ARPANET COMPLETION REPORT

DRAFT

September 9, 1977

Table of Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

CHAPTER I: EXECUTIVE SUMMARY

<this chapter has not been written yet>

CHAPTER II: FORMAL COMPLETION REPORT

Preface

In June 1968 an ARPA program plan entitled "Resource Sharing Computer Networks" was formally prepared and approved. The research carried out pursuant to this plan has since become known and internationally famous as the ARPANET. This ARPA program has created no less than a revolution in computer technology and has been one of the most successful projects ever undertaken by ARPA. The program has initiated a farreaching effect on the Defense Department's use of computers as well as on the use of computers by the entire public and private sectors, both in the United States and around the world. Just as the telephone, the telegraph, and the printing press had farreaching effects on human intercommunication, the widespread utilization of computer networks which has been catalyzed by the ARPANET project represents a similarly farreaching change in the use of computers by mankind. The full impact of the technical changes set in motion by this project may not be understood for many years.

1. PROGRAM OBJECTIVE AND TECHNICAL NEED

1.1 Defense Program Addressed

The June 1968 ARPA program plan states the objective of the program in the following way:

> "The objective of this program is twofold: (1) To develop techniques and obtain experience on interconnecting computers in such a way that a very broad class of interactions are possible, and (2) To improve and increase computer research productivity through resource sharing. By establishing a network tying IPT's research centers together, both goals are achieved. In fact, the most efficient way to develop the techniques needed for an effective network is by involving the research talent at these centers in prototype activity.
>
> Just as timeshared computer systems have permitted groups of hundreds of individual users to share hardware and software resources with one another, networks connecting dozens of such systems will permit

resource sharing between thousands of users. Each system, by virtue of being timeshared, can offer any of its services to another computer system on demand. The most important criterion for the type of network interconnection desired is that any user or program on any of the networked computers can utilize any program or subsystem available on any other computer without having to modify the remote program."

This objective was an entirely new and different approach to an extremely serious problem which existed throughout both the Defense Department and the society at large. The many thousands of computer centers in the Defense Department and the many other thousands of computer centers in the private and public sectors operate almost completely autonomously. Each computer center is forced to recreate all the software and data files that it wishes to utilize. In many cases this involves the complete reprogramming of software or reformatting of data files. This duplication and redundant effort is extremely costly and time consuming. The June 1968 ARPA program plan estimated that "such duplicative efforts more than double the national costs of creating and maintaining the software". There had been other completely different attempts to address this problem, such as attempts at language standards for computers, attempts at standardizing the types of hardware, and attempts at automatic translation between computer languages. Although each such approach had some value and utility, the problems of trying to share computer software resources or files was truly enormous.

In addition to the general problem shared with the rest of the scientific community, the Defense Department also faces certain special problems having to do with training; quoting the program plan;

> "Military personnel trained to use one manufacturer's equipment must often be trained again to use a different station to another. Machines procured from different manufacturers require as many different user training programs as there are machines thus inhibiting positive transfer of training that could accumulate through the rotation of military personnel. Those data files and programs which have common utility to many military organizations and installations must be stored, created and maintained separately at each

different machine. Military systems interconnected in a distributed interactive network obviate such constraints."

Another objective noted in the program plan was to permit the linking of specialized computers to the many general purpose computer centers. The program plan noted this objective as follows:

"With recent improvements in the hardware area, it will become most cost effective to design and construct computers particularly efficient at specialized tasks (e.g., compiling, list processing and information retrieval). Making such machines available to all the computer research establishments would significantly increase the capability at these other centers."

This program was addressed at no less than changing the use of computers by the entire Defense Department. It was clearly intended that the use of such a computer network would permit resource sharing within and across the military services and throughout the Defense research community.

1.2 State of the Art at Program Inception

By the date of the program plan in the late 1960s most of the specific technologies required for a computer network had individually been achieved in some form. For example, there had been many connections of phone lines to computers (e.g., the SAGE system, Air Line reservations systems, and Time sharing systems). However, there had been only a very small number of attempts to connect computers together for the purpose of experimenting with the sharing of resources.

- o In the early 1960s an attempt was made to link computers together at the Western Data Processing Center at UCLA for the purpose of enabling similar computers to perform load sharing. A similar experiment was also performed at Bell Laboratories and achieved reasonable success for several years.

