condition list 0>
          <command list 0 size>   <command list 0>
                    *
                    *
                    *
          <condition list n size> <condition list n>
          <command list n size>   <command list n>
<state 1 size>
                    <etc>
     *
     *
     *
<state n size>
```

Breakpoint Data Stream Format
Figure 64

Sizes

All sizes are stored in 16-bit words, and include their own length. The state size gives the total number of octets of breakpoint data for the state. The condition list size gives the total octets of breakpoint data for the following condition list. A condition list size of 2 indicates an empty condition list: in this case the following command list is executed unconditionally. The command list size gives the total octets of breakpoint data for the following command list.

Lists

Condition and command lists come in pairs. When the breakpoint occurs, the condition list controls whether the following command list should be executed. A condition list consists of one or more commands from the CONDITION command class. A command list consists one or more LDP commands. Valid commands are any commands from the BREAKPOINT, DATA_TRANSFER or CONTROL command classes.

CHAPTER 10

Conditional Commands

Conditional commands are used in breakpoints to control the execution of breakpoint commands. One or more conditions in sequence form a condition list. If a condition list is satisfied (evaluates to TRUE), the breakpoint command list immediately following it is executed. (See Breakpoints and Watchpoints, above, for a discussion of the logic flow in conditional/command lists.) Conditional commands perform tests on local breakpoint variables, and other locations. Each condition evaluates to either TRUE or FALSE. Figure 65 contains a summary of conditional commands:

Command	Description
CHANGED <loc>	Determine if a location has changed
COMPARE <loc1> <mask> <loc2>	Compare two locations, using a mask
COUNT_[EQ | GT | LT] <value>	Compare the counter to a value
TEST <loc> <mask> <value>	Compare a location to a value

Conditional Command Summary
Figure 65

The rules for forming and evaluating condition lists are:

o consecutive conditions have an implicit logical AND between them. A sequence of such conditions is called an 'and_list'. and_lists are delimited by an OR command and by the end of the condition list.

o the breakpoint OR command may be inserted between any pair of conditions

o AND takes precedence over OR

o nested condition lists are not supported. A condition list is simply one or more and_lists, separated by ORs.

o the condition list is evaluated in sequence until either a TRUE and_list is found (condition list <- TRUE), or the end of the condition list is reached (condition list <- FALSE). An and_list is TRUE if all its conditions are TRUE.

The distillation of these rules into BNF is:

```
<condition_list> := <and_list>  [OR <and_list>]*
<and_list>       := <condition> [AND <condition>]*
<condition>      := CHANGED | COMPARE | COUNT | TEST

where:  OR  is a breakpoint command
        AND is implicit for any pair of consecutive conditions
```

For example, the following condition list, with one command per line,

```
COUNT_EQ 1
OR
COUNT_GT 10
COUNT_LT 20
```

evaluates to:

```
(COUNT = 1) OR (COUNT > 10  AND COUNT < 20)
```

and will cause the command list that follows it to be executed if the counter is equal to one, or is between 10 and 20.

10.1 Condition Command Format

Condition commands start with the standard four-octet command header. The high-order bit of the command type byte is used as a negate flag: if this bit is set, the boolean value of the condition is negated. This flag applies to one condition only, and not to other conditions in the condition list.

```
           0             0 0   1         1
           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
          +---------------+---------------+
        0 |        Command Length         |
          +---------------+---------------+
        1 | CONDITION     |N|    Type     |
          +---------------+---------------+
```

Condition Command Header
Figure 66

10.2 COUNT Conditions

The COUNT conditions (COUNT_EQ, COUNT_GT and COUNT_LT) are used to compare the breakpoint counter to a specified value. The counter is set to zero when the breakpoint is STARTed, and is incremented by the INC_COUNT breakpoint command. The format is the same for the COUNT_EQ, COUNT_GT and COUNT_LT conditions.

```
           0             0 0   1         1
           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
          +---------------+---------------+
        0 |               8               |
          +---------------+---------------+
        1 | CONDITION     |N|  Type       |
          +---------------+---------------+
        2 |                               |
          +--            Value          --+
        3 |                               |
          +---------------+---------------+
```

COUNT Condition Format
Figure 67

COUNT_* Condition FIELDS:

Type

One of COUNT_EQ, COUNT_LT and COUNT_GT. The condition is TRUE if the breakpoint counter is [EQ | LT | GT] the specified value.

Value

A 32-bit value to be compared to the counter.

10.3 CHANGED Condition

The CHANGED condition is TRUE if the contents of the specified location have changed since the last time this breakpoint occurred. Only one location may be specified as the object of CHANGED conditions per breakpoint. The CHANGED condition is always FALSE the first time the breakpoint occurs.

```
      0             0 0   1         1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |              14               |
     +---------------+---------------+
   1 | CONDITION     |N|  CHANGED    |
     +---------------+---------------+
   2 |                               |
     +--                           --+
   3 |            Address            |
     +--                           --+
   4 |                               |
     +--                           --+
   5 |                               |
     +--                           --+
   6 |                               |
     +---------------+---------------+
```

CHANGED Condition
Figure 68

CHANGED FIELDS:

Address

The full 5-word address of the location to be tested by the CHANGED command.

10.4 COMPARE Condition

The COMPARE condition compares two locations using a mask. The condition is TRUE if (<loc1> & <mask>) = (<loc2> & <mask>).

```
          0                     0   0       1                     1
          0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5
        +-------------------------------+-------------------------------+
     0  |                              28                               |
        +-------------------------------+-------------------------------+
     1  | CONDITION                     |N|  COMPARE                    |
        +-------------------------------+-------------------------------+
     2  |                                                               |
        +--                                                           --+
     3  |                           Address 1                           |
        +--                                                           --+
     4  |                                                               |
        +--                                                           --+
     5  |                                                               |
        +--                                                           --+
     6  |                                                               |
        +-------------------------------+-------------------------------+
     7  |                                                               |
        +--                          Mask                             --+
     8  |                                                               |
        +---------------------------------------------------------------+
     9  |                                                               |
        +--                                                           --+
    10  |                           Address 2                           |
        +--                                                           --+
    11  |                                                               |
        +--                                                           --+
    12  |                                                               |
        +--                                                           --+
    13  |                                                               |
        +---------------------------------------------------------------+
```

COMPARE Condition
Figure 69

COMPARE FIELDS:

Address 1
Address 2

The 5-word addresses of the locations to be compared.

Mask

A 32-bit mask specifying which bits in the locations should be compared.

10.5 TEST Condition

The TEST condition is used to compare a location to a value, using a mask. The condition is TRUE if (<loc> & <mask>) = <value>.

```
         0               0 0   1           1
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
        +-------------------------------+
     0  |              22               |
        +---------------+---------------+
     1  | CONDITION     |N|   TEST      |
        +---------------+---------------+
     2  |                               |
        +--                           --+
     3  |            Address            |
        +--                           --+
     4  |                               |
        +--                           --+
     5  |                               |
        +--                           --+
     6  |                               |
        +---------------+---------------+
     7  |                               |
        +--           Mask            --+
     8  |                               |
        +-------------------------------+
     9  |                               |
        +--           Value           --+
     10 |                               |
        +-------------------------------+
```

TEST Condition
Figure 70

TEST FIELDS:

Address

The 5-word address of the location to be compared to the value.

Mask

A 32-bit mask specifying which bits in the location should be compared.

Value

A 32-bit value to compare to the masked location.

CHAPTER 11

Breakpoint Commands

Breakpoint commands are used to set the value of breakpoint variables. These commands are only valid within breakpoints and watchpoints. They are sent from the host to the target as data in BREAKPOINT_DATA commands. Figure 71 contains a summary of breakpoint commands:

Command	Description
INCREMENT <location>	Increment the specified location
INC_COUNT	Increment the breakpoint counter
OR	OR two breakpoint condition lists
SET_PTR <n> <location>	Set pointer <n> to the contents of <location>
SET_STATE <n>	Set the breakpoint state variable to <n>

Breakpoint Command Summary
Figure 71

11.1 INCREMENT Command

The INCREMENT command increments the contents of a specified location. The location may be in any address space writable from LDP.

```
             0             0 0   1         1
             0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
            +---------------+---------------+
          0 |              14               |
            +---------------+---------------+
          1 | BREAKPOINT    |  INCREMENT    |
            +---------------+---------------+
          2 |                               |
            +--                           --+
          3 |           Address             |
            +--                           --+
          4 |                               |
            +--                           --+
          5 |                               |
            +--                           --+
          6 |                               |
            +---------------+---------------+
```

INCREMENT Command Format
Figure 72

INCREMENT FIELDS:

Address

The full address of the location whose contents are to be incremented.

11.2 INC_COUNT Command

The INC_COUNT command increments the breakpoint counter. There is one counter variable for each breakpoint. It is initialized to zero when the breakpoint is created, when it is armed with the START command, and whenever the breakpoint state changes. The counter is tested by the COUNT_* conditions.

```
                0             0 0   1         1
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
               +---------------+---------------+
             0 |               4               |
               +---------------+---------------+
             1 | BREAKPOINT    |   INC_COUNT   |
               +---------------+---------------+
```

INC_COUNT Command Format
Figure 73

11.3 OR Command

The OR command delineates two and_lists in a breakpoint condition list. A condition list is TRUE if any of the OR separated and_lists in it are TRUE. A breakpoint condition list may contain zero, one or, many OR commands. See 'Condition Commands' for an explanation of condition lists.

```
                0             0 0   1         1
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
               +---------------+---------------+
             0 |               4               |
               +---------------+---------------+
             1 | BREAKPOINT    |      OR       |
               +---------------+---------------+
```

OR Command Format
Figure 74

11.4 SET_PTR Command

The SET_PTR command loads the specified breakpoint pointer with the contents of a location. The pointer variables and the SET_PTR command are intended to provide a primitive but unlimited indirect addressing capability. Two addressing modes, BPT_PTR_OFFSET and BPT_PTR_INDIRECT, are used for referencing the breakpoint pointers. For example, to follow a linked list, use SET_PTR to load a pointer with the start of the list, then use successive SET_PTR commands with addressing mode BPT_PTR_OFFSET to get successive elements.

```
                0             0 0     1         1
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
               +---------------+---------------+
            0  |              16               |
               +-------------------------------+
            1  |  BREAKPOINT   |    SET_PTR    |
               +-------------------------------+
            2  |            Pointer            |
               +---------------+---------------+
            3  |                               |
               +--                           --+
            4  |            Address            |
               +--                           --+
            5  |                               |
               +--                           --+
            6  |                               |
               +--                           --+
            7  |                               |
               +---------------+---------------+
```

SET_PTR Command Format
Figure 75

SET_PTR FIELDS:

Pointer

The pointer to be changed. Allowable values are 0 and 1.

Address

The full address of the location whose contents are to be loaded into the given pointer variable.

11.5 SET_STATE Command

The SET_STATE command sets the breakpoint state variable to the specified value. This is the only method of changing a breakpoint's state from within a breakpoint. The breakpoint's state may be also be changed by a START command from the host. The state variable is initialized to zero when the breakpoint is created.

```
       0             0 0   1         1
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
      +---------------+---------------+
    0 |               6               |
      +---------------+---------------+
    1 | BREAKPOINT    |  SET_STATE    |
      +---------------+---------------+
    2 |          State Value          |
      +-------------------------------+
```

SET_STATE Command Format
Figure 76

SET_STATE FIELDS:

State Value

The new value for the breakpoint state variable. Must not be greater than the maximum state value specified in the CREATE BREAKPOINT command that created this breakpoint.

APPENDIX A

Diagram Conventions

Command and message diagrams are used in this document to illustrate the format of these entities. Words are listed in order of transmission down the page. The first word is word zero. Bits within a word run left to right, most significant to least. However, following a convention observed in other protocol documents, bits are numbered in order of transmission; the most significant bit in a word is transmitted first. The bit labelled '0' is the most significant bit.

```
      0             0 0   1           1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
     +---------------+---------------+
   0 |M|                           |L|
     +---------------+---------------+
   1 | Most Sig Octet| Least S. Octet|
     +---------------+---------------+