- o A number of networks were constructed for the primary purpose of message handling, including a Westinghouse inventory control system and several air line reservations networks. The SITA network for air line reservations was surprisingly advanced in concept in the mid 1960s, but details about SITA were generally not known in the U.S. computing community. In any case, the techniques used for such message systems were special purpose in nature and were not readily transferable into the general area of inter-computer communications.

o A direct progenitor of the ARPANET was an effort made in the mid 1960s to achieve a coupling between academic computing expertise and the operation of the SDC Q32 computer. This effort led to a phone line connection between the Q32 at SDC and the TX2 at Lincoln Laboratory, and demonstrated the relative ease of modifying time sharing systems to permit network interactions.

Aside from the technical problems of interconnecting computers with communications circuits, the notion of computer networks had been considered in a number of places from a theoretical point of view. Of particular note was work done by Paul Barran and others at the Rand Corporation in a study "On Distributed Communications" in the early 1960's. Also of note was work done by Donald Davies and others at the National Physical Laboratory in England in the mid 1960's.

In sum, at the time of the ARPA program plan in 1968 many of the requisite ideas had been considered and many of the requisite technical bits and pieces had been attempted in some form, but no significant attempt had ever been made to put them together into a resource sharing computer network.

1.3 Specific Technological Problem Addressed

The technological problems of building the ARPANET can be considered at many different levels of detail. At the top level, there were really two problems:

1. To construct a "subnetwork" consisting of telephone circuits and switching nodes whose reliability, delay characteristics, capacity, and cost would facilitate resource sharing between the computers on the network.

2. To understand, design, and implement the protocols and procedures within the operating systems of each connected computer, in order to allow the use of the new subnetwork by those computers in sharing resources.

Within these two major technological problems, there were, of course, a large number of sub-problems, including the engineering of the phone circuit connections, the topology of the network, the selection of switching node equipment, the design of line disciplines to work through phone line errors, the routing problem, and many others.

1.4 Expected Payoff/Time Frame/Costs

The goals for the ARPANET project were very broad and envisaged a significant eventual impact on the use of computers in both the public and private sectors. However in addition to these long range goals, ARPA visualized some quite specific initial payoff in the form of improved productivity of the ARPA research program itself, and a resulting cost/performance benefit to the services from ARPA research. Quoting from the 1968 program plan:

> "Seventeen computer research centers throughout the country are supported in whole or in part by Information Processing Techniques. The installation of an effective network tying these location together should substantially reduce duplication and improve the transfer of scientific results, as well as develop the network techniques needed by the military. The

research output of these projects is important to all three Services and it is expected that this output can be substantially increased for the same dollar cost if a portion of the funds are utilized for the network."

In addition, initial payoff was anticipated in the form of technology transfer from the ARPANET project in three ways:

o By dissemination of new scientific knowledge through conferences and the appropriate literature.

o By transfer of management of the ARPANET to a common carrier, and the resulting availability of ARPANET services to other groups (such as Office of Education Regional Laboratories, NSF-supported universities, and various user groups supported by the NIH).

o By adoption of the network technology by specific military groups (such as the National Military Command System Support Center and other military centers affiliated with it; e.g., CINCPAC, CINCEUR, and MACV).

The expected events and the schedules for those events was stated in the June 1968 ARPA Program Plan:

a. July 1968 -- Award IMP contract

b. March 1969 -- Demonstrate initial net operation with four nodes

c. April 1969 -- Approve design and extend contract to include installation of 19 IMPs

d. December 1969 -- Complete network operational

e. 1970 -- Add communication lines as necessary

f. 1971 - 1972 -- Arrange with a common carrier the transfer of the communications system

The costs anticipated by the Program Plan were as follows:

Year	Costs		
	Communication Line	IMP Contractor	Total
FY 68	0	563K	563K
FY 69	25K	1000K	1025K
FY 70	600K	200K	800K
FY 71	900K	100K	1000K

This cost estimate did not include the costs required for the various resource sharing experiments at the host sites in the ARPA community. The Program Plan avoided this issue, perhaps by necessity, as follows:

"The majority of this second class of costs will be bourne by each of the computer research contracts now extant. They will vary across a wide range of extremes bounded by, for example, a single researcher's small experimental program and a group of researcher's concern with studies of on-line documentation."

2. PROGRAM DESCRIPTION AND EVOLUTION

2.1 Program Structure

With the signing of the program plan in June 1968, ARPA began work in earnest on three parallel paths of effort: (1) to obtain the network circuits; (2) to select the system contractor for the switching nodes and the overall design; and (3) to initiate efforts within the ARPA research community for resource sharing experiments and specialized network support.