     M = most significant bit in word zero,
         transmitted first
     L = least significant bit in word zero,
         transmitted last
```

Sample Diagram
Figure 77

APPENDIX B

Command Summary

The following table lists all non-breakpoint LDP commands in alphabetical order, with a brief description of each.

Command	Sender Host	Sender Target	Function
ABORT	X		Abort outstanding commands
ABORT_DONE		X	Acknowledge ABORT
ADDRESS_LIST		X	Return valid address ranges
BREAKPOINT_DATA	X		Send breakpoint commands
BREAKPOINT_LIST		X	Return list of breakpoints
CONTINUE	X		Resume execution
CREATE	X		Create target object
CREATE_DONE		X	Acknowledge CREATE
DELETE	X		Delete target object
DELETE_DONE		X	Acknowledge DELETE
EXCEPTION		X	Report target exception
ERROR		X	Report error with a host command
ERRACK	X		Acknowledge ERROR
GET_OBJECT	X		Get object descriptor from name
GET_PHYS_ADDRESS	X		Get address in physical form
GOT_OBJECT		X	Return object descriptor
GOT_PHYS_ADDRESS		X	Return physical address
HELLO	X		Initiate LDP session
HELLO_REPLY		X	Return LDP parameters
LIST_ADDRESSES	X		Request valid address ranges
LIST_BREAKPOINTS	X		Request breakpoint list
LIST_NAMES	X		Request name list
LIST_PROCESSES	X		Request process list
MOVE	X		Read data from target
MOVE_DONE		X	Acknowledge MOVE completion
MOVE_DATA		X	Send data request by MOVE
NAME_LIST		X	Return name list
PROCESS_LIST		X	Return process list
READ	X		Read data from target
READ_DATA		X	Return data requested by READ
READ_DONE		X	Acknowledge READ completion
REPEAT_DATA	X		Write copies of data
REPORT	X		Request status of object
START	X		Start target object
STATUS		X	Return status of object
STEP	X		Step execution of target object
STOP	X		Stop target object
SYNCH	X		Check sequence number
SYNCH_REPLY		X	Confirm sequence number
WRITE	X		Write data
WRITE_MASK	X		Write data with mask

Command Summary
Figure 78

APPENDIX C

Commands, Responses and Replies

The following table shows the relationship between commands, responses and replies. Commands are sent from the host to the target. Some commands elicit responses and/or replies from the target. Responses and replies are sent from the target to the host. The distinction between them is that the target sends only one reply to a command, but may send multiple responses. Responses always contain data, whereas replies may or may not.

Command	Response	Reply
ABORT		ABORT_DONE
BREAKPOINT_DATA		
CONTINUE		
CREATE		CREATE_DONE
DELETE		DELETE_DONE
GET_OBJECT		GOT_OBJECT
GET_PHYS_ADDRESS		GOT_PHYS_ADDRESS
HELLO		HELLO_REPLY
LIST_ADDRESSES		ADDRESS_LIST
LIST_BREAKPOINTS		BREAKPOINT_LIST
LIST_NAMES		NAME_LIST
LIST_PROCESSES		PROCESS_LIST
MOVE	MOVE_DATA	MOVE_DONE
READ	READ_DATA	READ_DONE
REPEAT_DATA		
REPORT		STATUS
START		
STEP		
STOP		
SYNCH		SYNCH_REPLY
WRITE		
WRITE_MASK		

Commands, Responses and Replies
Figure 79

APPENDIX D

Glossary

FSM

Finite state machine. Commands of each breakpoint or watchpoint are implemented as part of a finite state machine. A list of breakpoint commands is associated with each state. There are several breakpoint commands to change from one state to another.

host

The 'host' in an LDP session is the timesharing system on which the user process runs.

long

A long is a 32-bit quantity.

octet

An octet is an eight-bit quantity.

RDP

The Reliable Data Protocol (RDP) is a transport layer protocol designed as a low-overhead alternative to TCP. RDP is a connection oriented protocol that provides reliable, sequenced message delivery.

server process

The LDP server process is the passive participant in an LDP session. The server process usually resides on a target machine such as a PAD, PSN or gateway. The server process waits for a user process to initiate a session, and responds to commands from the user process. In response to user commands, the server may perform services on the target like reading and writing memory locations or setting breakpoints. 'Server' is sometimes employed as a shorthand for 'server process'.

target

The 'target' in an LDP session is the PSN, PAD or gateway that is being loaded, dumped or debugged by the host. Normally, LDP will be implemented in the target as a server process. However, in some targets with strange requirements, notably the Butterfly, the target LDP may be a user process.

user process

The LDP user process is the active participant in an LDP session. The user process initiates and terminates the session and sends commands to the server process which control the session. The user process usually resides on a timesharing host and is driven by a higher-level entity (e.g., an application program like an interactive debugger). 'User' is sometimes employed as a shorthand for 'user process'.

word

A word is a sixteen-bit quantity.

INDEX

The CSNET Name Server

Marvin Solomon, Lawrence H. Landweber and Donald Neuhengen
University of Wisconsin-Madison, Computer Sciences Department, 1210 W. Dayton St., Madison, WI 53706, USA

CSNET is a project designed to facilitate electronic data communication among academic computer science departments and other groups doing computer-science research in the United States. CSNET will provide communications facilities for electronic mail and file transfer between users of computers connected to a variety of networks. For the system to be simple and easy to use, users must be able to identify each other to the system in a way that is natural to them and which does not require them to understand the details of network organization or to memorize cryptic names. To this end CSNET is implementing a *name server* sevice, composed of programs and data residing on a central Service Host computer and on individual member hosts of CSNET. This paper describes the architecture of the name server and discusses the considerations that lead to its design.

Keywords: Name server, directory assistance, computer networks, database, CSNET

Marvin H. Solomon received a B.S. Degree in Mathematics from The University of Chicago in 1970 and the M.S. and Ph.D. degrees in Computer Science from Cornell University in 1974 and 1977, respectively. He was visiting lecturer at Aarhus University (Denmark) during the 1975-76 academic year. Since 1976 he has been on the faculty of the Computer Sciences Department of the University of Wisconsin-Madison, where he is currently Associate Professor. His interests include design and implementation of programming languages, interconnection structures and operating systems for multicomputers, and computer networks. He is currently a co-principal investigator in the DARPA-funded "Charlotte" distributed operating system project and is in charge of the CSNET name server implementation.

Lawrence H. Landweber received a B.S. in Mathematics from Brooklyn College in 1963 and a Ph.D. in Computer Science from Purdue University in 1967. Since 1967 he has been on the faculty of the Computer Sciences Department of the University of Wisconsin-Madison. His research interests include computer networks, electronic mail, and theory of computing. For the past three years he has been involved in the CSNET project. He is currently chairman of the CSNET Management Committee and is a participant in the name server component of the project.

North-Holland Publishing Company
Computer Networks 6 (1982) 161-172

1. Introduction

CSNET is a project, funded by the National Science Foundation, to develop common protocols, software packages, and an administrative organization to facilitate communication among academic computer science researchers in the United States. An important component of CSNET will be a directory service called the *CSNET Name Server*, which is implemented by a central database at the University of Wisconsin and by software running at Wisconsin and on the computers of CSNET member institutions. This paper describes the architecture of the name server facility.

In early stages of CSNET, the principal use of the name server will be to facilitate sending of electronic mail by providing directory assistance in locating addresses of mail recipients and aiding in forwarding mail and establishment of nicknames and aliases. It is on this aspect of the name server that this paper focuses. In later stages of CSNET, the name server will also help support other facilities such as file transfer and remote access to computing resources.

In the next section, we briefly describe CSNET and explain how its characteristics have influenced design of the name server. The structure of CSNET is described in more detail in [1,2].

We have designed the name server to be implemented in a series of phases, progressing from facilities that already exist, through more and more sophisticated structures, to a system that will eventually provide all of the desired features. In doing so, we have attempted to be conservative in early phases, using the simplest structure that will fulfill the immediate needs of CSNET users, while leaving the door open for more ambitious enhance-

Donald Neuhengen received B.S. and M.S. degrees in Computer Science from the University of Wisconsin-Madison in 1979 and 1981, respectively. He is currently a member of the academic staff at the University of Wisconsin-Madison where he is the chief programmar on the name server implementation project. His current interests include computer-aided design tools and computer networks.

ments in the future. While the services described here will be implemented with available CSNET staff and resources, we expect the project to identify several challenging additional research areas.

Section 2 describes CSNET and discusses project characteristics that have influenced the name server design. Section 3 describes name server design requirements and implementation performance goals. Section 4 includes definitions of terms and an outline of the various phases. The four phases of the name server implementation are described in Sections 5-8. Section 9 briefly addresses the issue of mailing lists, and Section 10 provides a summary and comparison to related work.

2. Overview of CSNET

CSNET is a logical network that uses communications services provided by ARPANET [3], the commercial value-added network Telenet, and a telephone-based mail relay service called PhoneNet, based on the MMDF mail transport system [4]. Member institutions access the services of CSNET by connecting a computer ("host") to ARPANET or Telenet, or if their budget is limited and they are willing to accept reduced service, by arranging for their host to exchange mail periodically with a PhoneNet *relay machine* that is directly connected to ARPANET and Telenet. CSNET will provide electronic mail, file transfer, and remote login (virtual terminal) services to directly-connected hosts. PhoneNet hosts will only have access to electronic mail sevices. In addition, CSNET maintains a *Public Host*, which is a VAX computer connected to ARPANET and Telenet, running the UNIX operating system, and providing mail-only accounts to individuals who do not have access to any other CSNET member host. Users access the Public Host using terminal-to-host services of Telenet. Although CSNET is being subsidized in its initial stages by the National Science Foundation, it is expected to become self-supporting in a few years, with all members paying a fair share of the costs.

One of the challenges of CSNET is to reconcile differences between characteristics of these communications media and provide users with as uniform an interface as is possible. ARPANET provides a high-bandwidth, low-delay communications path between computers connected to it. Telenet provides similar (but lower bandwidth) service. Whereas the cost of an ARPANET connection is fixed, Telenet charges are highly dependent on the amount of traffic. PhoneNet charges are even more dependent on traffic, since the only fixed charges are the cost of a modem and a telephone line. The remaining charges depend on the number of calls placed and on their duration. However, a much more important difference between PhoneNet and direct connection is delay. CSNET clients not directly connected to ARPANET or Telenet must rely on periodic exchanges of mail with a PhoneNet relay machine for their connection to the network. The frequency of such exchanges may be as low as once daily. We shall see that these wide variations in delay (from minutes or seconds to days) is an important consideration in the design of the name server.

3. Goals

The name server facility is designed to satisfy the following service requirements:

1. The system must be simple to use. While most CSNET users will be computer science researchers, many will have little experience with computer-based mail systems. Users should be free to concentrate on their research without worrying about details of addressing or mail-transport systems.
2. A sender of mail must be able to identify a recipient in a variety of convenient ways. A user may refer to frequent correspondents by nicknames of his own choosing. In addition, a host may make available to its users aliases for other hosts and users.
3. A receiver of mail must have control over the information provided to others for use in identifying him. For example, he can supply his full name, organization, location, title, nicknames, common misspellings of his name, etc.
4. Correspondents must be able to send mail to any user of any host in CSNET, without prior explicit effort on the part of the receiver, although reduced services will be available for communication with "unregistered" users. Similarly, CSNET users must be able to communicate with others "outside" the network, in particular users on hosts that are in the DARPA

Internet address space but are not running CSNET-specific software.

5. The mail system must never force a user to use more than one "mailbox" to receive mail, although a user may choose to establish more than one mailbox to reflect differing roles. In the latter case, each mailbox may be thought of as representing a different "virual user".
6. A user must be able to move his mailbox to a different host computer with a minimum amount of difficulty. Senders need not be explicitly notified; mail will be automatically forwarded.
7. Support must be provided for mailing lists.

In addition to these service requirements, the implementation is designed to satisfy the following additional performance and utility goals:

1. The system should expand gracefully to include more member sites, additonal users, and even additional networks. In particular, anyone able to send electronic mail to the University of Wisconsin should be able to gain access to at least some of the name server services.
2. The system design should provide for phased implementation so that basic services can be put into place immediately, while more sophisticated facilities may be added incrementally until all desired features are available.
3. Network traffic should be minimized. Control messages should be infrequent and user text should be sent over the most efficient route. Parts of the name server rely on existing networks and mail transport systems for communication. While the name server has no control over routing algorithms used by these facilities, the cost of communications must be taken into account in the design. In particular, situations in which user text is sent over multiple hops through the mail system should be avoided.
4. The system should continue to function, perhaps in a degraded mode, if components fail. This consideration precludes a design which makes mail transfer impossible if the central database is temporarily unavailable.
5. Delay between the submission of a message by a sender and its delivery to a recipient should be minimized. In particular, if the sender is on a machine that is only periodically connected to the rest of CSNET (a PhoneNet host), the number of interactions between that host and the rest of CSNET required to dispatch the message should be minimized.
6. The system should work with a minimum of human intervention, either on the part of users or of administrative staff.

4. Definitions

Throughout this paper, we will be talking about *users* and *hosts*. For our purposes the term "user" always refers to a human being (and will not, for example, be used to mean a "user program"). A host is a computer connected to a communications network. Users gain access to network facilities through accounts on hosts. Hosts can be classified as *CSNET member hosts*, which subscribe to CSNET-defined conventions and run CSNET-provided software packages, and *other hosts*, which are capable of exchanging mail with CSNET member hosts but do not necessarily run CSNET software. There are also several CSNET-run hosts. The *Service Host*, is a computer at the University of Wisconsin that maintains a central database and programs for accessing it. *PhoneNet relays* are computers (initially at the University of Delaware and the Rand Corporation) that periodically place telephone calls to other hosts to pick up and deliver mail. *The Public Host* is a computer at the University of Wisconsin that is run by CSNET but otherwise is treated exactly like any other CSNET member host. Hosts may also be classified as *ARPANET*, *Telenet*, or *PhoneNet* hosts depending on the principal method used to exchange information with the rest of CSNET. The name server relies for many of its functions on a *mail transport system* which is a collection of protocols and programs that run on hosts and provide the mechanism for transferring messages from sources to destinations. Users normally interact with the mail transport system through a *user-interface program (uip)*, which is a program that interacts with users for composing, sending, receiving, reading, and filing messages.

The various services and mechanisms described in this paper comprise the *name server* facility. It is provided by a combination of files and programs residing on the Service Host and on other CSNET member hosts. The *name server database* is a database that includes directory information for registered CSNET users and hosts and is distributed among a *central directory database* that resides on the Service Host, *per-host tables* that reside on hosts that originate mail, and *per-user tables* maintained by local mail systems on behalf of individual users. The *registrar* is a collection of software that runs on the Service Host and mediates access to and modification of the central directory database.

Users may access the name server database by sending messages directly to the registrar. However, users will normally compose their queries by interacting with a *name server agent program*, copies of which reside on CSNET member hosts. A copy of the agent program will also reside on the Service Host for the convenience of users on non-CSNET hosts who have virtual-terminal access to the Service Host. The agent programs communicate with the registrar using the best means available, either by direct connection or by exchange of messages through the mail transport system. In the latter case, there is necessarily a large delay, so users will receive a limited level of service.

The name server facility is specified and implemented in four phases. As new phases are implemented, all features provided by earlier phases will remain available to users. Phase 0 provides basic services and is compatible with current addressing and naming schemes employed in the DARPA Internet. Phase 1 introduces a centralized directory database at the Service Host and a directory assistance service that users may access by exchanging mail with the registrar or by interacting with an agent program. In Phase 2, user interaction with the directory assistance service will be further automated. Phase 3 adds support for automatic forwarding of mail and for mailing lists.

5. Phase 0: Basic Services

Phase 0 provides services that are very close to those currently available in the DARPA Internet environment. Each host in CSNET has an unambiguous name, such as "UWISC", "UDEL", or "WASHINGTON". A site with a local network may choose to designate a particular computer to serve a as a mail forwarder and assign it a name that designates the site. Arpanet hosts already have unambiguous names. Other mail transport systems also rely on unambiguous names for hosts. As sites join CSNET, they will register their hosts with the CSNET administration, which will certify that names are not duplicated. (By "unambiguous" we mean that no two hosts will have the same name; there is no reason to prevent a host from having more than one name.)

Users interact with the mail system through accounts on hosts; each account is assigned a *user name*. Each host will guarantee that a user name unambiguously identifies one user of that host. In other words, "user name" represents some name for a mailbox that is printable, is assigned by a host administration, and identifies a unique user of that host. Hence the pair "user-name@host", which we call a *mailbox address*, can be used to uniquely identify any mailbox in CSNET.

The details of the structure of mailbox addresses are not important to the design of the name server. The essential features of a mailbox address are that it be acceptable to the mail transport systems as an unambiguous designation of the final destination of a message and that it be sufficiently readable and mnemonic that a human user can supply it manually if necessary. The later property alllows a sending user who does not have access to CSNET software to bypass the name server entirely and specify the mailbox address directly. As we shall see, however, more convenient methods will be available to users who use the name server.

6. Phase 1: Directory Assistance

Phase 1 augments the basic facilities described in the previous section with a "directory assistance" service. A central directory database on the Service Host will contain information about users. Each entry in this database contains the address of one mailbox. This information is supplied by the owner and includes, at a minimum his full name and the name of his sponsoring organization (e.g., university or research lab). In addition, it may include other keywords such as titles, aliases, and common misspellings of the user's name, postal address, phone number, and any other information the user wishes to provide. Registration in this database is entirely voluntary and it will be possible to communicate with non-registered users even if their local site has not installed the CSNET name-server software.

The database is accessed by transmitting proping his mailbox but including keywords that match some other user, with the intent of fooling users into sending mail for the other user to the perpetrator's mailbox. This ruse would be particularly pernicious when lookup is automated so that human users don't normally even look at the mailbox address returned (see Phase 2). The situation is comparable (in the non-electronic world) to Marvin Solomon putting an advertisement in the newspaper saying that the address of the First National Bank of Madison is 850 Terry Place (Solomon's home address). The name server mechanism cannot prevent such a fraud without understanding the semantics of all the keywords in an entry. But the injured party, if he discovers that mail is being misdirected, can query the central database to find the bogus entry. Similarly, a sender might notice that certain queries identify two entries, one of which looks suspicious, and report the fraud. Once the fraudulent entry is found, the culprit can be traced as far as his host.

Authentication is a difficult but important area. Further study will be required if a more elaborate scheme than that described above is found to be necessary.

6.3. Using the Database

To access the central directory, a query is delivered to the registrar or mailed to it at the address REGISTRAR@CSNET-SH. The mail format is designed to allow human users who do not have access to CSNET software but who can send mail to CSNET-SH to compose the query manually. Normally, however, users will use a "whois" command of the agent to compose such a request. The query will include lists of *mandatory* and *optional* keywords. Only entries that contain all mandatory keywords will be selected. If more than one entry matches, the optional keywords can be used to select the entry with the most matches, or the registrar may be instructed to return only entries containing at least *k* of the optional keywords.

We are assuming that there is no access control on read-only access to the database. Registration is entirely voluntary, so users who wish to remain anonymous need only avoid registering. We could add a facility for restricted entries or fields in entries (visible only to selected users) using the authentication mechanism described above. We currently have no plans to implement such a facility, although there are some fields in entries (for example the password field) which are restricted to use by the registrar itself and are never shown to users.

Keywords can be parameterized so as to allow specification of pattern matching. Keywords may also contain "wild cards" to allow inexact matches. For example, the keyword "landwe*er" can be used by those not knowing whether his name is spelled as "landweber", "landwever", or "landwebber". Upper and lower case are considered equivalent for matching purposes, but the entries will be displayed to the requester in the same case as they were originally specified at registration. The requester can then select the appropriate entry (if there is more than one match) based on other information in the entries, and use the mailbox address included in the entry to send mail.

The provision of mandatory and optional keywords is primarily for the benefit of the user of a PhoneNet host, to maximize the chances of him getting the right answer on the first try. Too few keywords will flood him with bogus matches, but too many mandatory keywords may exclude valid matches. The ability to get a unique match on the first try will become particularly beneficial in phase 2 (as we shall see).

Incidentally, directory assistance could be useful for services outside CSNET proper, such as looking up a user's phone number or (U.S. Post Office) mailing address.

6.4. Example

Here is a sample use of the name server in phase 1: To make it easier for others to find Marvin Solomon, he issues the "register" command to the agent, which engages in a dialogue with him to authenticate his identity and gather information about him. It then composes and encrypts a message to REGISTRAR@CSNET-SH containing text something like this:

```
register
home
    solomon@uwisc
mailbox
    solomon@uwisc
name
    Marvin Solomon
address
    University of Wisconsin Madison
    Computer Sciences Department
```

erly formatted queries to the registrar at the Service Host. Users will not normally communicate directly with the registrar but rather with an agent program that formats the request and forwards it to the Service Host by a direct connection or by the mail transport system. However, users at non-CSNET computers may also query the database by mailing requests directly to the registrar.

Since we intend that each user have the ability (and responsibility) to maintain the database entry describing him, certain access controls must be in place from the very beginning to maintain the integrity of the database.

6.1. Registering

The user adds or modifies entries in the database by interacting with his local agent using the commands "register", "update", and "unregister". The local agent creates a message containing the user request to insert, modify or delete a central database entry and sends it to the registrar either directly or by mailing it to the address "REGISTRAR@CSNET-SH". The registrar will parse the message and perform the requested operation.

6.2. Authentication

An important issue is authentication of a user requesting insertion or modification of an entry. A user may only register himself with the cooperation of a CSNET member site, which we may call his *home* or *sponsoring* host. When an organization joins CSNET, it is assigned a key (password). The administration at the site will be responsible for controlling its distribution and for changing it when appropriate.

A user at a member host who wishes to register himself in the database interacts with his local agent. This agent runs as a privileged program that has access to the site password. The agent engages in a dialogue with the user to authenticate his identity (for example, by asking for a password) and verifies that the proposed database entry is appropriate to the user (in particular, that the "mailbox address" field properly identifies the user). Having satisfied itself of the validity of the request, the agent formats it, encrypts it using the site password as an encryption key, adds an unencrypted header identifying the local site, and forwards it to the registrar. The registrar decrypts the request and installs the information in the database. Even though the request passes through unsecure channels, the potential for subversion is limited, since the message contains neither the request nor the password in plain text. Thus an adversary cannot modify the request or use it to construct a bogus message. The encrypted portion of the message also contains a timestamp, so that an interloper cannot confuse the name server by holding the request and retransmitting it later.

This scheme delegates authority for authenticating users to the member sites. Each site is held responsible for all database entries that designate that site as sponsoring organization. The agent program (which is provided by CSNET) gives a mechanism for controlling these entries. A user cannot bypass this mechanism and send a registration request directly to the registrar because he does not know the site password. The registrar will not accept requests for new entries mailed directly from a non-member host, nor will the the copy of the agent program resident on the Service Host accept a registration request, since the user cannot be authenticated in either case. In other words, only a user of a CSNET member host may add entries to the database, and he may only do it using a local copy of the CSNET agent program.

Upon registration, a user may provide a password to be used by him when modifying or deleting his directory database entry. This password can be used to initiate a change request from a host other than his sponsoring machine. This feature is particularly useful when an individual moves to a new site and changes his mailbox address. The registrar will inform the host specified in the original database entry that the entry has been changed. This notification will provide an additional check to ensure that the change is authorized.

To perform housekeeping functions, particularly deletion of defunct entries, a site may authorize a special trusted user to perform commands on behalf of other users at that site.

This authentication mechanism is not "airtight", but should provide an adquate level of protection at modest cost. More importantly, it delegates authority for security, so that if breaches are detected, the responsible party can be identified.

An interesting example of a fraud that is *not* prevented by this scheme might be called "false advertising". The owner of mailbox VESCO@COSTA-RICA inserts an entry address-

```
    1210 W. Dayton St. Madison WI 53706
phone
    608 262-1204
keys
    soloman csnet contractor service host public host computer
    science
end
```

A user who wishes to send mail to Solomon might issue the command

whois solomon [csnet implementor]

where the keyword "solomon" is mandatory, but the keywords "csnet" and "implementor" are optional. There may be several entries containing the keyword "solomon", but the one shown above is the only one containing either of the words "csnet" or "implementor". He would receive the response:

```
In response to (whois solomon [csnet implementor]):

solomon@uwisc
Marvin Solomon
University of Wisconsin Madison
Computer Sciences Department
1210 W. Dayton St. Madison WI 53706
608-262-1204
soloman csnet contractor service host public host
computer science.
```

(The response would be abbreviated in an interactive setting.) He could then send mail to Solomon by addressing it to "solomon@uwisc". Existing mail user interface programs generally have a nickname (also called "alias") facility that allows the user to say something like:

alias marv = solomon@uwisc

to avoid having to remember the address.

Solomon included "soloman" as a keyword, since he knew that people frequently misspell his name that way. The user querying the database can also protect himself from misspelling by using a combination of wildcards and optional keywords. For example, he could say

whois s* wisconsin [soloman solomon salemon]

7. Phase 2: Automated Lookup

Phase 2 adds services to decrease the amount of interaction required between a human user and the central database. In particular, the mail uip and the local agent are integrated.

7.1. Automatic Nickname Establishment

In Phase 1, responses resulting from central database lookup queries are always returned directly to the user. In Phase 2, facilities will be added to automate establishment of local nicknames.

Continuing the previous example, the interaction with the name server and the establishment of a local nickname could be combined by issuing the command

alias marv = whois(solomon
[csnet implementor])

to the mail uip, which would format a request and send it to the registrar. (No authentication is required since no change to the nameserver database is being requested.) If the query matches exactly one entry, the nickname is installed in the user's private nickname table. If no entries or more than one entry are returned, the response is returned to the user requesting more information. In the PhoneNet environment, the user receives notification of success or failure of nickname establishment in the form of a message mailed to him. A facility will also be provided by which a local administrator can install commonly used aliases in a table accessible to all users at a site.

Finally, the user will be able to combine query of the database, establishment of a nickname, and sending of the first message with a command such as

sendto marv = whois(solomon
[csnet implementor])

The ability to combine these operations will be particularly advantageous to PhoneNet users. The initial message will be delivered to the PhoneNet relay with the keyword information and a placeholder instead of a mailbox address in the "To" field. The relay composes a query to the registrar and intercepts the reply. Assuming that a unique match is found, the relay updates the message header to include the mailbox address (leaving the keyword information in as a comment) and delivers the message as usual. It also forwards the reply from the registrar back to the originating host so that the private nickname table of the sender can be updated. The advantage of this scheme is that the message can be delivered after only one interaction between the sending host and the relay. For example, if the sending host is only polled once each day but the destination host is directly connected, this scheme provides next-day delivery, assuming that the list of keywords uniquely identifies one mailbox. The reason for requiring each database entry to include the user's full name and institution (CSNET member organization), is to give the careful sender a reasonable chance of constructing a query that will match uniquely on the first try.

8. Phase 3: Forwarding

Suppose an existing user is assigned a mailbox on a new host. Under some circumstances, he may want that mailbox to be considered different from his previous mailbox. For example, he may have changed jobs. Under existing mail transport systems, a message sent to the old mailbox (assuming it was deleted) would be returned to the sender with an indication that the mailbox no longer exists. A user who is really interested in sending to him as a person, rather than in his official capacity at his old job, could query the database to determine his new address and resend the mail. This situation corresponds closely to the telephone system (where "address" corresponds to phone number). However, under other circumstances, the user would like the change of address to be invisible to his correspondents. For example, suppose he is moved to a different host on the basis of an administrative decision such as load-leveling, or he is temporarily visiting another site and finds it more convenient to have mail forwarded there (compare with the phone company's "call forwarding" service). In the latter case, we feel that the mailbox is the same; only its *address* has changed.

We can identify a "logical" mailbox with an entry in the nameserver database. The entry contains the address of a "physical" mailbox to which it is currently bound. Each time a message is sent using keywords or a nickname to specify the destination, the name server database could be searched to find the correct current address. This solution has several problems. First, if the Service Host were queried each time a message was sent using keywords, the load would be severe. Second, a failure of the Service Host or the communications path to it would delay sending of all such messages. A third problem is the increased communications cost. Finally, messages originating at a PhoneNet site could be delayed an extra day or more waiting for a response from the Service Host.

Some of these problems can be mitigated by distributing he database. This solution was chosen for both the Grapevine Registration Server [5] and the Clearinghouse [11]. However, a distributed database poses certain difficulties. If not all servers contain the entire database, there is the additional complexity of finding an appropriate server. On the other hand, if any of the information is replicated, there is the problem of maintaining consistency over updates. Grapevine and Clearinghouse solve the latter problem by allowing a limited amount of temporary inconsistency, assuring only that multiple copies are eventually consistent. We have adopted a different approach. We use a single centralized database, but cache some of the information at multiple sites. The cached information is treated as a "hint"; if a part of the system discovers or suspects it is out of date, it consults the Service Host to obtain the correct information.

8.1. The Forwarding Mechanism

To simplify various aspects of forwarding, each nameserver database entry will contain a *registration ID* that uniquely and unambiguously identifies the database entry. This ID is included in database entries from the start, but only comes into play in Phase 3. (This idea was inspired by a suggestion of Denning and Comer [6]). Note that the registration ID identifies the *database entry* and therefore the logical mailbox. The "mailbox address" field of the entry is the address of a physical mailbox to which the logical mailbox is currently bound. The mailer will be modified to include the registration ID in per-user nickname tables. For example, if a user types

```
alias mary = whois(solomon
                  [csnet implementor])
```

the nickname table will store with the nickname "mary" not only the mailbox address (SOLOMON@UWISC), but also the registration ID for the associated database entry.

The forwarding mechanism is best described by an example. Suppose Pat Kane is at site A and has a mailbox with address "PAT@SITEA". He moves to site B and is refused the name "PAT" as his mailbox name since there is already a PAT there, so he chooses the name "PKANE". The mailbox PAT@SITEA is deleted from site A. Users who bypass the CSNET name service entirely and send to "PAT@SITEA" will have their messages returned as undeliverable. They must learn from channels outside of CSNET (such as word-of-mouth) that mail to Pat Kane must now be addressed to "PKANE@SITEB". However, Pat Kane may inform the registrar that he has moved. (Authentication of the information uses a similar mechanism to that described above for registering users.) His entry in the central database is updated to indicate his new mailbox address, so that new correspondents looking for him by keyword search will find his new address. Old correspondents will still have mail returned, but now senders who use the name server can have their local mail systems recover without manual intervention.

Suppose sender "ABE@SITEC" has established an alias for Pat Kane by the command

alias pk = whois(pat kane)

When the nickname was established, the mailer on SITEC estabished the mappings:

(pk, ABE) → (PAT@SITEA, 0012345)

where 0012345 is the registration ID of the entry describing Pat Kane. When ABE tries sending to "pk" after Pat has moved, as message addressed to "PAT@SITEA (CSNET-ID: 0012345)" is sent to SITEA, refused, and returned to sender. (Current mail systems treat the material in parentheses as a comment.) The sender's mailer can query the registrar to find out if there have been any changes in entry 0012345. In this case, the registrar sends the new address "PKANE@SITEB", and SITEC's mailer updates its tables to contain the mapping

(pk, ABE) → (PKANE@SITEB, 0012345)

and re-sends the letter to "PKANE@SITEB (CSNET-ID: 0012345)". The sending user is never bothered, and all his future mail to "pk" will be sent directly to the correct address.

One additional complication arises. Continuing the previous example, suppose after Pat Kane leaves site A, Pat Able appears and wants to be known locally as "PAT". She might well be unhappy about being told that she couldn't use the name "PAT" because there once was man named Pat Kane who already reserved that same name. On the other hand, SITEA will have no way of knowing whether mail addressed to "PAT@SITEA" was intended for Pat Kane or Pat Able. Once again, senders who bypass CSNET software can still send mail, but they receive reduced services. In this case, they run the risk of a message going to the wrong person.

If SITEA is a CSNET member site, however, it will store the registration ID of all its local users who are registered. If an incoming message contains a registration ID that does not match the registration ID of the addressee, the message will be rejected. When Pat Kane changes his address to "PKANE@SITEB", the registrar informs SITEA the the user id "PAT:0012345" is no longer valid.

Phase 3 requires modifications both to the mail user-interface program and to the programs that send and receive mail from outside the host. The uip must store the registration ID with cached addresses from the name server and include it in messages. The program that receives mail from the network will be modified to do additional checking on the validity of messages that contain a registration ID, rejecting them if the address and registration ID do not match according to tables at the receiving host. The program that delivers mail to the network will be modified to do extra checking for messages rejected by the destination, going to the central database to verify addresses of rejected messages.

These changes might be viewed as delaying the binding of name to address [7]. In the existing system, the "user@host" string is an address supplied by the mail-preparation software when the message is created. With the phase-3 modifications, the uip denotes the destination of the message by its registration ID (the name of the logical mailbox) and includes a suggested address where it might be found. The binding of name to address occurs in two stages, first trying the suggested address and then accessing the central database if the suggestion proves incorrect.

8.2. Optimizations

The update message from the registrar to SITEA could include the forwarding address, and SITEA could cache forwarding addresses for recently moved mailboxes. When the letter addressed to "PAT@SITEA (CSNET-ID 0012345)" arrives at SITEA, SITEA could then send it directly to PKANE@SITEB rather than returning it to the sender. It should still inform the sender of the change, and the sender may well wish to check the new address with the registrar rather than trusting SITEA, but the delay in delivering the letter would still be reduced from five message-transition times (sender to SITEA; SITEA to sender; sender to Service Host; Service Host to sender; sender to SITEB), to two (sender to SITEA; SITEA to SITEB). The possibility of this sort of forwarding raises difficult problems in billing, however (e.g., who pays for the forwarding hop and how is he billed), which are beyond the scope of this paper.

Another optimization is based on the observation that it is common for several users at one site to correspond with the same person. If they all have obsolete nicknames for him, the overhead of a misdirected message can be moved from the first time each of them sends to him to the first time *any* of them sends to him. Instead of storing the entry "pk": PAT@SITEA (CSNET-ID 0012345)" in the nickname table for a user, we will store an entry like "pk: 0012345" in the per-user table and maintain a per-host table with the entry "0012345:Pat@SITEA".

Finally, more of the information may be cached in various hosts. In particular, each PhoneNet relay will keep a copy of the complete mapping from registration ID's to mailbox addresses. When a message is sent from a PhoneNet site to an obsolete address, the relay can query the central database and retransmit the message to the proper destination without the cost and delay of communication with the sending site. From the point of view of a PhoneNet site, the registration ID acts as an address, and the translation to the correct USER@HOST mailbox address may be considered part of the translation from address to route.

9. Mailing Lists

All the mechanisms described thus far are techniques for discovering the address of one mailbox. There is nothing to prevent them from being used repeatedly and in combination to develop multiple addresses on a single message, such as

```
sendto
    marv,
    larry = whois(lawrence landweber
    wisconsin),
    donn@uwisc
```

which names the three authors of this paper by a nickname, a keyword search, and a mailbox address, respectively.

A related facility is the *mailing list* which is a name for a list of mailboxes that are often used together. Existing user interface programs often provide a mailing list function using the nickname facility to do a macro expansion of a mailing list name. In Phase 3, the CSNET name server will allow users to register mailing lists in the central directory database. A mailing list entry is similar to other entries in that it contains a list of keywords and identification of a user responsible for the entry. But it also contains a list of items, which can be mailbox addresses, CSNET ID's, and names of other mailing lists. A request to add a mailing list to the directory contains the keywords describing the list, as well as descriptions of the members, specified by any convenient means (i.e., keywords, nickname, or mailbox address). On receiving such a request (which must pass the usual access checks), the registrar stores the list, after replacing those members specified by keywords or nicknames by the corresponding CSNET ID's. (The list is allowed to contain mailbox addresses to accommodate unregistered members.) If any member specification fails to resolve to a unique address, the request will be returned to the sender for correction.

When a mailing list is installed, the registrar will send a message containing the names of all the members to each member. (This notification may be suppressed.) Similarly, a change in the mailing list can generate a notification to all parties involved.

10. Comparison to Other Work

Several reports have been published on "name servers" for computer networks [8-10]. In the ARPA internet community, the term "name server" is often used to mean a service for translating host names (such as "CSNET-SH") to 32-bit internet addresses (such as 1200200136) used by the transport level to address a host. The Service Host will also provide a name service in this sense, but discussion of that service is outside the scope of this paper.

Two particularly interesting related name server designs are the Grapevine registration service [5] and the Clearinghouse [11] both at Xerox PARC. Both services provide for naming, authenticating, and locating people, machines, and services in a multinetwork environment. The Grapevine registration service provides for the translation of two-part names to other names, lists of names, or internet addresses. The set of names with the same first component is called a *registry*. The registration service is provided by several dedicated *registration servers*, each of which holds all information about one or more registries, and each registry may be stored at one or more servers.

The Clearinghouse extends these ideas by providing a more complex organization of registries and servers and by extending the class of values to which names are bound. Names have the format "Individual@Domain@Organization", where the intention seems to be that "Organization" is a corporate entity (such as "Xerox"), "Domain" is an administrative division of the company (such as "SDD"), and "Individual" may be a person or a component of the computer system, such as a server. Names are guaranteed to be globally unambiguous (that is no two objects have the same name) by requiring the domain name to be unique in a domain. In the case of human individuals, the name is the person's full name, possibly with a "birthmark" affixed to assure uniqueness. Names are proposed by a human administrator and verified for uniqueness by the system itself. Each name is bound to a property-list, with property names chosen from a fixed set of possibilities and values which may be individuals or lists. The database contains information on its own structure, for example, there is a registry "SDD@Xerox@Clearinghouse" that lists servers for names in domain "SDD" of organization "Xerox". An interesting algorithm uses this information to find a server with the binding of a given name.

Both of these systems are designed to operate in an internetwork environment with associated databases and services distributed among different machines on different networks. Therefore, many of the complications that concern the authors of these papers, such as how to find a name server and how to maintain consistency between replicated copies of a registry do not arise in our context, in wich there is a unique name server at a well-known address. On the other hand, these systems are designed for environments in which message-passing is cheap and quick, and in which broadcast is a viable means for locating services. Some of their techniques do not apply in an environment in which a single message "hop" can take more than a day.

Another difference is in how the systems are to be used. The designers of Grapevine and the Clearinghouse have chosen to limit their functionality to a simple name lookup, removing more complex database query functions to client software that uses the services of the registration server. A lookup request to Clearinghouse, for example, must fully specify a three-part name, possibly with "wildcard" characters, for example "*@SDD@Xerox". There is no support for content-addressed queries. In our system, the primary goal is to facilitate lookup of mailbox addresses based on incomplete information. Hence, it is not necessary to know any particular piece of information such as the user's complete na [illegible] to locate his entry.

11. Summary

We have presented a detailed specification of a name server facility for CSNET and have sketched out the algorithms for implementing it. The facility is implemented by a postmaster general program running on the Service Host and local agent programs running on local hosts. The facility will be implemented as a series of enhancements to existing services, each adding more convenience for users. It assumes a mail transport system that can deliver a message when presented with a list of destination addresses. It also allows for interactive database access in cases in which the user or the user's host is capable of direct connection to the Service Host.

We have deliberately avoided discussing issues involving the mail transport system, such as routing, mail relays, multicast delivery, or reply-to-sender, except as they are directly related to the name server, but we do not believe that the name server facility creates any new problems in these areas since address specifications ultimately resolve to addresses in the style already in use. We have also not tied the name server specification to a particular mail interface program.

The central database is structured as a sequence of fixed length records, one per entry, with a separate overflow area for long entries. An inverted index also consists of a sequence of records, one for each word appearing in the database. Each record contains a list of pointers to entries containing the word. A hash table speeds access to the index. A database access program does all lookups and modifications of the database. It can be invoked by an agent program on the Service Host, an agent connected to the access program by a virtual-terminal protocol, or by a mail opener program that accepts and validates mail requests. A monitor process controls concurrent access to the database by multiple copies of the access program.

The techniques for implementing the algorithms described here are well understood, and tools (such as a flexible filesystem and encryption programs) are already in place in the operating system for the Service Host. Therefore, we feel that the name server can be implemented quickly and begin to provide services to users soon. At the time of this writing (February, 1982) Phase 1 is implemented and undergoing testing. The remaining phases are expected to be completed in the next year.

Acknowledgments

We gratefully acknowledge helpful comments and suggestions from numerous colleagues, including Vint Cerf, Doug Comer, Yogen Dalal, Peter Denning, Stuart Feldman, Keith Lantz, Mike Litzkow, Jon Postel, and Fred Schneider. The design was strongly influenced by a memorandum of Comer and Denning [6], and the presentation in section 8 was helped by discussions with Fred Schneider.

References

[1] L.H. Landweber, "CSNET-A computer research network," *Proposal to the National Science Foundation*, (October, 1980).

[2] L. Landweber and M. Solomon, "The use of multiple networks in CSNET," *Proceedings of Compcon Spring 1982*, (February, 1982).

[3] L.G. Roberts and B.D. Wessler, "Computer network development to achieve resource sharing," *Proceedings of SJCC*, pp. 543-549 (1970).

[4] D. Crocker, E. Szurkowski, and D. Farber, "An Internetwork memo distribution capability-mmdf," *Proceedings of the 6th Data Communications Symposium*, (November, 1979).

[5] A. Birrell, R. Levin, R. Needham, and G. Schroeder., "Grapevine," *Proceedings of the [illegible] ACM Symposium on Operating Systems Principles*, (December, 1981).

[6] P. Denning and D. Comer, *The CSNET user environment*, Computer Science Department, Purdue University (July, 1981) unpublished note.

[7] J. Shoch, "Inter-network naming, addressing, and routing," *Proceedings of the 17th IEEE Computer Society International Conference*, (September, 1978).

[8] J.R. Pickens, E.J. Feinler, and J.E. Mathis, "An experimental network information center name server (NICNAME)," IEN 103, SRI International, Menlo Park, California (May 1979).

[9] J.R. Pickens, E.J. Feinler, and J.E. Mathis, "The NIC Name Server–A datagram based information utility," *Proceedings 4th Berkeley Workshop on Distributed Data Management and Computer Networks*, (August 1979).

[10] J. Postel, *Internet Name Server*, Information Sciences Institute, University of Southern California (May, 1979)

[11] D.C. Oppen and Y.K. Dalal, "The Clearinghouse: A decentralized agent for locating named objects in a distributed environment," Technical Report OPD-T8103, Xerox Office Products Division (October 1981).

IEN 116

J. Postel
ISI

August 1979

Obsoletes: 89, 61

INTERNET NAME SERVER

INTRODUCTION

This memo defines the procedure to access an Internet Name Server. Such a server provides the actual addresses of hosts in the internet when supplied with a host name. An Internet Name Server is a dynamic name-to-number translation service.

This server utilizes the User Datagram Protocol (UDP) [2], which in turn calls on the Internet Protocol (IP) [3].

NAME SYNTAX

It is strongly reccommended that the use of host names in programs be consistent for both input and output across all hosts. To promote such consistency of the internet level, the following syntax is specified:

The SYNTAX of names as presented to the user and as entered by the user is:

! NET ! REST

where:

NET is a network name or number as defined in "Assigned Numbers" [1]

and

REST is a host name within that network expressed as a character string or as a number. When a number is used, it is expressed in decimal and is prefixed with a sharp sign (e.g., #1234).

Note that this syntax has minimal impact on the allowable character strings for host names within a network. The only restriction is that a REST string cannot begin with an exclamation point (!).

The !NET! may be omitted when specifying a host in the local network. That is "!" indicates the network portion of a name string.

BASIC NAME SERVER

To aid in the translation of names to internet addresses, several name server processes will be provided. The name server process will accept a name in the above form and will return a name, address pair.

The name server processes will have well-known addresses; addresses that are constant over long periods of time and published in documents such as "Assigned Numbers" [1].

A request sent to a name server is sent as a user datagram [2] with the following content:

```
+--------+--------+--------+--------+--------+--------+---\\---+
|        |        |                                              |
|  NAME  | LENGTH |                 NAME STRING                  |
|        |        |                                              |
+--------+--------+--------+--------+--------+--------+---\\---+
```

where:

NAME is a one octet code indicating that the following is a name,

LENGTH is a one octet count of the number of octets in the name string, and

NAME STRING is an ASCII character string of the form ! NET ! REST.

A reply to a successful translation is sent as a user datagram with the following content:

```
+--------+--------+--------+--------+--------+--------+---\\---+
|        |        |                                              |
|  NAME  | LENGTH |                 NAME STRING                  |
|        |        |                                              |
+--------+--------+--------+--------+--------+--------+---\\---+
|        |        |                                   |
| ADDRESS| LENGTH |          INTERNET ADDRESS         |
|        |        |                                   |
+--------+--------+--------+--------+--------+--------+
```

where:

ADDRESS is a one octet code indicating that the following is an internet address,

LENGTH is a one octet count (=4) of the length of the internet address, and

INTERNET ADDRESS is the internet address.

Actually a particular name might map to several internet addresses, in this case the response would include a list of internet addresses.

When a name is not found, an error is reported via a user datagram as follows:

```
+--------+--------+--------+--------+--------+--------+---\\---+
|        |        |                                            |
|  NAME  | LENGTH |                NAME STRING                 |
|        |        |                                            |
+--------+--------+--------+--------+--------+--------+---\\---+
|        |        | ERROR  |                                   |
| ERROR  | LENGTH | CODE   |          ERROR STRING             |
|        |        |        |                                   |
+--------+--------+--------+--------+--------+--------+---\\---+
```

where:

ERROR CODE specifies the error.

ERROR STRING explains the error.

Error Codes

The following error codes are defined:

CODE	MEANING
0	Undetermined or undefined error
1	Name not found
2	Improper name syntax

Communication with a Name Server Process

Communication with a name server process is via user datagrams. User datagrams do not guarantee reliable communication. Thus, some requests or replies may be lost.

The name server process is a transaction oriented process; furthermore, the nature of the transactions allows them to be processed in any order and even to be duplicated. This allows the use of a very simple communication protocol.

If a request is made to the name server process and no response is received within a reasonable time, then the requester should make the request again. This recovers from communication errors which cause the loss of either the request or the reply.

In order to use this simple strategy, care must be taken to allow replies to be properly matched with requests. The name server process does this by including in each reply a copy of the entire request.

The user datagram protocol does provide a checksum for the detection of errors.

Format

The requests and replies to and from a name server process are encoded as "items". An item consists of an item-code an item-length and the item-data. The item-length includes in its count the item-count and the item-length octets.

```
Item  :=  Item-Code  Item-Length  Item-Data

+--------+--------+--------+--------+--------+--------+---\\---+
|        |        |                                            |
|  Item  |  Item  |                 Item                       |
|  Code  | Length |                       Data                 |
|        |        |                                            |
+--------+--------+--------+--------+--------+--------+---\\---+
```

A request is typically one item, and a reply is typically two items.

```
+--------+--------+--------+--------+
|ItemCode|Item Len|... Item Data ...|
+--------+--------+--------+--------+
| ......... Item Data cont ........ |
+--------+--------+--------+--------+
| Item Data cont. |ItemCode|Item Len|
+--------+--------+--------+--------+
| ........... Item Data ........... |
+--------+--------+--------+--------+
```

Item Code Value Assignments:

```
  NAME    = 1

  ADDRESS = 2

  ERROR   = 3
```

Example

a typical request:

```
+--------+--------+--------+--------+
|   1    |   12   |   !    |   A    |
+--------+--------+--------+--------+
|   R    |   P    |   A    |   !    |
+--------+--------+--------+--------+
|   I    |   S    |   I    |   B    |
+--------+--------+--------+--------+
```

and the reply:

```
+--------+--------+--------+--------+
|   1    |   12   |   !    |   A    |
+--------+--------+--------+--------+
|   R    |   P    |   A    |   !    |
+--------+--------+--------+--------+
|   I    |   S    |   I    |   B    |
+--------+--------+--------+--------+
|   2    |   6    |   10   |   3    |
+--------+--------+--------+--------+
|   0    |   52   |
+--------+--------+
```

EXTENDED NAME SERVER

Several extensions have been proposed [4], the following two are adopted: partially specified names, and a service field.

In the first extension partially specified names are allowed and are indicated by the use of "wild card" fields or characters.

Wild Card Field	Meaning
*	All
~	Local (Same as that of the requestor)

Wild Card Character	Meaning
*	Any substring

Examples:

!~!*	all hosts on the net of the requestor.
!*!SRI*	all hosts with names whose first three characters are SRI on all nets

In general, there are three cases for each of the net and host fields. Using the symbols N for named network and H for named host the 9 cases are:

!~!~	local net, local host
!~!*	local net, all hosts
!~!H	local net, named host
!*!~	all nets, local host
!*!*	all nets, all hosts
!*!H	all nets, named host
!N!~	named net, local host
!N!*	named net, all hosts
!N!H	named net, named host

When such a request is processed and the result is more than one name/address pair, the response is all the pairs.

Examples:

```
  1)

    request:

      !ARPA!ISI*

    response:

      !ARPA!ISIA  10 1 0 22

      !ARPA!ISIB  10 3 0 52

      !ARPA!ISIC  10 2 0 22

      !ARPA!ISID  10 3 0 22

      !ARPA!ISIE  10 1 0 52

  2)

    request:

      !~!SRI-R2D2

    response:

      !ARPA!SRI-R2D2    10 3 0 51

      !SF-PR-1!SRI-R2D2  2 0 0 11

  3)

    request:

      !*!ISIA

    response:

      !ARPA!ISIA  10 1 0 22
```

The second extension is that a third field may be appended to the name. This is the SERVICE field.

```
  ! NET ! HOST ! SERVICE
```

To reply to a request of this form the name server must provide the internet address (net and host), the protocol number, and the port number.

```
    +--------+--------+--------+--------+--------+--------+---\\---+
    |        |        |                                            |
    |  NAME  | LENGTH |                NAME STRING                 |
    |        |        |                                            |
    +--------+--------+--------+--------+--------+--------+---\\---+
    |        |        |                                   |
    | ADDRESS| LENGTH |          INTERNET ADDRESS         |
    |        |        |                                   |
    +--------+--------+--------+--------+--------+--------+
    |        |                 |
    |PROTOCOL|       PORT      |
    |        |        |        |
    +--------+--------+--------+
```

Examples:

1)

request:

```
!ARPA!ISIA!TELNET
```

response:

```
!ARPA!ISIA!TELNET  10 1 0 22 6 0 23
```

2)

request:

```
!ARPA!*!NAME-SERVER
```

response:

```
!ARPA!SRI-KL!NAME-SERVER  10 1 0 2 17 42
```

References

[1] J. Postel. "Assigned Numbers," IEN 117, USC/Information Sciences Institute, August 1979.

[2] J. Postel. "User Datagram Protocol," IEN 88, USC/Information Sciences Institute, May 1979.

[3] J. Postel. "Internet Protocol," IEN 111, USC/Information Sciences Institute, August 1979.

[4] J. Pickens, E. Feinler, and J. Mathis. "The NIC Name Server -- A Datagram Based Information Utility," Proceedings of the Fourth Berkeley Conference on Distributed Data Management and Computer Networks, pp. 275-283, August 1979.

Acknowledgments

John Pickens contributed the ideas for the Extended Name Server.

IEN: 113
RFC: 759

INTERNET MESSAGE PROTOCOL

Jonathan B. Postel

August 1980

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

(213) 822-1511

August 1980

Internet Message Protocol

TABLE OF CONTENTS

Internet Message Protocol
Table Of Contents

PREFACE

This is the second edition of this specification and should be treated as a request for comments, advice, and suggestions. A great deal of prior work has been done on computer aided message systems and some of this is listed in the reference section. This specification was shaped by many discussions with members of the ARPA research community, and others interested in the development of computer aided message systems. This document was prepared as part of the ARPA sponsored Internetwork Concepts Research Project at ISI, with the assistance of Greg Finn, Suzanne Sluizer, Alan Katz, Paul Mockapetris, and Linda Sato.

Jon Postel

IEN: 113
RFC: 759

J. Postel
USC-ISI
August 1980

INTERNET MESSAGE PROTOCOL

1. INTRODUCTION

This document describes an internetwork message system. The system is designed to transmit messages between message processing modules according to formats and procedures specified in this document. The message processing modules are processes in host computers. Message processing modules are located in different networks and together constitute an internetwork message delivery system.

This document is intended to provide all the information necessary to implement a compatible cooperating module of this internetwork message delivery system.

1.1. Motivation

As computer supported message processing activities grow on individual host computers and in networks of computers, there is a natural desire to provide for the interconnection and interworking of such systems. This specification describes the formats and procedures of a general purpose internetwork message system, which can be used as a standard for the interconnection of individual message systems, or as a message delivery system in its own right.

This system also provides for the communication of data items beyond the scope of contemporary message systems. Messages can include data objects which could represent drawings, or facsimile images, or digitized speech. One can imagine message stations equipped with speakers and microphones (or telephone hand sets) where the body of a message or a portion of it is recorded digitized speech. The output terminal could include a graphics display, and the message might present a drawing on the display, and verbally (via the speaker) describe certain features of the drawing. This specification provides for the composition of complex data objects and their encoding in machine independent basic data elements.

1.2. Scope

The Internet Message Protocol is intended to be used for the transmission of messages between networks. It may also be used for the local message system of a network or host. This specification was developed in the context of the ARPA work on the interconnection of networks, but it is thought that it has a more general scope.

The focus here is on the internal mechanisms to transmit messages, rather than the external interface to users. It is assumed that a number of user interface programs will exist. These will be both new programs designed to work with this system and old programs designed to work with earlier systems.

1.3. The Internetwork Environment

The internetwork message environment consists of processes which run in hosts which are connected to networks which are interconnected by gateways. Each network consists of many different hosts. The networks are tied together through gateways. The gateways are essentially hosts on two (or more) networks and are not assumed to have much storage capacity or to "know" which hosts are on the networks to which they are attached [1,2].

1.4. Model of Operation

This protocol is implemented in a process called a Message Processing Module or MPM. The MPMs exchange messages by establishing full duplex communication and sending the messages in a fixed format described in this document. The MPM may also communicate other information by means of commands described here.

A message is formed by a user interacting with a User Interface Program or UIP. The user may utilize several commands to create various fields of the message and may invoke an editor program to correct or format some or all of the message. Once the user is satisfied with the message it is submitted for transmission by placing it in a data structure read by the MPM.

The MPM discovers the unprocessed input data (either by a specific request or by a general background search), examines it, and, using routing tables (or some other method), determines which outgoing link to use. The destination may be another user on the same host, one on another host on a network in common with the same host, or a user in another network.

In the first case, another user on this host, the MPM places the message in a data structure read by the destination user, where that user's UIP will look for incoming messages.

In the second case, the user on another host in this network, the MPM transmits the message to the MPM on that host. That MPM then repeats the routing decision, and discovering the destination is local to it, places the message in the data structure shared with the destination user.

In the third case, the user on a host in another network, the MPM transmits the messages to an MPM in that network if it knows how to establish a connection directly to it; otherwise, the MPM transmits the message to an MPM that is "closer" to the destination. An MPM might not know of direct connections to MPMs in all other networks, but it must be able to select a next MPM to handle the message for each possible destination network.

An MPM might know a way to establish direct connections to each of a few MPMs in other nearby networks, and send all other messages to a particular big brother MPM that has a wider knowledge of the internet environment.

An individual network's message system may be quite different from the internet message system. In this case, intranet messages will be delivered using the network's own message system. If a message is addressed outside the network, it is given to an MPM which then sends it through the appropriate gateways to (or towards) the MPM in the destination network. Eventually, the message gets to an MPM on the network of the recipient of the message. The message is then sent via the local message system to that host.

When local message protocols are used, special conversion programs are required to transform local messages to internet format when they are going out, and to transform internet messages to local format when they come into the local environment. Such transformations potentially lead to information loss. The internet message format attempts to provide features to capture all the information any local message system might use. However, a particular local message system is unlikely to have features equivalent to all the possible features of the internet message system. Thus, in some cases the transformation of an internet message to a local message discards some of the information. For example, if an internet message carrying mixed text and speech data in the body is to be delivered in a local system which only carries text, the speech data may be replaced by the text string "There was some speecn here". Such discarding of information is to be avoided when at all possible, and to be deferred as long as possible; still, the possibility remains that in some cases it is the only reasonable thing to do.

1.5. Interfaces

The MPM calls on a reliable communication procedure to communicate with other MPMs. This is a Transport Level protocol such as the Transmission Control Protocol (TCP) [3]. The interface to such a procedure conventionally provides calls to open and close connections, send and receive data on a connection, and some means to signal and be notified of special conditions (i.e., interrupts).

The MPM receives input and produces output through data structures that are produced and consumed respectively by user interface (or other) p-ograms.

2. FUNCTIONAL DESCRIPTION

This section gives an overview of the Internet Message System and its environment.

2.1. Terminology

The messages are routed by a process called the Message Processing Module or MPM. Messages are created and consumed by User Interface Programs (UIPs) in conjunction with users.

The basic unit transferred between MPMs is called a message. A message is made up of a transaction identifier (which uniquely identifies the message), a command (which contains the necessary information for delivery), and document. The document may have a header and a body.

For a personal letter the document body corresponds to the contents of the letter; the document header corresponds to the date line, greeting, and signature.

For an inter-office memo the document body corresponds to the text; the document header corresponds to the header of the memo.

The commands correspond to the information used by the Post Office or the mail room to route the letter or memo. Some of the information in the command is supplied by the UIP.

2.2. Assumptions

The following assumptions are made about the internetwork environment:

In general, it is not known what format intranet addresses will assume. Since no standard addressing scheme would suit all networks, it is safe to assume there will be several and that they will change with time. Thus, frequent software modification throughout all internet MPMs would be required if such MPMs were to know about the formats on many networks. Therefore, each MPM which handles internet messages is required to know only the minimum necessary to deliver them.

Each MPM is required to know completely only the addressing format of its own network(s). In addition, the MPM must be able to select an output link for each message addressed to another network or host. This does not preclude more intelligent behavior on the part of a given MPM, but at least this minimum is necessary. Each network has a unique name and numeric address. Such names and addresses are registered with a naming authority and may be listed in documents such as Assigned Numbers [4].

Each MPM will have a unique internet address. This feature will enable every MPM to place a unique "handling-stamp" on a message which passes through the MPM enroute to delivery.

2.3. General Specification

There are several aspects to a distributed service to be specified. First, there is the service to be provided; that is, the characteristics of the service as seen by its users. Second, there is the service it uses; that is, the characteristics it assumes to be provided by some lower level service. And third, there is the protocol used between the modules of the distributed service.

```
      User                                       User
         \                                       /
         UIP                                   UIP
           \                                   /
        --+-----------------------------------+-- Service
          |  \                             /  |  Interface
          |  +--------+              +--------+ |
          |  | Module | <--Protocol--> | Module | |
          |  +--------+              +--------+ |
          |        \                     /       |
          |        +----------------------+      |
          |        | Communication Service |      |
          |        +----------------------+      |
          |                                      |
          +--------------------------------------+
```

Message Service

Figure 1.

The User/Message Service Interface

The service the message delivery system provides is to accept messages conforming to a specified format, to attempt to deliver those messages, and to report on the success or failure of the delivery attempt. This service is provided in the context of an interconnected system of networks and may involve relaying a message through several intermediate MPMs via different communication services.

The Message/Communication Service Interface

The message delivery system calls on a communication service to transfer information from one MPM to another. There may be different communication services used between different pairs of MPMs, though all communication services must meet the service characteristics described below.

It is assumed that the communication service provides a reliable two-way data stream. Such a data stream can usually be obtained in computer networks from the transport level protocol, for example, the Transmission Control Protocol (TCP) [3]. In any case, the properties the communication service must provide are:

- o Logical connections for two way simultaneous data flow of arbitrary data (i.e., no forbidden codes). All data sent is delivered in order.
- o Simple commands to open and close the connections, and to send and receive data on the connections.
- o Controlled flow of data so that data is not transmitted faster that the receiver chooses to consume it (on the average).
- o Transmission errors are corrected without user notification or involvement of the sender or receiver. Complete breakdown on communication is reported to the sender or receiver.

The Message-Message Protocol

The protocol used between the distributed modules of the message delivery system, that is, the MPMs, is a small set of commands which convey requests and replies. These commands are encoded in a highly structured and rigidly specified format.

2.4. Mechanisms

MPMs are processes which use some communication service. A pair of MPMs which can communicate reside in a common interprocess communication environment. An MPM might exist in two (or more) interprocess communication environments, and such an MPM might act to relay messages between MPMs. Messages may be held for a time in an MPM; the total path required for delivery need not be available simultaneously.

From the time a message is accepted from a UIP by an MPM until it is delivered to a UIP by an MPM and an acknowledgment is returned to the originating UIP, the message is considered to be active in the message system.

```
  User                                                    User
    \                                                     /
     UIP                                                UIP
       \                                                /
   +-----------------------------------------------------------+
   |     \                                            /        |
   |   +------+                +------+              +------+  |
   |   | MPM  | <--Protocol--> | MPM  | <--Protocol--> | MPM  |  |
   |   +------+                +------+              +------+  |
   |      |                      /  \                   |      |
   |   +----------------------+   +----------------------+    |
   |   |Communication Service A|   |Communication Service B|    |
   |   +----------------------+   +----------------------+    |
   |                                                           |
   +-----------------------------------------------------------+
```

Message Service with Internal Relaying

Figure 2.

It should be clear that there are two roles an MPM can play, an end-point MPM or a relay MPM. Most MPMs will play both roles. A relay MPM acts to relay messages from one communication environment to another. An end-point MPM acts as a source or destination of messages.

The transfer of data between UIPs and MPMs is viewed as the exchange of data structures which encode messages. The transfer of data between MPMs is also in terms of the transmission of structured data.

```
            +------+     DATA      +------+
   USER-->| UIP  |-->STRUCTURES-->| MPM  |-->other
            +------+   +------+      +------+    MPMs
                       |      |
                       |  +------+
                       +--|      |
                          |  +------+
                          +--|      |
                             |      |
                             +------+

            +------+     DATA      +------+
   other-->| MPM  |-->STRUCTURES-->| UIP  |-->USER
   MPMs     +------+   +------+      +------+
                       |      |
                       |  +------+
                       +--|      |
                          |  +------+
                          +--|      |
                             |      |
                             +------+
```

Message Flow

Figure 3.

In the following, a message will be described as a structured data object represented in a particular kind of typed data elements. This is how a message is presented when transmitted between MPMs or exchanged between an MPM and a UIP. Internal to an MPM (or a UIP), a message may be represented in any convenient form.

2.5. Relation to Other Protocols

This protocol the benefited from the earlier work on message protocols in the ARPA Network [5,6,7,8,9], and the ideas of others about the design of computer message systems [10,11,12,13,14,15,16,17,18,19,20,21].

Figure 4 illustrates the place of the message protocol in the ARPA internet protocol hierarchy:

```
 +-------+ +-----+ +-------+ +-----+     +-----+
 |Telnet| | FTP | |Message| |Voice| ... |     | Application Level
 +-------+ +-----+ +-------+ +-----+     +-----+
       \    |    /             |           |
         +-----+           +-----+     +-----+
         | TCP |           | RTP | ... |     |  Host Level
         +-----+           +-----+     +-----+
            |                 |           |
       +-------------------------------------+
       |          Internet Protocol          |  Gateway Level
       +-------------------------------------+
                       |
         +-------------------------------+
         |     Local Network Protocol    |    Network Level
         +-------------------------------+
                       |
```

Protocol Relationships

Figure 4.

Note that "local network" means an individual or specific network. For example, the ARPANET is a local network.

The message protocol interfaces on one side to user interface programs and on the other side to a reliable transport protocol such as TCP.

In this internet message system the MPMs communicate directly using the lower level transport protocol. In the old ARPANET system, message transmission was part of the file transfer protocol.

```
          +-------+  +-----+   +--------+
          |Telnet|   | FTP |---|Message|        Application Level
          +-------+  +-----+   +--------+
                \      /
      +-----+  +-----+
      |Voice|--| NCP |                          Host Level
      +-----+  +-----+
                  |
                  |
                  |                             Gateway Level
                  |
                  |
           +-----------------+
           |    ARPA NET     |                  Network Level
           +-----------------+
```

Old ARPANET Protocols

Figure 5.

Note that in the old ARPANET protocols one can't send messages (or communicate in any way) to other networks since it has no gateway level or internet protocol [5].

3. DETAILED SPECIFICATION

The presentation of the information in this section is difficult since everything depends on everything, and since this is a linear medium it has to come in some order. In this attempt, a brief overview of the message structure is given, the detail of the message is presented in terms of data objects, the various data objects are defined, and finally the representation of the data elements is specified. Several aspects of the message structure are based on the NSW Transaction Protocol [22], and similar (but more general) proposals [23,24].

3.1. Overview of Message Structure

A message is normally composed of three parts: the identification, the command, and the document. Each part is in turn composed of data objects.

The identification part is composed of a transaction number assigned by the originating MPM and the MPM identifier.

The command part is composed of an operation type, an operation code, the arguments to the operation, error information, the destination mailbox, and a trace. The trace is a list of the MPMs that have handled this message.

The document part is a data structure. The message delivery system does not depend on the contents of the document part. A standard for the document part is defined in reference [25].

The following sections define the representation of a message as a structured object composed of other objects. Objects in turn are represented using a set of basic data elements.

The basic data elements are defined in section 3.7. In summary, these are exact forms for representing integers, strings, booleans, et cetera. There are also two elements for building data structures: list and property list. Lists are simple lists of elements, including lists. Property lists are lists of pairs of elements, where the first element of each pair names the pair. That is, a property list is a list of <name,value> pairs. In general, when an object is composed of multiple instances of a simpler object it is represented as a list of the simpler objects. When an object is composed of a variety of simpler objects it is represented as a property list of the simpler objects. In most uses of the property list representation, the presence of <name,value> pairs in addition to those specifically required is permitted.

3.2. Message Structure

An internet message is composed of two or three parts. The first is the Identification which identifies the transaction; the second is the Command; and the optional third part is the Document.

When shipped between two MPMs, a message will take the form of a property list, with the <name,value> pairs in this order.

MESSAGE is:

```
( Identification, Command [, Document ] )
```

It is convenient to batch several messages together, shipping them as a unit from one MPM to another. Such a group of messages is called a message-bag.

A message-bag will be a list of Messages; each Message is of the form described above.

MESSAGE-BAG is:

```
( Message, Message, ... )
```

The Identification

This is the transaction identifier. It is assigned by the originating MPM. The identification is composed of the MPM identifier, and a transaction number unique in that context for this message.

The Command

The command is composed of a mailbox, an operation code, the arguments to that operation, some error information, and a trace of the route of this message. The command is implemented by a property list which contains <name,value> pairs, where the names are used to identify the associated argument values.

The Document

The document portion of an internet message is optional and when present is a data structure as defined in [25].

3.3. Identification

Each message must have a unique identifier while it exists in the message delivery system. This is provided by the combination of the unique identifier of the MPM and a unique transaction number chosen for the message by this MPM.

```
IDENTIFICATION is:

   ( mpm-identifier, transaction-number )
```

The mpm-identifier is based on the host address of the computer in which the MPM resides. If there is more than one MPM in a host the mpm-identifier must be extended to distinguish between the co-resident MPMs.

3.4. Command

This section describes the commands MPMs use to communicate between themselves. The commands come in pairs, with each request having a corresponding reply.

```
COMMAND is:

   ( mailbox, operation, [arguments,]
                             [error-class, error-string,] trace )
```

The mailbox is the "To" specification of the message. Mailbox is a property list of general information, some of which is the essential information for delivery, and some of which could be extra information which may be helpful for delivery. Mailbox is different from address in that address is a very specific property list without extra information. The mailbox includes a specification of the user, when a command is addressed to the MPM itself (rather than a user it serves) the special user name "*MPM*" is specified.

The operation is the name of the operation or procedure to be performed.

The arguments to the operation vary from operation to operation.

The error information is composed of a error class code and a character string, and indicates what, if any, error occurred. The error information is normally present only in replies, and not present in requests.

The trace is a list of the MPMs that have handled the message. Each MPM must add its handling-stamp to the list.

3.4.1. Command: DELIVER

function: Sends a document to a mailbox.

reply: The reply is ACKNOWLEDGE.

arguments:

type-of-service: one or more of the following:

"REGULAR"	regular delivery
"FORWARD"	message forwarding
"GENDEL"	general delivery
"PRIORITY"	priority delivery

3.4.2. Command: ACKNOWLEDGE

function: Reply to DELIVER.

arguments:

reference: the identifier of the originating message.

address:

The address is the final mailbox the message was delivered to. This would be different from the original mailbox if the message was forwarded, and is limited to the essential information needed for delivery.

type-of-service: one of the following:

"GENDEL"	message was accepted for general delivery
"REGULAR"	message was accepted for normal delivery
"PRIORITY"	message was accepted for priority delivery

error-class:
error-string:

If the document was delivered successfully, the error information has class 0 and string "ok". Otherwise, the error information has a non-zero class and the string would be one of "no such user", "no such host", "no such network", "address ambiguous", or a similar response.

trail: the trace from the DELIVER command.

3.4.3. Command: PROBE

function: Finds out if specified mailbox (specified in mailbox of the command) exists at a host.

reply: The reply is RESPONSE.

arguments: none.

3.4.4. Command: RESPONSE

function: Reply to PROBE.

arguments:

reference: the identification of the originating PROBE.

address: a specific address.

error-class:
error-string:

If the mailbox was found the error class is 0 and the error string is "OK". If the mailbox has moved and a forwarding address in known the error class is 1 and the error string is "Mailbox moved, see address". Otherwise the error class is greater than 1 and the error string may be one of the following: "Mailbox doesn't exist", "Mailbox full", "Mailbox has moved, try the new location indicated in the address".

trail: the trace which came from the originating PROBE.

3.4.5. Command: CANCEL

function: Abort request for specified transaction.

reply: The reply is CANCELED.

arguments:

reference: identification of transaction to be canceled.

3.4.6. Command: CANCELED

function: Reply to CANCEL.

arguments:

reference: identification of canceled transaction.

error-class:
error-string:

If the command was canceled the error class is 0 and the error string is "OK". Otherwise the error class is positive and the error string may be one of the following: "No such transaction", or any error for an unreachable mailbox.

trail: the trace of the CANCEL command.

To summarize again, a command generally consists of a property list of the following objects:

name	value
mailbox	property list of address information
operation	name of operation
arguments	---
error-class	numeric class of the error
error-string	text description of the error
trace	list (handling-stamp, ...)

3.5. Document

The actual document follows the command. The message delivery system does not depend on the document, examine it, or use it in any way. The standard for the contents of the document is reference [25]. The document must be the last <name,value> pair in the message property list.

3.6. Message Objects

In the composition of messages, we use a set of objects such as mailbox or date. These objects are encoded in basic data elements. Some objects are simple things like integers or character strings, other objects are more complex things built up of lists or property lists.

The following is a list of the objects used in messages. The object descriptions are in alphabetical order.

Action

The type of handling action taken by the MPM when processing a message. One of ORIGIN, RELAY, FORWARD, or DESTINATION.

Address

Address is intended to contain the minimum information necessary to deliver a message, and no more (compare with mailbox).

An address is a property list. An address contains the following <name,value> pairs:

name	description
NET	network name
HOST	host name
USER	user name

or:

name	description
MPM	mpm-identifier
USER	user name

Answer

A yes (true) or no (false) answer to a question.

Arguments

Many operations require arguments, which differ from command to command. This "object" is a place holder for the actual arguments when commands are described in a general way.

City

The character string name of a city.

Command

```
(mailbox, operation [ ,arguments ]
                              [ ,error-class, error-string ], trace)
```

Country

The character string name of a country.

Date

The date and time are represented according to the International Standards Organization (ISO) recommendations [26,27,28]. Taken together the ISO recommendations 2014, 3307, and 4031 result in the following representation of the date and time:

```
yyyy-mm-dd-hh:mm:ss,fff+hh:mm
```

Where yyyy is the four-digit year, mm is the two-digit month, dd is the two-digit day, hh is the two-digit hour in 24 hour time, mm is the two-digit minute, ss is the two-digit second, and fff is the decimal fraction of the second. To this basic date and time is appended the offset from Greenwich as plus or minus hh hours and mm minutes.

The time is local time and the offset is the difference between local time and Coordinated Universal Time (UTC). To convert from local time to UTC algebraically subtract the offset from the local time.

```
For example, when the time in
          Los Angeles is  14:25:00-08:00
          the UTC is      22:25:00

or when the time in
          Paris is        11:43:00+01:00
          the UTC is      10:43:00
```

Document

The document is the user's composition and is not used by the message delivery system in any way.

Error-Class

A numeric code for the class of the error. The error classes are coded as follows:

```
= 0: indicates success, no error.
  This is the normal case.
= 1: failure, address changed.
  This error is used when forwarding is possible, but not allowed
  by the type of service specified.
= 2: failure, resources unavailable.
  These errors are temporary and the command they respond to may
  work if attempted at a later time.
= 3: failure, user error.
  For example, unknown operation, or bad arguments.
= 4: failure, MPM error.  Recoverable.
  These errors are temporary and the command they respond to may
  work if attempted at a later time.
= 5: failure, MPM error.  Permanent.
  These errors are permanent, there is no point in trying the same
  command again.
= 6: Aborted as requested by user.
  The response to a successfully canceled command.
```

Error-String

This is a character string describing the error. Possible errors:

error-string	error-class
No errors	0
Ok	0
Mailbox Moved, see address	1
Mailbox Full, try again later	2
Syntax error, operation unrecognized	3
Syntax error, in arguments	3
No Such User	3
No Such Host	3
No Such Network	3
No Such Transaction	3
Mailbox Does Not Exist	3
Ambiguous Address	3
Server error, try again later	4
No service available	5
Command not implemented	5
Aborted as requested by user	6

Handling-Stamp

The handling-stamp indicates the MPM, the date (including the time) that a message was processed by an MPM, and the type of handling action taken.

(mpm-identifier, date, action)

Host

The character string name of a host.

Identification

This is the transaction identifier associated with a particular message. It is the transaction number, and the MPM identifier of the originating MPM.

(mpm-identifier, transaction-number)

Internet Address

This identifies a host in the ARPA internetwork environment. When used as a part of identification, it identifies the originating host of a message. The internet address is a 32 bit number, the higher order 8 bits identify the network, and the lower order 24 bits identify the host on that network [2]. For use in the MPMs the internet address is divided into eight bit fields and the value of each field is represented in decimal digits. For example, the ARPANET address of ISIE is 167837748 and is represented as 10,1,0,52. Further, this representation may be extended to include an address within a host, such as the TCP port of the MPM, for example, 10,1,0,52,0,45.

Mailbox

This is the destination address of a user of the internetwork mail system. Mailbox contains information such as network, host, location, and local user indentifier of the recipient of the message. Some information contained in mailbox may not be necessary for delivery.

As an example, when one sends a message to someone for the first time, he may include many items which are not necessary simply to insure delivery. However, once he gets a reply to this message, the reply will contain an Address (as opposed to Mailbox) which may be used from then on.

A mailbox is a property list. A mailbox might contain the following <name,value> pairs:

name	description
MPM	mpm-identifier
NET	network name
HOST	host name
PORT	address of MPM within the host
USER	user name
ORG	organization name
CITY	city
STATE	state
COUNTRY	country
ZIP	zip code
PHONE	phone number

The minimum mail box is an Address.

MPM-Identifier

The internetwork address of the MPM. This may be the ARPA Internet Address or an X.121 Public Data Network Address [29]. The mpm-identifier is a property list which has one <name,value> pair. This unusual structure is used so that it will be easy to determine the type of address used.

Network

This character string name of a network.

Operation

This names the operation or procedure to be performed. It is a character string name.

Organization

This character string name of a organization.

Phone

This character string name representation of a phone number. For example the phone number of ISI is 1 (international region) + 213 (area code) + 822 (central office) + 1511 (station number) = 12138221511.

Port

This names the port or subaddress within a host of the MPM. The default port for the MPM is 45 (55 octal) [4].

Reference

The reference is an identification from an earlier message.

State

The character string name of a state.

Trace

Each MPM that handles the message must add its handling-stamp to this list. This will allow detection of messages being sent in a loop within the internet mail system, and aid in fault isolation.

Trail

When a message is sent through the internetwork environment, it acquires this trace, a list of MPMs that have handled the message. This list is then carried as the trail in a reply or acknowledgment of that message. Requests and replies always have a trace and each MPM adds its handling-stamp to this trace. Replies, in addition, have a trail which is the complete trace of the original message.

Transaction Number

This is a number which is uniquely associated with this transaction by the originating MPM. It identifies the transaction. (A transaction is a message and acknowledgment.) A transaction number must be unique during the time which the message (a request or reply) containing it could be active in the network.

Type-of-Service

A service parameter for the delivery of a message, for instance a message could be delivered (REGULAR), forwarded (FORWARD), turned over to general delivery (GENDEL) (i.e., allow a person to decide how to further attempt to deliver the message), or require priority handling (PRIORITY).

User

The character string name of a user.

X121 Address

This identifies a host in the Public Data Network environment. When used as a part of identifier, it identifies the originating host of a message. The X121 address is a sequence of up to 14 digits [29]. For use in the MPMs the X121 address is represented in decimal digits.

Zip Code

The character string representation of a postal zip code. The zip code of ISI is 90291.

3.7. Data Elements

The data elements defined here are similar to the data structure and encoding used in NSW [30].

Each of the diagrams which follow represent a sequence of octets. Field boundaries are denoted by the "|" character, octet boundaries by the "+" character. Each element begins with a one-octet code. The order of the information in each element is left-to-right. In fields with numeric values the high-order (or most significant) bit is the left-most bit. For transmission purposes, the leftmost octet is transmitted first. Cohen has described some of the difficulties in mapping memory order to transmission order [31].

```
Code  Type          Representation
----  ----          --------------

                    +------+
   0  No Operation  |  0   |
                    +------+

                    +------+------+------+------+------
   1  Padding       |  1   |     octet count    | Data ...
                    +------+------+------+------+------

                    +------+------
   2  Boolean       |  2   | 1/0  |
                    +------+------+

                    +------+------+------+
   3  Index         |  3   |    Data     |
                    +------+------+------+

                    +------+------+------+------+------+
   4  Integer       |  4   |           Data            |
                    +------+------+------+------+------+
     Extended       +-----+-----+-----+-----+-----
   5 Precision      |  5  |    octet count    | Data ...
     Integer        +-----+-----+-----+-----+-----

                    +-----+-----+-----+-----+-----
   6 Bit String     |  6  |     bit count     | Data ...
                    +-----+-----+-----+-----+-----

                    +-----+-----+-----
   7 Name String    |  7  |count|  Data ...
                    +-----+-----+-----

                    +-----+-----+-----+-----+-----
   8 Text String    |  8  |    octet count    | Data ...
                    +-----+-----+-----+-----+-----

                    +-----+-----+-----+-----+-----
   9 List           |  9  |    octet count    | Data ...
                    +-----+-----+-----+-----+-----

                    +-----+-----+-----+-----+-----
   10 Proplist      | 10  |    octet count    | Data ...
                    +-----+-----+-----+-----+-----

                    +-----+
   11 End of List   | 11  |
                    +-----+
```

Element code 0 (NOP) is an empty data element used for padding when it is necessary. It is ignored.

Element code 1 (PAD) is used to transmit large amounts of data with a message for test or padding purposes. The type-octet is followed by a three-octet count of the number of octets to follow. No action is taken with this data but the count of dummy octets must be correct to indicate the next element code.

Element code 2 (BOOLEAN) is a boolean data element. The octet following the type-octet has the value 1 for True and 0 for False.

Element code 3 (INDEX) is a 16-bit unsigned integer datum. Element code 3 occupies only 3 octets.

Element code 4 (INTEGER) is a signed 32-bit integer datum. This will always occupy five octets. Representation is two's complement.

Element code 5 (EPI) is an extended precision integer. The type octet is followed by a three-octet count of the number of data octets to follow. Representation is two's complement.

Element code 6 (BITSTR) is a bit string element for binary data. The bit string is padded on the right with zeros to fill out the last octet if the bit string does not end on an octet boundary. This data type must have the bit-count in the three-octet count field instead of the number of octets.

Element code 7 (NAME) is used for the representation of character string names (or other short strings). The type octet is followed by a one-octet count of the number of characters (one per octet) to follow. Seven bit ASCII characters are used, right justified in the octet. The high order bit in the octet is zero.

Element code 8 (TEXT) is used for the representation of text. The type octet is followed by a three-octet count of the number of characters (one per octet) to follow. Seven bit ASCII characters are used, right justified in the octet. The high order bit in the octet is zero.

Element code 9 (LIST) can be used to create structures composed of other elements. The three-octet octet count specifies the number of octets in the whole list (i.e., the number of octets following this count field to the end of the list, not including the ENDLIST octet). The two-octet item count contains the number of elements which follow. Any element may be used including list itself.

```
+------+------+------+------+------+------+
|   9  |     octet count      | item count |
+------+------+------+------+------+------+
                          +------+------/---+
                 repeated |      element    |
                          +------+------/---+
                                       +--------+
                                       |ENDLIST|
                                       +--------+
```

In some situations it may not be possible to know the length of a list until the head of it has been transmitted. To allow for this a special ENDLIST element is defined. A list of undetermined length is transmitted with the octet count cleared to zero, and the item count cleared to zero. A null or empty List, one with no elements, has an octet count of two (2) and an item count of zero (0). The ENDLIST element always follows a LIST, even when the length is determined.

Element code 10 (PROPLIST) is the Property List element. It is a special case of the list element, in which the elements are in pairs and the first element of each pair is a name. It has the following form:

```
+-----+-----+-----+-----+-----+
| 10  |     octet count | pair |
+-----+-----+-----+-----+-----+
                  +-----+-----/---+-----+-----/---+
         repeated | name element  | value element  |
                  +-----+-----/---+-----+-----/---+
                                              +-------+
                                              |ENDLIST|
                                              +-------+
```

The Property List structure consists of a set of unordered <name,value> pairs. The pairs are composed of a name which must be a NAME element and a value which may be any kind of element. Following the type code is a three-octet octet count of the following octets. Following the octet count is a one-octet pair count of the number of <name,value> pairs in the property list.

The name of a <name,value> pair is to be unique within the property list, that is, there shall be at most one occurrence of any particular name in one property list.

In some situations it may not be possible to know the length of a property list until the head of it has been transmitted. To allow for this the special ENDLIST element is defined. A property list of undetermined length is transmitted with the octet count cleared to zero, and the pair count cleared to zero. A null or empty property list, one with no elements, has an octet count of one (1) and an pair count of zero (0). The ENDLIST element always follows a property list, even when the length is determined.

Element code 11 (ENDLIST) is the end of list element. It marks the end of the corresponding list or property list.

Structure Sharing

When messages are batched in message-bags for transmission, it may often be the case that the same document will be sent to more than one recipient. Since the document portion can usually be expected to be the major part of the message, much repeated data would be sent if a copy of the document for each recipient were to be shipped in the message-bag.

To avoid this redundancy, messages may be assembled in the message-bag so that actual data appears on its first occurrence and only references to it appear in later occurrences. When data is shared, the first occurrence of the data will be tagged, and later locations where the data should appear will only reference the earlier tagged location. All references to copied data point to earlier locations in the message-bag. The data to be retrieved is indicated by the tag.

This is a very general sharing mechanism. PLEASE NOTE THAT THE MPM WILL NOT SUPPORT THE FULL USE OF THIS MECHANISM. THE MPM WILL ONLY SUPPORT SHARING OF WHOLE DOCUMENTS. No other level of sharing will be supported by the MPMs.

This sharing mechanism may be used within a document as long as all references refer to tags within the same document.

Sharing is implemented by placing a share-tag on the first occurrence of the data to be shared, and placing a share-reference at the locations where copies of that data should occur.

```
                            +------+-----+------+
  12 Share Tag              |  12  | share-index |
                            +------+-----+------+


                            +------+-----+------+
  13 Share Reference        |  13  | share-index |
                            +------+-----+------+
```

Element code 12 (S-TAG) is a share tag element. The two octets following the type-octet specify the shared data identification code for the following data element. Note that s-tag is not a DATA element, in the sense that data elements encode higher level objects.

Element code 13 (S-REF) is a share reference element. The two octets following the type-octet specify the referenced shared data identification code.

An example of using this mechanism is

```
( ( <a>, <b> ) ( <c>, <b> ) )
```

could be coded as follows to share <b>

```
( ( <a>, <s-tag-1><b> ) ( <c>, <s-ref-1> ) )
```

To facilitate working with structures which may contain shared data, the two high-order bits of the list and property list element codes are reserved for indicating if the structure contains data to be shared or contains a reference to shared data. That is, if the high-order bit of the list or property list element code octet is set to one then the property list contains a share-reference to shared data. Or, if the second high-order bit is set to one the structure contains a share-tag for data to be shared.

The example above is now repeated in detail showing the use of the high-order bits.

```
+------+------+------+------+------+------+------+-------+
|11 - 9|01 - 9|  <a> |  12  |   0  |   1  |  <b> |  11   |
+------+------+------+------+------+------+------+-------+
                      +------+------+------+------+------+------+------+
                      |10 - 9|  <c> |  13  |   0  |   1  |  11  |  11  |
                      +------+------+------+------+------+------+------+
```

It is not considered an error for an element to be tagged but not referenced.

A substructure with internal sharing may be created. If such a substructure is closed with respect to sharing -- that is, all references to its tagged elements are within the substructure -- then there is no need for the knowledge of the sharing to propagate up the hierarchy of lists. For example, if the substructure is:

```
00-LIST ( a b c b )
```

which with sharing is:

```
11-LIST ( a T1:b c R1 )
```

When this substructure is included in a large structure the high order bits can be reset since the substructure is closed with respect to sharing. For example:

```
  00-LIST ( x 11-LIST ( a T1:b c R1 ) y )
```

Note: While sharing adds transmission and memory efficiency, it is costly in processing to separate shared elements. This is the main reason for restricting the sharing supported by the MPM. At some later time these restrictions may be eased.

It is possible to create loops, "strange loops" and "tangled hierarchies" using this mechanism [32]. The MPM will not check for such improper structures within documents, and will not deliver messages involved in such structures between documents.

If an encryption scheme is used to ensure the privacy of communication it is unlikely that any parts of the message can be shared. This is due to the fact that in most case the encryption keys will be specific between two individuals. There may be a few cases where encrypted data may be shared. For example, all the members of a committee may use a common key when acting on committee business, or in a public key scheme a document may be "signed" using the private key of the sender and inspected by anyone using the public key of the sender.

4. OTHER ISSUES

This section discusses various other issues that need to be dealt with in a computer message system.

4.1. Accounting and Billing

Accounting and billing must be performed by the MPM. The charge to the user by the message delivery system must be predictable, and so cannot depend on the actual cost of sending a particular message which incurs random delays, handling and temporary storage charges. Rather, these costs must be aggregated and charged back to the users on an average cost basis. The user of the service may be charged based on the destination or distance, the length of the message, type of service, or other parameters selected as the message is entered into the delivery system, but must not depend on essentially random handling by the system of the particular message.

This means it is pointless to have each message carry an accumulated charge (or list of charges). Rather, the MPM will keep a log of messages handled and periodically bill the originators of those messages.

It seems that the most reasonable scheme is to follow the practice of the international telephone authorities. In such schemes the authority where the message originates bills the user of the service for the total charge. The authorities assist each other in providing the international message transfer and the authorities periodically settle any differences in accounts due to an imbalance in international traffic.

Thus the MPMs will keep logs of messages handled and will periodically charge their neighboring MPM for messages handled for them. This settlement procedure is outside the message system and between the administrators of the MPMs.

As traffic grows it will be impractical to log every message individually. It will be necessary to establish categories of messages (e.g., short, medium, large) and only count the number in each category.

The MPM at the source of the message will have a local means of identifying the user to charge for the message delivery service. The relay and destination MPMs will know which neighbor MPMs to charge (or settle with) for delivery of their messages.

4.2. Addressing and Routing

The mailbox provides for many types of address information. The MPMs in the ARPA internet can most effectively use the internet address [2]. The use of other address information is not yet very clear. Some thoughts on addressing issues may be found in the references [33,34,35].

An MPM sometimes must make a routing decision when it is acting as a relay-MPM (or source MPM). It must be able to use the information from the mailbox to determine to which of its neighbor MPMs to send the message. One way this might be implemented is to have a table of destination networks with corresponding neighbor MPM identifiers to use for routing toward that network.

It is not expected that such routing tables would be very dynamic. Changes would occur only when new MPMs came into existence or MPMs went out of service for periods of days.

Even with relatively slowly changing routing information the MPMs need an automatic mechanism for adjusting their routing tables. The routing problem here is quite similar to the problem of routing in a network of packet switches such as the ARPANET IMPs or a set of internet gateways. A great deal of work has been done on such problems and many simple schemes have been found faulty. There are details of these procedures which may become troublesome when the number of nodes grows beyond a certain point or the frequency of update exchanges gets large.

A basic routing scheme is to have a table of <network-name, mpm-identifier> pairs. The MPM could look up the network name found in the mailbox of the message and determine the internet mpm-identifier of the next MPM to which to route the message. To permit automatic routing updates another column would be added to indicate the distance to the destination. This could be measured in several ways, for example, the number of relay MPM (or hops) to the final destination. In this case each entry in the table is a triple of <network-name, mpm-identifier, distance>.

To update the routing information when changes occur an MPM updates its table. It then sends to each next MPM in its table a table of pairs <network-name, distance>, which say in effect "I can get a message to each of these networks with "cost" distance." An MPM which receives such an update will add to all the distances the distance to the MPM sending the update (e.g., one hop) and compare the information with its own table.

If the update information shows that the distance to a destination network is now smaller via the MPM which sent the update, the MPM changes its own table to reflect the better route, and the new distance. If the MPM has made changes in its table it sends update information to all the MPMs listed as next-MPMs in its table.

One further feature is that when a new network comes into existence an entry must be added to the table in each MPM. The MPMs should therefore expect the case that update information may contain entries which are new networks, and in such an event add these entries to their own tables.

When a new MPM comes into existence it will have an initial table indicating that it is a good route (short distance) to the network it is in, and will have entries for a few neighbor networks. It will send an initial "update" to those neighbor MPMs which will respond with more complete tables, thus informing the new MPM of routes to many networks.

This routing update mechanism is a simple minded scheme and may have to be replaced as the system of MPMs grows. In addition it ignores the opportunity for MPMs to use other information (besides destination network name) for routing. MPMs may have tables that indicate next-MPMs based on city, telephone number, organization, or other categories of information.

4.3. Encryption

It is straightforward to add the capability to have the document portion of messages either wholly or partially encrypted. An additional basic data element is defined to carry encrypted data. The data within this element may be composed of other elements, but that could only be perceived after the data was decrypted.

```
                    +-------+-------+-------+-------+
   14 Encrypt       |  14   |      octet count      |
                    +-------+-------+-------+-------+

                            +-------+-------+-------+--------
                            |alg id|    key id     | Data ...
                            +-------+-------+-------+--------
```

Element code 14 (ENCRYPT) is used to encapsulate encrypted data. The format is the one-octet type code, the three-octet octet count, a one-octet algorithm identifier, a two-octet key identifier, and count octets of data. Use of this element indicates that the data it contains is encrypted. The encryption scheme is indicated by the algorithm identifier, and the key used is indicated by the key identifier (this is not the key itself). The NBS Data Encryption Standard (DES) [36], public key encryption [37,38,39], or other schemes may be used.

To process this data element, the user is asked for the appropriate key and the data can then be decrypted. The data thus revealed will be in the form of complete data element fields. Encryption cannot occur over a partial field. The revealed data is then processed normally.

Note that there is no reason why all fields of a document could not be encrypted including all document header information such as From, Date, etc.

5. THE MPM: A POSSIBLE ARCHITECTURE

The heart of the internet message system is the MPM which is responsible for routing and delivering messages. Each network must have at least one MPM. These MPMs are logically connected together, and internet mail is always transferred along logical channels between them. The MPMs interface with existing local message systems.

Since the local message system may be very different from the internet system, special programs may be necessary to convert incoming internet messages to the local format. Likewise, messages outgoing to other networks may be converted to the internet format and sent via the MPMs.

5.1. Interfaces

User Interface

It is assumed that the interface between the MPM and the UIP provides for passing data structures which represent the document portion of the message. In addition, this interface must pass the delivery address information (which becomes the information in the mailbox field of the command). It is assumed that the information is passed between the UIP and the MPM via shared files, but this is not the only possible mechanism. These two processes may be more strongly coupled (e.g., by sharing memory), or less strongly coupled (e.g., by communicating via logical channels).

When a UIP passes a document and a destination address to the MPM, the MPM assigns a transaction-number and forms a message to send. The MPM must record the relationship between the transaction-number, the document, and the UIP, so that it can inform the UIP about the outcome of the delivery attempt for that document when the acknowledgment message is received at some later time.

Assuming a file passing mode of communication between the UIP and the MPM the sending and receiving of mail might involve the following interactions:

A user has an interactive session with a UIP to compose a document to send to a destination (or list of destinations). When the user indicates to the UIP that the document is to be sent, the UIP places the information into a file for the MPM. The UIP may then turn to the next request of the user.

The MPM finds the file and extracts the the information. It creates a message, assigning a transaction-number and forming a deliver command. The MPM records the UIP associated with this message. The MPM sends the message toward the destination.

When the MPM receives a deliver message from another MPM addressed to a user in its domain, it extracts the document and puts it into a file for the UIP associated with the destination user. The MPM also sends an acknowledge message to the originating MPM.

When the MPM receives an acknowledgment for a message it sent, the MPM creates a notification for the associated UIP and places it in a file for that UIP.

The format of these files is up to each UIP/MPM interface pair. One reasonable choice is to use the same data structures used in the MPM-MPM communication.

Communication Interface

It is assumed here that the MPMs use an underlying communication system, and TCP [3] has been taken as the model. In particular, the MPM is assumed to be listening for a TCP connection on a TCP port, i.e., it is a server process. The port is either given explicitly in the mpm-identifier or takes the default vaule 45 (55 octal) [4]. Again, this is not intended to limit the implementation choices; other forms of interprocess communication are allowed, and other types of physical interconnection are permitted. One might even use dial telephone calls to interconnect MPMs (using suitable protocols to provide reliable communication) [12,19,20,21].

5.2. The MPM Organization

Messages in the internet mail system are transmitted in lists called message-bags (or simply bags), each bag containing one or more messages. Each MPM is expected to implement functions which will allow it to deliver local messages it receives and to forward non-local ones to other MPMs presumably closer to the message's destination.

Loosely, each MPM can be separated into six components:

1--Acceptor

Receives incoming message-bags, from other MPMs, from UIPs, or from conversion programs.

2--Message-Bag Processor

Splits a bag into these three portions:

a. Local Host Messages
b. Local Net Messages
c. Foreign Net Messages

3--Local Host Delivery

Delivers local host messages, may call on conversion program.

4--Local Net Delivery

Delivers local net messages, may call on conversion program.

5--Foreign Net Router

Forms message-bags for transmission to other MPMs and determines the first step in the route.

6--Foreign Net Sender

Activates transmission channels to other MPMs and sends message-bags to foreign MPMs.

If the local net message system uses the protocol of the MPMs, then there need be no distinction between local net and foreign net delivery procedures.

All of these components can be thought of as independent. The function of the Acceptor is to await incoming message-bags and to insert them into the Bag-Input Queue.

The Bag-Input queue is read by the message-bag Processor which will separate and deliver suitable portions of the message-bags it retrieves from the queue to one of three queues:

a. Local Host Queue
b. Local Net Queue
c. Foreign Net Queue

When an MPM has a message to send to another MPM, it must add its own handling-stamp to the trace field of the command. The trace then becomes a record of the route the message has taken. An MPM should examine the trace field to see if the message is in a routing loop. All commands require the return of the trace as a trail in the matching reply command.

All of these queues have as elements complete message-bags created by selecting messages from the input message-bags.

The Local Host queue serves as input to the Local Host Delivery process. This component is responsible for delivering messages to its local host. It may call on a conversion program to reformat the messages into a form the local protocol will accept. This will probably involve such things as copying shared information.

The Local Net queue serves as input to the Local Net Delivery process. This component is responsible for delivering messages to other hosts on its local net. It must be capable of handling whatever error conditions the local net might return, and should include the ability to retransmit. It may call on a conversion program to reformat the messages into a form the local protocol will accept. This will probably involve such things as copying shared information.

The other two processes are more closely coupled. The Foreign Net Router takes its input bags from the Foreign Net Queue. From the internal information it contains, it determines which of the MPMs to which it is connected should receive the bag.

It then places the bag along with the routing information into the Send Mail Queue. The Foreign Net Sender retrieves it from that queue and transmits it across a channel to the intended foreign MPM. The Sender aggregates messages to the same next MPM into a bag.

The Foreign Net Router should be capable of receiving external input to its routing information table. This may come from the Foreign Net Sender in the case of a channel going down, requiring a decision to either postpone delivery or to determine a new route. The Router is responsible for maintaining sufficient information to determine where to send any incoming message-bag.

Forwarding

An MPM may have available information on the correct mailboxes of users which are not at its location. This information, called a forwarding data base, may be used to return the correct address in response to a probe command, or to actually forward a deliver command (if allowed by the type of service).

Because such forwarding may cause the route of a message to pass through an MPM already on the trace of this message, only the portion of the trace back to the most recent forward action should be used for loop detection by a relay relaying MPM, and only the forward action entries in the trace should be checked by a forwarding MPM.

Implementation Recommendations

Transaction numbers can be assigned sequentially, with wrap around when the highest value is reached. This should ensure that no message with a particular transaction number from this source is in the network when another instance of this transaction number is chosen.

The processing to separate shared elements when the routes of the shared elements diverge while still preserving the sharing possible appears to be an O(N*M**2) operation where N is the number of distinct objects in a message which may be shared across message boundaries and M is the number of messages in the bag.

Also note that share-tags may be copied into separate message bags which are not referenced. These could be removed with another pass over the message bag.

6. EXAMPLES & SCENARIOS

Example 1: Message Format

Suppose we want to send the following message:

```
Date: 1979-03-29-11:46-08:00
From: Jon Postel <Postel@ISIE>
Subject: Meeting Thursday
To: Danny Cohen <Cohen@USC-ISIB>
CC: Linda

Danny:

Please mark your calendar for our meeting Thursday at 3 pm.

--jon.
```

It will be encoded in the structured format. The following will present successive steps in the top down generation of this message. The actual document above will not be shown in the coded form.

```
1.  message

2.  (identification, command, document)

3.  (ID:(mpm-identifier, transaction-number),
     CMD:(MAILBOX:mailbox, OPERATION:operation,
                                      arguments, TRACE:trace),
     DOC:<<document>>)

4.  (ID:(mpm-identifier, transaction-number),
     CMD:(MAILBOX:mailbox, OPERATION:operation,
                           TYPE-OF-SERVICE:regular, TRACE:trace),
     DOC:<<document>>)

5.  (ID:(MPM:(IA:12,1,0,52,0,45), TRANSACTION:37),
     CMD:(MAILBOX:(MPM:(IA:12,3,0,52,0,45),
                   NET:ARPA,
                   HOST:ISIB,
                   PORT:45,
                   USER:Cohen),
          OPERATION:DELIVER,
          TYPE-OF-SERVICE:REGULAR,
          TRACE:(MPM:(IA:12,1,0,52,0,45)
                 DATE:1979-03-29-11:46-08:00,
                 ACTION:ORIGIN)),
     DOC:<<document>>)
```

```
  6.  PROPLIST:(
        ID:PROPLIST:(
             MPM:PROPLIST:(
                   IA:12,1,0,52,0,45),
                 ENDLIST
             TRANSACTION:37)
           ENDLIST,

        CMD:PROPLIST(
             MAILBOX:(PROPLIST:(
                        MPM:PROPLIST(
                              IA:12,3,0,52,0,45),
                            ENDLIST
                        NET:ARPA,
                        HOST:ISIB,
                        PORT:45,
                        USER:Cohen ),
                      ENDLIST
             OPERATION:DELIVER,
             TYPE-OF-SERVICE:REGULAR,
             TRACE:(PROPLIST:MPM:
                       (PROPLIST:
                          IA:12,1,0,52,0,45)
                        ENDLIST
                      DATE:1979-03-29-11:46-08:00,
                      ACTION:ORIGIN)),
                    ENDLIST
          ENDLIST
        DOC:<<document>>)
      ENDLIST
```

Example 2: Delivery and Acknowledgment

The following are four views of the message of example 1 during the successive transmission from the origination MPM, through a relay MPM, to the destination MPM, and the return of the acknowledgment, through a relay MPM, to the originating MPM.

```
+-----------------------------------------------------------------+
|                                A         B                       |
| sending --> originating --> relay --> destination --> receiving |
|   user          MPM          MPM          MPM            user   |
|                                                                 |
|                                D         C                       |
|             originating <-- relay <-- destination               |
|                 MPM          MPM          MPM                    |
+-----------------------------------------------------------------+
```

Transmission Path

Figure 6.

A. Between the originating MPM and the relay MPM.

```
PROPLIST:
  NAME:"ID",
    PROPLIST:
      NAME:"MPM",
        PROPLIST:
          NAME:"IA", NAME:"10,1,0,52,0,45"
        ENDLIST
      NAME:"TRANSACTION", INTEGER:37
    ENDLIST
  NAME:"CMD",
    PROPLIST:
      NAME:"MAILBOX",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", NAME:"10,3,0,52,0,45"
            ENDLIST
          NAME:"NET", NAME:"ARPA"
          NAME:"HOST", NAME:"ISIB"
          NAME:"PORT", NAME:"45"
          NAME:"USER", NAME:"Cohen"
        ENDLIST
      NAME:"OPERATION", NAME:"DELIVER"
      NAME:"TYPE-OF-SERVICE", NAME:"REGULAR"
      NAME:"TRACE",
        LIST:
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", NAME:"10,1,0,52,0,45"
              ENDLIST
            NAME:"DATE", NAME:"1979-03-29-11:47.5-08:00"
            NAME:"ACTION", NAME:"ORIGIN"
          ENDLIST
        ENDLIST
    ENDLIST
  NAME:"DOC", <<document>>
ENDLIST
```

B. Between the relay MPM and the destination MPM.

```
PROPLIST:
  NAME:"ID",
    PROPLIST:
      NAME:"MPM",
        PROPLIST:
          NAME:"IA", NAME:"10,1,0,52,0,45"
        ENDLIST
      NAME:"TRANSACTION", INTEGER:37
    ENDLIST
  NAME:"CMD",
    PROPLIST:
      NAME:"MAILBOX",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", NAME:"10,3,0,52,0,45"
            ENDLIST
          NAME:"NET", NAME:"ARPA"
          NAME:"HOST", NAME:"ISIB"
          NAME:"PORT", NAME:"45"
          NAME:"USER", NAME:"Cohen"
        ENDLIST
      NAME:"OPERATION", NAME:"DELIVER"
      NAME:"TYPE-OF-SERVICE", NAME:"REGULAR"
      NAME:"TRACE",
        LIST:
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", NAME:"10,1,0,52,0,45"
              ENDLIST
            NAME:"DATE", NAME:"1979-03-29-11:47.5-08:00"
            NAME:"ACTION", NAME:"ORIGIN"
          ENDLIST
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", NAME:"10,2,0,52,0,45"
              ENDLIST
            NAME:"DATE", NAME:"1979-03-29-11:48-08:00"
            NAME:"ACTION", NAME:"RELAY"
          ENDLIST
        ENDLIST
    ENDLIST
  NAME:"DOC", <<document>>
```

```
    ENDLIST

  C.  Between the destination MPM and the relay MPM.

    PROPLIST:
      NAME:"ID",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", NAME:"10,3,0,52,0,45"
            ENDLIST
          NAME:"TRANSACTION", INTEGER:1993
        ENDLIST
      NAME:"CMD",
        PROPLIST:
          NAME:"MAILBOX",
            PROPLIST:
              NAME:"MPM",
                PROPLIST:
                  NAME:"IA", INTEGER:"10,1,0,52,0,45"
                ENDLIST
              NAME:"NET", NAME:"ARPA"
              NAME:"HOST", NAME:"ISIE"
              NAME:"PORT", NAME:"45"
              NAME:"USER", NAME:"*MPM*"
            ENDLIST
          NAME:"OPERATION", NAME:"ACKNOWLEDGE"
          NAME:"REFERENCE",
            PROPLIST:
              NAME:"MPM",
                PROPLIST:
                  NAME:"IA", NAME:"10,1,0,52,0,45"
                ENDLIST
              NAME:"TRANSACTION", INTEGER:37
            ENDLIST
          NAME:"ADDRESS",
            PROPLIST:
              NAME:"MPM",
                PROPLIST:
                  NAME:"IA", INTEGER:"10,3,0,52,0,45"
                ENDLIST
              NAME:"USER", NAME:"Cohen"
            ENDLIST
          NAME:"TYPE-OF-SERVICE", NAME:"REGULAR"
          NAME:"ERROR-CLASS", INDEX:0
          NAME:"ERROR-STRING", NAME:"Ok"
          NAME:"TRAIL",
                LIST:
                  PROPLIST:
                    NAME:"MPM",
                      PROPLIST:
                        NAME:"IA", NAME:"10,1,0,52,0,45"
                      ENDLIST
                    NAME:"DATE", NAME:"1979-03-29-11:47.5-08:00"
                    NAME:"ACTION", NAME:"ORIGIN"
                  ENDLIST
                  PROPLIST:
                    NAME:"MPM",
                      PROPLIST:
                        NAME:"IA", NAME:"10,2,0,52,0,45"
                      ENDLIST
                    NAME:"DATE", NAME:"1979-03-29-11:48-08:00"
                    NAME:"ACTION", NAME:"RELAY"
                  ENDLIST
                  PROPLIST:
                    NAME:"MPM",
                      PROPLIST:
                        NAME:"IA", NAME:"10,3,0,52,0,45"
                      ENDLIST
                    NAME:"DATE", NAME:"1979-03-29-11:51.567-08:00"
                    NAME:"ACTION", NAME:"DESTINATION"
                  ENDLIST
                ENDLIST
              NAME:"TRACE",
                LIST:
                  PROPLIST:
                    NAME:"MPM",
                      PROPLIST:
                        NAME:"IA", NAME:"10,3,0,52,0,45"
                      ENDLIST
                    NAME:"DATE", NAME:"1979-03-29-11:52-08:00"
                    NAME:"ACTION", NAME:"ORIGIN"
                  ENDLIST
                ENDLIST
            ENDLIST
        ENDLIST
```

D. Between the relay MPM and the originating MPM.

```
PROPLIST:
  NAME:"ID",
    PROPLIST:
      NAME:"MPM",
        PROPLIST:
          NAME:"IA", NAME:"10,3,0,52,0,45"
        ENDLIST
      NAME:"TRANSACTION", INTEGER:1993
    ENDLIST
  NAME:"CMD",
    PROPLIST:
      NAME:"MAILBOX",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", INTEGER:"10,1,0,52,0,45"
            ENDLIST
          NAME:"NET", NAME:"ARPA"
          NAME:"HOST", NAME:"ISIE"
          NAME:"PORT", NAME:"45"
          NAME:"USER", NAME:"*MPM*"
        ENDLIST
      NAME:"OPERATION", NAME:"ACKNOWLEDGE"
      NAME:"REFERENCE",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", NAME:"10,1,0,52,0,45"
            ENDLIST
          NAME:"TRANSACTION", INTEGER:37
        ENDLIST
      NAME:"ADDRESS",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", INTEGER:"10,3,0,52,0,45"
            ENDLIST
          NAME:"USER", NAME:"Cohen"
        ENDLIST
      NAME:"TYPE-OF-SERVICE", NAME:"REGULAR"
      NAME:"ERROR-CLASS", INDEX:0
      NAME:"ERROR-STRING", NAME:"Ok"
      NAME:"TRAIL",
        LIST:
          PROPLIST:
                NAME:"MPM",
                  PROPLIST:
                    NAME:"IA", NAME:"10,1,0,52,0,45"
                  ENDLIST
                NAME:"DATE", NAME:"1979-03-29-11:47.5-08:00"
                NAME:"ACTION", NAME:"ORIGIN"
              ENDLIST
              PROPLIST:
                NAME:"MPM",
                  PROPLIST:
                    NAME:"IA", NAME:"10,2,0,52,0,45"
                  ENDLIST
                NAME:"DATE", NAME:"1979-03-29-11:48-08:00"
                NAME:"ACTION", NAME:"RELAY"
              ENDLIST
              PROPLIST:
                NAME:"MPM",
                  PROPLIST:
                    NAME:"IA", NAME:"10,3,0,52,0,45"
                  ENDLIST
                NAME:"DATE", NAME:"1979-03-29-11:51.567-08:00"
                NAME:"ACTION", NAME:"DESTINATION"
              ENDLIST
            ENDLIST
          NAME:"TRACE",
            LIST:
              PROPLIST:
                NAME:"MPM",
                  PROPLIST:
                    NAME:"IA", NAME:"10,3,0,52,0,45"
                  ENDLIST
                NAME:"DATE", NAME:"1979-03-29-11:52-08:00"
                NAME:"ACTION", NAME:"ORIGIN"
              ENDLIST
              PROPLIST:
                NAME:"MPM",
                  PROPLIST:
                    NAME:"IA", NAME:"10,2,0,52,0,45"
                  ENDLIST
                NAME:"DATE", NAME:"1979-03-29-11:52.345-08:00"
                NAME:"ACTION", NAME:"RELAY"
              ENDLIST
            ENDLIST
        ENDLIST
    ENDLIST
```

7. SPECIFICATION SUMMARY

7.1. Message Fields

All keywords used in this protocol are to be recognized independent of case.

```
action: NAME (one of)
  "ORIGIN" | "RELAY" | "FORWARD" | "DESTINATION"

address: PROPLIST (one of)

  NAME: "MPM", <mpm-identifier>
  NAME: "USER", <user>

    or

  NAME: "NET", <net>
  NAME: "HOST", <host>
  NAME: "PORT", <port>
  NAME: "USER", <user>

answer: BOOLEAN

city: NAME

command: PROPLIST
  NAME: "MAILBOX", <mailbox>
  NAME: "OPERATION", <operation>
  <<arguments>>
  NAME: "ERROR-CLASS", <error-class>    (only in replies)
  NAME: "ERROR-STRING", <error-string>  (only in replies)
  NAME: "TRACE", <trace>

country: NAME

document: <<document>>

error-class: INDEX

error-string: NAME

host: NAME
```

```
handling-stamp: PROPLIST
  NAME: "MPM", <mpm-identifier>
  NAME: "DATE", <date>
  NAME: "ACTION", <action>

identification: LIST
  NAME: "MPM", <mpm-identifier>
  NAME: "TRANSACTION", <transaction-number>

internet-address: NAME

mailbox:  PROPLIST (some of)
  NAME: "MPM", <mpm-identifier>
  NAME: "NET", <net>
  NAME: "HOST", <host>
  NAME: "PORT", <port>
  NAME: "USER", <user>
  NAME: "ORG", <organization>
  NAME: "CITY", <city>
  NAME: "STATE", <state>
  NAME: "COUNTRY", <country>
  NAME: "ZIP", <zip-code>
  NAME: "PHONE", <phone-number>
  <<other-items>>

message: PROPLIST
  NAME: "ID", <identification>
  NAME: "CMD", <command>
  NAME: "DOC", <document> (only in deliver)

mpm-identifier: PROPLIST (one of)

  NAME: "IA", <internet-address>

    or

  NAME: "X121", <x121-address>

net: NAME

operation: NAME (one of)
    "DELIVER" | "ACKNOWLEDGE
  | "PROBE"   | "RESPONSE
  | "CANCEL"  | "CANCELED"

organization: NAME

phone-number: NAME
```

```
port: NAME

state: NAME

trace: LIST
  <handling-stamp>
  ...

trail: LIST
  <handling-stamp>
  ...

transaction-number: INTEGER

type-of-service: NAME (one or more of)
  "REGULAR" | "FORWARD" | "GENDEL" | "PRIORITY"

user: NAME

x121-address: NAME

zip-code: NAME
```

7.2. Deliver Message

```
  PROPLIST:
    NAME:"ID",
      PROPLIST:
        NAME:"MPM",
          PROPLIST:
            NAME:"IA", NAME:<internet-address>
          ENDLIST
        NAME:"TRANSACTION", INTEGER:<transaction-number>
      ENDLIST
    NAME:"CMD",
      PROPLIST:
        NAME:"MAILBOX",
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"NET", NAME:<net>
            NAME:"HOST", NAME:<host>
            NAME:"PORT", NAME:<port>
            NAME:"USER", NAME:<user>
            NAME:"ORG", NAME:<organization>
            NAME:"CITY", NAME:<city>
            NAME:"STATE", NAME:<state>
            NAME:"COUNTRY", NAME:<country>
            NAME:"ZIP", NAME:<zip-code>
            NAME:"PHONE", NAME:<phone-number>
            <<other-items>>
          ENDLIST
        NAME:"OPERATION", NAME:"DELIVER"
        NAME:"TYPE-OF-SERVICE", NAME:<type-of-service>
        NAME:"TRACE",
          LIST:
            PROPLIST:
              NAME:"MPM",
                PROPLIST:
                  NAME:"IA", INTEGER:<internet-address>
                ENDLIST
              NAME:"DATE", NAME:<date>
              NAME:"ACTION", NAME:<action>
            ENDLIST
            ...
          ENDLIST
      ENDLIST
    NAME:"DOC", <<document>>
  ENDLIST
```

7.3. Acknowledge Message

```
PROPLIST:
  NAME:"ID",
    PROPLIST:
      NAME:"MPM",
        PROPLIST:
          NAME:"IA", NAME:<internet-address>
        ENDLIST
      NAME:"TRANSACTION", INTEGER:<transaction-number>
    ENDLIST
  NAME:"CMD",
    PROPLIST:
      NAME:"MAILBOX",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", INTEGER:<internet-address>
            ENDLIST
          NAME:"USER", NAME:"*MPM*"
          NAME:"NET", NAME:<net>
          NAME:"PORT", NAME:<port>
          NAME:"HOST", NAME:<host>
        ENDLIST
      NAME:"OPERATION", NAME:"ACKNOWLEDGE"
      NAME:"REFERENCE",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", NAME:<internet-address>
            ENDLIST
          NAME:"TRANSACTION", INTEGER:<transaction-number>
        ENDLIST
      NAME:"ADDRESS",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", INTEGER:<internet-address>
            ENDLIST
          NAME:"USER", NAME:<user>
        ENDLIST
      NAME:"TYPE-OF-SERVICE", NAME:<type-of-service>
      NAME:"ERROR-CLASS", INDEX:<error-class>
      NAME:"ERROR-STRING", NAME:<error-string>
      NAME:"TRAIL",
        LIST:
          PROPLIST:
            NAME:"MPM",
                PROPLIST:
                  NAME:"IA", INTEGER:<internet-address>
                ENDLIST
              NAME:"DATE", NAME:<date>
              NAME:"ACTION", NAME:<action>
            ENDLIST
            ...
          ENDLIST
        NAME:"TRACE",
          LIST:
            PROPLIST:
              NAME:"MPM",
                PROPLIST:
                  NAME:"IA", INTEGER:<internet-address>
                ENDLIST
              NAME:"DATE", NAME:<date>
              NAME:"ACTION", NAME:<action>
            ENDLIST
            ...
          ENDLIST
      ENDLIST
  ENDLIST
```

7.4. Probe Message

```
PROPLIST:
  NAME:"ID",
    PROPLIST:
      NAME:"MPM",
        PROPLIST:
          NAME:"IA", NAME:<internet-address>
        ENDLIST
      NAME:"TRANSACTION", INTEGER:<transaction-number>
    ENDLIST
  NAME:"CMD",
    PROPLIST:
      NAME:"MAILBOX",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", INTEGER:<internet-address>
            ENDLIST
          NAME:"NET", NAME:<net>
          NAME:"HOST", NAME:<host>
          NAME:"PORT", NAME:<port>
          NAME:"USER", NAME:<user>
          NAME:"ORG", NAME:<organization>
          NAME:"CITY", NAME:<city>
          NAME:"STATE", NAME:<state>
          NAME:"COUNTRY", NAME:<country>
          NAME:"ZIP", NAME:<zip-code>
          NAME:"PHONE", NAME:<phone-number>
          <<other-items>>
        ENDLIST
      NAME:"OPERATION", NAME:"PROBE"
      NAME:"TRACE",
        LIST:
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"DATE", NAME:<date>
            NAME:"ACTION", NAME:<action>
          ENDLIST
          ...
        ENDLIST
    ENDLIST
ENDLIST
```

7.5. Response Message

```
PROPLIST:
  NAME:"ID",
    PROPLIST:
      NAME:"MPM",
        PROPLIST:
          NAME:"IA", NAME:<internet-address>
        ENDLIST
      NAME:"TRANSACTION", INTEGER:<transaction-number>
    ENDLIST
  NAME:"CMD",
    PROPLIST:
      NAME:"MAILBOX",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", INTEGER:<internet-address>
            ENDLIST
          NAME:"NET", NAME:<net>
          NAME:"HOST", NAME:<host>
          NAME:"PORT", NAME:<port>
          NAME:"USER", NAME:"*MPM*"
        ENDLIST
      NAME:"OPERATION", NAME:"RESPONSE"
      NAME:"REFERENCE",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", NAME:<internet-address>
            ENDLIST
          NAME:"TRANSACTION", INTEGER:<transaction-number>
        ENDLIST
      NAME:"ADDRESS",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", INTEGER:<internet-address>
            ENDLIST
          NAME:"USER", NAME:<user>
        ENDLIST
      NAME:"ERROR-CLASS", INDEX:<error-class>
      NAME:"ERROR-STRING", NAME:<error-string>
      NAME:"TRAIL",
        LIST:
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"DATE", NAME:<date>
            NAME:"ACTION", NAME:<action>
          ENDLIST
          ...
        ENDLIST
      NAME:"TRACE",
        LIST:
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"DATE", NAME:<date>
            NAME:"ACTION", NAME:<action>
          ENDLIST
          ...
        ENDLIST
    ENDLIST
ENDLIST
```

7.6. Cancel Message

```
PROPLIST:
  NAME:"ID",
    PROPLIST:
      NAME:"MPM",
        PROPLIST:
          NAME:"IA", NAME:<internet-address>
        ENDLIST
      NAME:"TRANSACTION", INTEGER:<transaction-number>
    ENDLIST
  NAME:"CMD",
    PROPLIST:
      NAME:"MAILBOX",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", INTEGER:<internet-address>
            ENDLIST
          NAME:"NET", NAME:<net>
          NAME:"HOST", NAME:<host>
          NAME:"PORT", NAME:<port>
          NAME:"USER", NAME:<user>
          NAME:"ORG", NAME:<organization>
          NAME:"CITY", NAME:<city>
          NAME:"STATE", NAME:<state>
          NAME:"COUNTRY", NAME:<country>
          NAME:"ZIP", NAME:<zip-code>
          NAME:"PHONE", NAME:<phone-number>
          <<other-items>>
        ENDLIST
      NAME:"OPERATION", NAME:"CANCEL"
      NAME:"REFERENCE",
        PROPLIST:
          NAME:"MPM",
            PROPLIST:
              NAME:"IA", NAME:<internet-address>
            ENDLIST
          NAME:"TRANSACTION", INTEGER:<transaction-number>
        ENDLIST
      NAME:"TRACE",
        LIST:
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"DATE", NAME:<date>
```

```
              NAME:"ACTION", NAME:<action>
            ENDLIST
             ...
          ENDLIST
      ENDLIST
  ENDLIST
```

7.7. Canceled Message

```
  PROPLIST:
    NAME:"ID",
      PROPLIST:
        NAME:"MPM",
          PROPLIST:
            NAME:"IA", NAME:<internet-address>
          ENDLIST
        NAME:"TRANSACTION", INTEGER:<transaction-number>
      ENDLIST
    NAME:"CMD",
      PROPLIST:
        NAME:"MAILBOX",
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"NET", NAME:<net>
            NAME:"HOST", NAME:<host>
            NAME:"PORT", NAME:<port>
            NAME:"USER", NAME:"*MPM*"
          ENDLIST
        NAME:"OPERATION", NAME:"CANCELED"
        NAME:"REFERENCE",
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", NAME:<internet-address>
              ENDLIST
            NAME:"TRANSACTION", INTEGER:<transaction-number>
          ENDLIST
        NAME:"ADDRESS",
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"USER", NAME:<user>
          ENDLIST
        NAME:"ERROR-CLASS", INDEX:<error-class>
        NAME:"ERROR-STRING", NAME:<error-string>
        NAME:"TRAIL",
          LIST:
            PROPLIST:
              NAME:"MPM",
                PROPLIST:
```

```
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"DATE", NAME:<date>
            NAME:"ACTION", NAME:<action>
          ENDLIST
          ...
        ENDLIST
      NAME:"TRACE",
        LIST:
          PROPLIST:
            NAME:"MPM",
              PROPLIST:
                NAME:"IA", INTEGER:<internet-address>
              ENDLIST
            NAME:"DATE", NAME:<date>
            NAME:"ACTION", NAME:<action>
          ENDLIST
          ...
        ENDLIST
    ENDLIST
ENDLIST
```

7.8. Data Element Summary

CODE	NAME	STRUCTURE	LENGTH
0	NOP	CODE(1)	1
1	PAD	CODE(1),COUNT(3),DATA(C)	C+4
2	BOOLEAN	CODE(1),TRUE-FALSE(1)	2
3	INDEX	CODE(1),INDEX(2)	3
4	INTEGER	CODE(1),INTEGER(4)	5
5	EPI	CODE(1),COUNT(3),INTEGER(C)	C+4
6	BITSTR	CODE(1),COUNT(3),BITS(C/8)	C/8+4
7	NAME	CODE(1),COUNT(1),NAME(C)	C+2
8	TEXT	CODE(1),COUNT(3),TEXT(C)	C+4
9	LIST	CODE(1),COUNT(3),ITEMS(2),DATA(C-2)	C+4
10	PROPLIST	CODE(1),COUNT(3),PAIRS(1),DATA(C-1)	C+4
11	ENDLIST	CODE(1)	1
12	S-TAG	CODE(1),INDEX(2)	3
13	S-REF	CODE(1),INDEX(2)	3
14	ENCRYPT	CODE(1),COUNT(3),ALG-ID(1), KEY-ID(2),DATA(C-3)	C+4

The numbers in parentheses are the number of octets in the field.

REFERENCES

[1] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.

[2] Postel, J., "DOD Standard Internet Protocol," USC/Information Sciences Institute, IEN 128, NTIS number AD A079730, January 1980.

[3] Postel, J., "DOD Standard Transmission Control Protocol," USC/Information Sciences Institute, IEN 129, NTIS number AD A082609, January 1980.

[4] Postel, J., "Assigned Numbers," RFC 762, USC/Information Sciences Institute, January 1980.

[5] Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook," NIC 7104, for the Defense Communications Agency by the Network Information Center of SRI International, Menlo Park, California, Revised January 1978.

[6] Neigus, N., "File Transfer Protocol for the ARPA Network," RFC 542, NIC 17759, SRI International, August 1973.

[7] Bhushan, A., K. Progran, R. Tomlinson, and J. White, "Standardizing Network Mail Headers," RFC 561, NIC 18516, September 1973.

[8] Myer, T., and D. Henderson, "Message Transmission Protocol," RFC 680, NIC 32116, 30 April 1975.

[9] Crocker, D., J. Vittal, K. Progran, and D. Henderson, "Standard for the Format of ARPA Network Text Messages," RFC 733, NIC 41952, 21 November 1977.

[10] Barber, D., and J. Laws, "A Basic Mail Scheme for EIN," INWG 192, February 1979.

[11] Braaten, O., "Introduction [illegible] a Mail Protocol," Norwegian Computing Center, INWG 180, August 1978.

[12] Crocker, D., E. Szurkowski, and D. Farber, "An Internetwork Memo Distribution Capability - MMDF," Sixth Data Communications Symposium, ACM/IEEE, November 1979.

[13] Haverty, J., D. Henderson, and D. Oestreicher, "Proposed Specification of an Inter-site Message Protocol," 8 July 1975.

[14] Thomas, R., "Providing Mail Services for NSW Users," BBN NSW Working Note 24, Bolt Beranek and Newman, October 1978.

[15] White, J., "A Proposed Mail Protocol," RFC 524, NIC 17140, SRI International, 13 June 1973.

[16] White, J., "Description of a Multi-Host Journal," NIC 23144, SRI International, 30 May 1974.

[17] White, J., "Journal Subscription Service," NIC 23143, SRI International, 28 May 1974.

[18] Levin, R., and M. Schroeder, "Transport of Electronic Messages Through a Network," Teleinformatics 79, Boutmy & Danthine (eds.) North Holland Publishing Co., 1979.

[19] Earnest, L., and J. McCarthy, "DIALNET: A Computer Communications Study," Computer Science Department, Stanford University, August 1978.

[20] Crispin M., "DIALNET: A Telephone Network Data Communications Protocol," DECUS Proceedings, Fall 1979.

[21] Caulkins, D., "The Personal Computer Network (PCNET) Project: A Status Report," Dr. Dobbs Journal of Computer Calisthenics and Orthodontia, v.5, n.6, June 1980.

[22] Postel, J., "NSW Transaction Protocol (NSWTP)," USC/Information Sciences Institute, IEN 38, May 1978.

[23] Haverty, J., "MSDTP -- Message Services Data Transmission Protocol," RFC 713, NIC 34739, April 1976.

[24] Haverty, J., "Thoughts on Interactions in Distributed Services," RFC 722, NIC 36806, 16 September 1976.

[25] Postel, J., "A Structured Format for Transmission of Multi-Media Documents," RFC 767, USC/Information Sciences Institute, August 1980.

[26] ISO-2014, "Writing of calendar dates in all-numeric form," Recommendation 2014, International Organization for Standardization, 1975.

[27] ISO-3307, "Information Interchange -- Representations of time of the day," Recommendation 3307, International Organization for Standardization, 1975.

[28] ISO-4031, "Information Interchange -- Representation of local time differentials," Recommendation 4031, International Organization for Standardization, 1978.

[29] CCITT-X.121, "International Numbering Plan for Public Data Networks," Recommendation X.121, CCITT, Geneva, 1978.

[30] Postel, J., "NSW Data Representation (NSWB8)," USC/Information Sciences Institute, IEN 39, May 1978.

[31] Cohen, D., "On Holy Wars and a Plea for Peace," IEN 137, USC/Information Sciences Institute, 1 April 1980.

[32] Hofstadter, D., "Godel, Escher, Bach: An Eternal Golden Braid," Basic Books, New York, 1979..

[33] Harrenstien, K., "Field Addressing," ARPANET Message, SRI International, October 1977.

[34] Postel, J., "Out-of-Net Host Address for Mail," RFC 754, USC/Information Sciences Institute, April 1979.

[35] Shoch, J., "On Inter-Network Naming, Addressing, and Routing," IEEE Computer Society, COMPCON, Fall 1978.

[36] National Bureau of Standards, "Data Encryption Standard," Federal Information Processing Standards Publication 46, January 1977.

[37] Diffie, W., and M. Hellman, "New Directions in Cryptology," IEEE Transactions on Information Theory, IT-22, 6, November 1976.

[38] Rivest, R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" Communications of the ACM, Vol. 21, Number 2, February 1978.

[39] Merkle, R., and M. Hellman, "Hiding Information and Signatures in Trapdoor Knapsacks," IEEE Transactions of Information Theory, IT-24,5, September 1978.

Network Working Group
Request for Comments: 937

Obsoletes: RFC 918

M. Butler
J. Postel
D. Chase
J. Goldberger
J. K. Reynolds
ISI
February 1985

POST OFFICE PROTOCOL - VERSION 2

Status of this Memo

This RFC suggests a simple method for workstations to dynamically access mail from a mailbox server. This RFC specifies a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvement. This memo is a revision of RFC 918. Distribution of this memo is unlimited.

Introduction

The intent of the Post Office Protocol Version 2 (POP2) is to allow a user's workstation to access mail from a mailbox server. It is expected that mail will be posted from the workstation to the mailbox server via the Simple Mail Transfer Protocol (SMTP). For further information see RFC-821 [1] and RFC-822 [2].

This protocol assumes a reliable data stream such as provided by TCP or any similar protocol. When TCP is used, the POP2 server listens on port 109 [4].

System Model and Philosophy

While we view the workstation as an Internet host in the sense that it implements IP, we do not expect the workstation to contain the user's mailbox. We expect the mailbox to be on a server machine.

We believe it is important for the mailbox to be on an "always up" machine and that a workstation may be frequently powered down, or otherwise unavailable as an SMTP server.

POP2 is designed for an environment of workstations and servers on a low-delay, high-throughput, local networks (such as Ethernets). POP2 may be useful in other environments as well, but if the environment is substantially different, a different division of labor between the client and server may be appropriate, and a different protocol required.

Suppose the user's real name is John Smith, the user's machine is called FIDO, and that the mailbox server is called DOG-HOUSE. Then we expect the user's mail to be addressed to JSmith@DOG-HOUSE.ARPA (not JSmith@FIDO.ARPA).

That is, the destination of the mail is the mailbox on the server machine. The POP2 protocol and the workstation are merely a mechanism for viewing the messages in the mailbox.

The user is not tied to any particular workstation for accessing his mail. The workstation does not appear as any part of the mailbox address.

This is a very simple protocol. This is not a user interface. We expect that there is a program in the workstation that is friendly to the user. This protocol is not "user friendly". One basic rule of this protocol is "if anything goes wrong close the connection". Another basic rule is to have few options.

POP2 does not parse messages in any way. It does not analyze message headers (Date:, From:, To:, Cc:, or Subject:). POP2 simply transmits whole messages from a mailbox server to a client workstation.

The Protocol

The POP2 protocol is a sequence of commands and replies. The design draws from many previous protocols of the ARPA-Internet community.

The server must be listening for a connection. When a connection is opened the server sends a greeting message and waits for commands. When commands are received the server acts on them and responds with replies.

The client opens a connection, waits for the greeting, then sends the HELO command with the user name and password arguments to establish authorization to access mailboxes. The server returns the number of messages in the default mailbox.

The client may read the default mailbox associated with the user name or may select another mailbox by using the FOLD command. The server returns the number of messages in the mailbox selected.

The client begins a message reading transaction with a READ command. The read command may optionally indicate which message number to read, the default is the current message (incremented when a message is read and set to one when a new folder is selected). The server returns the number of characters in the message.

The client asks for the content of the message to be sent with the RETR command. The server sends the message data.

When all the data has been received the client sends an acknowledgment command. This is one of ACKS, ACKD, and NACK.

ACKS means "I've received the message successfully and please keep it in the mailbox".

ACKD means "I've received the message successfully and please delete it from the mailbox".

NACK means "I did not receive the message and please keep it in the mailbox".

In the case of ACKS or ACKD the server increments the current message indicator. In the case of NACK the current message indicator stays the same.

In all cases the server returns the number of characters in the (now) current message.

The client terminates the session with the QUIT command. The server returns an ok.

The Normal Scenario

```
          Client                     Server
          ------                     ------
                                Wait for Connection
    Open Connection  -->
                          <--   + POP2 Server Ready
                                Wait for Command
    HELO Fred Secret -->
                          <--   #13 messages for you
                                Wait for Command
    READ 13          -->
                          <--   =537 characters in that message
                                Wait for Command
    RETR             -->
                          <--   (send the message data)
                                Wait for Command
    ACKS             -->
                          <--   =0 no more messages
                                Wait for Command
    QUIT             -->
                          <--   + OK
    Close connection --> <--   Close connection
                                Wait for Connection (go back to start)
```

Conventions

Arguments

These arguments have system specific definitions.

user - A login account name.

password - The password for the login account.

mailbox - A mailbox name (also called a mail folder).

Default Mailboxes

TOPS-20

MAIL.TXT.1 - from login directory

UNIX

```
both
   /usr/spool/mail/user
and
   /usr/user/Mail/inbox/*
```

where "user" is the user value supplied in the HELO command.

End of Line

End of Line is Carriage Return (CR) followed by Line Feed (LF). This sequence is indicated by "CRLF" in this document. This end of line convention must be used for commands and replies.

Message Length

The reply to the READ command or an acknowledgment command (ACKS, ACKD, NACK) is the length (a character count) of the next message to be transmitted. This includes all the characters in the data transmitted. CRLF counts as two characters. A length of zero means the message does not exist or is empty. A request to transmit a message of zero length will result in the server closing the connection. The message is transmitted in the standard internet format described in RFC-822 [2] and NVT-ASCII. This may be different from the storage format and may make computing the message length from the stored message non-trivial.

Message Numbers

The reply to the HELO and FOLD commands is a count of the number of messages in a the selected mailbox. The READ command has a message number as an optional argument. These numbers are decimal, start at one, and computed with respect to the current mailbox. That is, the first message in a mailbox is message number 1.

Numbers

All numbers in this memo and protocol are decimal.

Quoting

In a few cases, there may be a need to have a special character in an argument (user, password, or mailbox) that is not allowed by the syntax For example, a space in a password. To allow for this, a quoting convention is defined. Unfortunately, such quoting conventions "use up" another otherwise uninteresting character. In this protocol the back slash "\" is used as the quote character. To include a space in an argument the two character sequence "back-slash, space" is transmitted. To include a back-slash in an argument the two character sequence "back-slash, back-slash" is transmitted. This quoting convention is used in the command arguments only, it is not used in the mail data transmitted in response to a RETR command.

Reply Strings

The first character is required to be as specified (i.e., "+", "-", "=", "#"). The optional strings that follow can be whatever the implementer thinks is appropriate.

Definitions of Commands and Replies

Summary of Commands and Replies

```
Commands                          Replies
--------                          -------
HELO user password                + OK
FOLD mailbox                      - Error
READ [n]                          #xxx
RETR                              =yyy
ACKS
ACKD
NACK
QUIT
```

Commands

HELO user password

The Hello command identifies the user to the server and carries the password authenticating this user. This information is used by the server to control access to the mailboxes. The Hello command is the "HELO" keyword, followed by the user argument, followed by the password argument, followed by CRLF.

Possible responses:

"#nnn"

where nnn is the number of messages in the default mailbox,"

"- error report" and Close the connection.

FOLD mailbox

The Folder command selects another mailbox or mail folder. The server must check that the user is permitted read access to this mailbox. If the mailbox is empty or does not exist, the number of messages reported is zero. The Folder command is the "FOLD" keyword, followed by the mailbox argument, followed by CRLF.

Possible responses:

"#nnn"

where nnn is the number of messages in this mailbox.

READ [nnn]

The Read command begins a message reading transaction. If the Read command is given without an argument the current message is implied (the current message indicator is incremented by the ACKS or ACKD commands). If an argument is used with the Read command it is the message number to be read, and this command sets the current message indicator to that value. The server returns the count of characters in the message to be transmitted. If there is no message to be read, the count of zero is returned. If the message was previously deleted with the ACKD command, the count of zero is returned. The Read command is followed by the RETR command, the READ command, the FOLD command, or the QUIT command. Do not attempt to RETR a message of zero characters. The Read command is the "READ" keyword, optionally followed by the message number argument, followed by CRLF.

Possible responses:

"=ccc"

where ccc is the number of characters in this message.

RETR

The Retrieve command confirms that the client is ready to receive the mail data. It must be followed by an acknowledgment command. The server will close the connection if asked to transmit a message of zero characters (i.e., transmit a non-existent message). The message is transmitted according to the Internet mail format standard RFC-822 [2] in NVT-ASCII. The Retrieve command is the "RETR" keyword, followed by CRLF.

Possible responses:

the message data

Close the connection

ACKS

The Acknowledge and Save command confirms that the client has received and accepted the message. The ACKS command ends the message reading transaction. The message is kept in the mailbox. The current message indicator is incremented. The server returns the count of characters in the now current message to be transmitted. If there is no message to be read or the message is marked deleted, the count of zero is returned. The Acknowledge and Save command is the "ACKS" keyword, followed by CRLF.

Possible responses:

"=ccc"

where ccc is the number of characters in the next message.

ACKD

The Acknowledge and Delete command confirms that the client has received and accepted the message. The ACKD command ends the message reading transaction. If the user is authorized to have write access to the mailbox, the message is deleted from the mailbox. Actually, the message is only marked for deletion. The actual change is made when the mailbox is released at the end of the session or when the client selects another mailbox with the FOLD command. The messages are not renumbered until the mailbox is released. If the user does not have write access to the mailbox no change is made to the mailbox. The response is the same whether or not the message was actually deleted. The current message indicator is incremented. The server returns the count of characters in the now current message to be transmitted. If there is no message to be read or the message is marked deleted, the count of zero is returned. The Acknowledge and Delete command is the "ACKD" keyword, followed by CRLF.

Possible responses:

"=ccc"

where ccc is the number of characters in the next message.

NACK

The Negative Acknowledge command reports that the client did not receive the message. The NACK command ends the message reading transaction. The message is kept in the mailbox. The current message indicator remains the same. The server returns the count of characters in the current message. Since the count to be returned is for the message just transmitted it the message must exist and not be marked deleted, and the count must be positive (non-zero). The Negative Acknowledge command is the "NACK" keyword, followed by CRLF.

Possible responses:

"=ccc"

where ccc is the number of characters in this message.

QUIT

The Quit command indicates the client is done with the session. The server sends an OK response and then closes the connection. The Quit command is the "QUIT" keyword, followed by CRLF.

Possible responses:

"+ OK" and Close the connection

Replies

Greeting

The greeting is sent by the server as soon as the connection is established. The greeting is a plus sign, followed by the protocol name ("POP2"), followed by the server host name, optionally followed by text, and ending with a CRLF.

+

The success or plus sign response indicates successful completion of the operation specified in the command. The success response is a plus sign, optionally followed by text, and ending with a CRLF.

-

The failure or minus sign response indicates the failure of the operation specified in the command. The failure response is a minus sign, optionally followed by text, and ending with a CRLF.

=

The length or equal sign response tells the length in characters of the message referenced by the command. The length response is a equal sign, followed by a number, optionally followed by text, and ending with a CRLF.

#

The count or number sign response tells the number of messages in a folder or mailbox referenced by the command. The count response is a number sign, followed by a number, optionally followed by text, and ending with a CRLF.

Timeouts

In any protocol of this type there have to be timeouts. Neither side wants to get stuck waiting forever for the other side (particularly is the other side has gone crazy or crashed).

The client expects a reply to a command fairly quickly and so should have a short timeout for this. This timeout is called T1.

For some servers, it may take some processing to compute the number of messages in a mailbox, or the length of a message, or to reformat a stored message for transmission, so this time out has to allow for such processing time. Also care must be taken not to timeout waiting for the completion of a RETR reply while a long message is in fact being transfered.

The server expects the session to progress with some but not excessive delay between commands and so should have a long timeout waiting for the next command. This time out is T2.

One model of use of this protocol is that any number of different types of clients can be built with different ways of interacting with the human user and the server, but still expecting the client to open the connection to the server, present a sequence of commands, and close the connection, without waiting for intervention by the human user. With such client implementations, it is reasonable for the server to have a fairly small value for timeout T2.

On the other hand, one could easily have the client be very human user directed with the user making decisions between commands. This would cause arbitrary delays between client commands to the server, and require the value of timeout T2 to be quite large.

Implementation Discussion

Comments on a Server on TOPS-20

On TOPS-20, a mailbox is a single file. New messages are appended to the file. There is a separator line between messages.

The tricky part of implementing a POP2 server on TOPS-20 is to provide for deleting messages. This only has to be done for the mailboxes (files) for which the user has write access. The problem is to avoid both (1) preventing other users from accessing or updating the mailbox for long periods, and (2) accidentally deleting a message the user has not seen.

One suggestion is as follows:

When a mailbox is first selected, if the user has write access, rename the mailbox file to some temporary name. Thus new messages will be placed in a new instance of the mailbox file. Conduct all POP2 operation on the temporary mailbox file (including deleting messages). When the POP2 session is over or another mailbox is selected, prepend any messages left undeleted in the temporary file to the new instance of the mailbox file.

Sizes

The maximum length of a command line is 512 characters (including the command word and the CRLF).

The maximum length of a reply line is 512 characters (including the success indicator (+, -, =, #) and the CRLF).

The maximum length of a text line is 1000 characters (including CRLF).

ISI has developed a POP2 server for TOPS-20 and for Berkeley 4.2 Unix, and a POP2 client for an IBM-PC and for Berkeley 4.2 Unix.

Extensions Not Supported

POP2 does not examine the internal data of messages. In particular, the server does not parse message headers.

The server doesn't have any state information (i.e., it doesn't know from one session to the next what has happened). For example, the server doesn't know which messages were received since the last time the user used POP2, so it can't send just the "new" messages.

Examples

Example 1:

```
             Client                    Server
             ------                    ------
                                  Wait for connection
        Open connection  -->
                             <--  + POP2 USC-ISIF.ARPA Server
        HELO POSTEL SECRET -->
                             <--  #2 messages in your mailbox
        READ               -->
                             <--  =537 characters in message 1
        RETR               -->
                             <--  [data of message 1]
        ACKD               -->
                             <--  =234 characters in message 2
        RETR               -->
                             <--  [data of message 2]
        ACKD               -->
                             <--  =0 no more messages
        QUIT               -->
                             <--  + OK, bye, bye
        Close connection --> <--  Close connection
                                  Go back to start
```

```
   Example 2:

                Client                      Server
                ------                      ------
                                        Wait for connection
      Open connection  -->
                            <--  + POP2 ISI-VAXA.ARPA server here
      HELO smith secret -->
                            <--  #35 messages
      FOLD /usr/spool/mail/smith -->
                            <--  #27 messages
      READ  27         -->
                            <--  =10123 characters in that message
      RETR             -->
                            <--  [data of message 27]
      ACKS             -->
                            <--  =0 no more messages
      QUIT             -->
                            <--  + bye, call again sometime.
      Close connection --> <--  Close connection
                                 Go back to start

   Example 3:

                Client                      Server
                ------                      ------
                                        Wait for connection
      Open connection  -->
                            <--  + POP2 ISI-VAXA.ARPA server here
      HELO Jones secret -->
                            <--  #0 messages
      READ             -->
                            <--  Close connection
      Close connection -->
                                 Go back to start
```

```
Formal Syntax

   <digit>    = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

   <letter>   = A | B | C | ... | Z
                a | b | c | ... | z

   <punct>    = ! | " | # | $ | % | & | ' | ( | ) | * |
                + | , | - | / | : | < | = | > | ? | @ |
                [ | ] | ^ | _ | ` | { | | | } | ~

   <quote>    = \

   <any>      = any one of the 128 ASCII codes

   <CR>       = carriage return, code 10

   <LF>       = line feed, code 13

   <SP>       = space, code 32

   <CRLF>     = <CR> <LF>

   <print>    = <letter> | <digit> | <punct> | <quote> <any>

   <char>     = <print> | <SP>

   <word>     = <print> | <print> <word>

   <string>   = <char> | <char> <string>

   <ld>       = <letter> | <digit>

   <ldh>      = <letter> | <digit> | -

   <ldhs>     = <ldh> | <ldh> <ldhs>

   <name>     = <letter> [ [ <ldhs> ] <ld> ]

   <host>     =  <name> | <name> . <host>

   <user>     = <word>

   <password> = <word>

   <mailbox>  = <string>

   <number>   = <digit> | <digit> <number>
```

```
    <helo>      = HELO <SP> <user> <SP> <password> <CRLF>

    <fold>      = FOLD <SP> <mailbox> <CRLF>

    <read>      = READ [<SP> <number>] <CRLF>

    <retr>      = RETR <CRLF>

    <acks>      = ACKS <CRLF>

    <ackd>      = ACKD <CRLF>

    <nack>      = NACK <CRLF>

    <quit>      = QUIT <CRLF>

    <ok>        = + [<SP> <string>] <CRLF>

    <err>       = - [<SP> <string>] <CRLF>

    <count>     = # <number> [<SP> <string>] <CRLF>

    <greet>     = + <SP> POP2 <SP> <host> [<SP> <string>] <CRLF>

    <length>    = = <number> [<SP> <string>] <CRLF>

    <command>   = <helo> | <fold> | <read> | <retr> |
                  <acks> | <ackd> | <nack> | <quit>

    <reply>     = <ok> | <err> | <count> | <length> | <greet>
```

Client State Diagram

```
                          |                    ^  + BYE
                          |  Open              |  -----
                          |           Greet    |  Close
                          V           -----    |
                      +-------+       QUIT    +-------+
                      | CALL  |-------------->| EXIT  |
                      +-------+               +-------+
                          |                       ^
                          |  Greet                |
                          |  -----                |
                          |  HELO                 |
              +---->+     |                       |
        #NNN  ^     |     |        #NNN           |
        ----  |     V     V        ----           |
        FOLD  |    +-------+       QUIT           |
              +<---| NMBR  |--------------------->+
                   +-------+                      ^
                    ^     |                       |
                    |     |  #NNN                 |
                    |     |  ----                 |
              =CCC  |     |  READ                 |
              ----  |     |                       |
              FOLD  |     |        =CCC           |
                    |     V        ----           |
        =CCC  +--->+-------+       QUIT           |
        ----  ^    | SIZE  |--------------------->+
        READ  +<---+-------+
                    ^     |
                    |     |  =CCC
              data  |     |  ----
              ----  |     |  RETR
              ack   |     |
                    |     V
                   +-------+
                   | XFER  |
                   +-------+
```

Server State Diagram

```
                     +<----------------------+  Close
                     |                       |  -----
              Listen |                       |  Close
                     V                       |
                +--------+              +--------+
                |  LSTN  |              |  DONE  |
                +--------+              +--------+
                     |                       ^
                     |  Open                 |
                     |  -----                |
                     |  Greet                |
                     |                       |
                     |           QUIT        |
                     V           -----       |
                +--------+       + BYE       |
                |  AUTH  |------------------>+
                +--------+                   ^
                     |                       |
                     |  HELO                 |
                     |  ----                 |
                     |  #NNN                 |
                     |                       |
                     |           QUIT        |
                     V           -----       |
     FOLD   +--->+--------+      + BYE       |
     ----   ^    |  MBOX  |----------------->+
     #NNN   +<---+--------+                  ^
                  ^     |                    |
                  |     |  READ              |
            FOLD  |     |  ----              |
            ----  |     |  =CCC              |
            #NNN  |     |        QUIT        |
                  |     V        -----       |
     READ   +--->+--------+      + BYE       |
     ----   ^    |  ITEM  |----------------->+
     =CCC   +<---+--------+
                  ^     |
                  |     |  RETR
            ack   |     |  ----
            ----  |     |  data
            =CCC  |     |
                  |     V
                 +--------+
                 |  NEXT  |
                 +--------+
```

Combined Flow Diagram

```
+-----+
|CALL |<--------------------------------------------------------------+
|LSTN |                                                               ^
+-----+                                                               |
 | Greet                                                              |
 |                                                                    |
 |  +-------------------------------------------------------->+       |
 |  ^ QUIT                                                    |       |
 V  |                                                         V       |
+-----+       +-----+                                       +-----+   |
|CALL | HELO  |NMBR |                                       |EXIT |   |
|AUTH |------>|AUTH |                                       |AUTH |   |
+-----+       +-----+                                       +-----+   |
                |  #NNN                                   + Bye  |    |
                |                                                |    |
                |  +-------------------------------------->+     |    |
                |  ^ QUIT                                  |     |    |
                V  |                                       V     |    |
         +--->+-----+        +-----+                     +-----+ |    |
    FOLD ^    |NMBR | READ   |SIZE |                     |EXIT | |    |
    ---- |    |MBOX |------->|MBOX |                     |MBOX | |    |
    #NNN +<---+-----+        +-----+                     +-----+ |    |
                   ^           | =CCC                  + Bye  |  |    |
                   |           |                              |  |    |
             FOLD  +<--------+ | +---------------------->+     |  |    |
             ----            ^ | ^ QUIT                  |     |  |    |
             #NNN            | V |                       V     |  |    |
                       +--->+------+        +-----+   +-----+ |  |    |
                 READ  ^    |SIZE  | RETR   |XFER |   |EXIT | |  |    |
                 ----  |    | ITEM |------->|ITEM |   |ITEM | |  |    |
                 =CCC  +<---+------+        +-----+   +-----+ |  |    |
                               ^              | data       |  |  |    |
                               |              |            |  |  |    |
                          =CCC |              V     + Bye  |  |  |    |
                             +-----+        +-----+        |  |  |    |
                             |SIZE |  Ack   |XFER |        |  |  |    |
                             |NEXT |<-------|NEXT |        |  |  |    |
                             +-----+        +-----+        |  |  |    |
                                                           |  |  |    |
                                                           |  |  |    |
                                                           V  V  V    |
                                                         +--------+   |
                                                         |  EXIT  |-->+
                                                         |  DONE  |
                                                         +--------+
```

Client Decision Table

	STATE				
INPUT	CALL	NMBR	SIZE	XFER	EXIT
Greet	2	1	1	1	6
#NNN	1	3	1	1	6
=CCC	1	1	4	1	6
data	1	1	1	5	6
+ Bye	1	1	1	1	6
Close	1	1	1	1	6
other	1	1	1	1	6
Timeout	1	1	1	1	6

Actions:

```
1.  This is garbage.  Send "QUIT", and go to EXIT state.

2.  (a) If the greeting is right then send "HELO"
        and go to NMBR state,
    (b) Else send "QUIT" and go to EXIT state.

3.  (a) If user wants this folder and NNN > 0
        then send "READ" and go to SIZE state,
    (b) If user wants a this folder and NNN = 0
        then send "QUIT" and go to EXIT state,
    (c) If user wants a different folder
        then send "FOLD" and go to NMBR state.

4.  (a) If user wants this message and CCC > 0
        then send "RETR" and go to XFER state,
    (b) If user wants a this message and CCC = 0
        then send "QUIT" and go to EXIT state,
    (c) If user wants a different message
        then send "READ" and go to SIZE state.

5.  (a) If user wants this message kept
        then send "ACKS" and go to SIZE state,
    (b) If user wants a this message deleted
        then send "ACKD" and go to SIZE state,
    (c) If user wants a this message again
        then send "NACK" and go to SIZE state.

6.  Close the connection.
```

Server Decision Table

	STATE					
INPUT	LSTN	AUTH	MBOX	ITEM	NEXT	DONE
Open	2	1	1	1	1	1
HELO	1	3	1	1	1	1
FOLD	1	1	5	5	1	1
READ	1	1	6	6	1	1
RETR	1	1	1	7	1	1
ACKS	1	1	1	1	8	1
ACKD	1	1	1	1	8	1
NACK	1	1	1	1	8	1
QUIT	1	4	4	4	1	1
Close	1	1	1	1	1	9
other	1	1	1	1	1	1
Timeout		1	1	1	1	1

Actions:

1. This is garbage. Send "- error", and Close the connection.
2. Send the greeting. Go to AUTH state.
3. (a) If authorized user then send "#NNN" and go tp MBOX state,
   (b) Else send "- error" and Close the connection.
4. Send "+ Bye" and go to DONE state.
5. Send "+NNN" and go to MBOX state.
6. Send "=CCC" and go to ITEM state.
7. If message exists then send the data and got to NEXT state,
   Else Close the connection.
8. Do what ACKS/ACKD/NACK require and go to ITEM state.
9. Close the connection.

Acknowledgment

We would like to acknowledge the helpful comments that we received on the first version of POP described in RFC 918, and the draft of POP2 distributed to interested parties.

References

[1] Postel, J., "Simple Mail Transfer Protocol", RFC 821, USC/Information Sciences Institute, August 1982.

[2] Crocker, D., "Standard for the Format of ARPA-Internet Text Messages", RFC 822, University of Delaware, August 1982.

[3] Reynolds, J.K., "Post Office Protocol", RFC 918, USC/Information Sciences Institute, October 1984.

[4] Reynolds, J.K., and J. Postel, "Assigned Numbers", RFC 923, USC/Information Sciences Institute, October 1984.