The ARPA IPT office had not had great experience in acquiring communications circuits and was rather pleasantly surprised to discover that another DoD organization, DECCO, was able to handle all the contractual details with the common carriers for circuit leases. Most of the required 50 kilobit circuits used in the ARPANET were leased through DECCO from AT&T, but a small number of circuits were leased from other carriers such as General Telephone. In addition, ARPA was fortunate in being able to arrange a rather high-level contact in AT&T (long lines), which greatly facilitated the interactions between the network system contractor, ARPA, and AT&T. The selection of network node locations and the internode connections (and, therefore, the location of circuit terminations) was a specialized topology problem and represented a difficult theoretical problem in its own right. To help solve this particular problem, ARPA contracted with the Network Analysis Corporation (NAC). NAC had developed certain networking analysis tools and via this ARPA support, such tools were refined; NAC's advice on topology was sought through the various stages of ARPANET growth.

In a procedure relatively unusual for the IPT office in ARPA, a competitive procurement was planned for the selection of the contractor to design the switching nodes and act as general systems contractor. An RFP was prepared and issued on ( ), and after an evaluation of approximately 15 bidders, Bolt Beranek and Newman Inc. (BBN) was selected as the systems contractor for the design of the subnetwork and the switching nodes. BBN subcontracted with Honeywell for the switching node hardware itself. Over the life of the ARPANET program from January 1969 until the transfer to DCA in July of 1975, BBN served as the systems contractor for the design, implementation, operation and maintenance of the subnetwork, and in 1977, BBN is still serving that role under the aegis of the Defense Communications Agency (DCA).

In a somewhat less structured way, the research groups receiving ARPA IPTO support were then encouraged to begin considering the design and implementation of protocols and procedures and, in turn, computer program modifications, in the various host computers in order to use the subnetwork. Several specific responsibilities were arranged: UCLA was specifically asked to take on the task of a "Network Measurement Center" with the objective of studying the performance of the network as it was built, grown, and modified; SRI was specifically asked to take on the task of a "Network Information Center" with the objective of collecting information about the network, about host resources, and at the same time generating computer based tools for storing and accessing that collected information. Beyond these two specific contracts, some rather ad hoc mechanisms were pursued to reach agreement between the various research contractors about the appropriate "host protocols" for intercommunicating over the subnetwork. The "Network Working Group" of interested individuals from the various host sites was rather informally encouraged by ARPA. After a time, this Network Working Group became the forum for, and eventually a semi-official approval authority for, the discussion of and issuance of host protocols to be implemented by the various research contractors. Since this part of the program was much less structured, progress was rather slow for a time, but with some ARPA IPTO pressure on the research contractors, this mechanism eventually was surprisingly successful in establishing effective host protocols.

2.2 Major Technical Problems and Approaches

In a program of this duration and complexity, it is not difficult to identify many dozens of important technical problems and approaches to those problems. We here list a few of the problems which were most technically challenging in the early few years of the ARPANET program. A few additional major technical problems will be listed in the next section on "Major Changes and Objectives".

- TOPOLOGY - For any network of this type with even a dozen nodes, an obvious, early recognized, and quite formidable problem is topological optimization. Assuming that the node locations are known, the number of ways of arranging M links among N nodes is very large; the links are usually available in discreet sizes (bandwidth); and the complement cost structures, time delay functions, and reliabilty functions are all typically non-linear. The design may be subject to many constraints, including maximum or average time delay, average or peak throughput requirements, and reliability requirements. The usual goal of the optimization is to provide a network design that meets all constraints at the lowest cost. The approach to this problem was to design an elaborate computer program to assist in the optimization. It is not possible to use an exhaustive approach, and instead the approach used was to generate a "starting network" and then to perform local transformations to the topology in order to reach a locally optimum network. If this procedure is repeated with many starting networks and if the resulting locally optimum networks are evaluated, it is possible to find a feasible solution with costs that are close to optimal.

o LINE ERRORS - A critical necessity for a resource sharing computer network was to provide reliable communication and one component of such reliability was an ability to work through the expected phone line errors on the 50 kilobit circuits. The approach taken was to design special checksumming hardware at the transmitting and receiving end of each 50 kilobit circuit in the network. As part of the switching node transmission procedure, a powerful 24 bit cyclic checksum is appended to every packet of information to be transmitted. Then, at the receiving node, the checksum is recomputed in hardware and compared with the transmitted checksum. If there is no error, an acknowledgment (with its own checksum) is sent back to the transmitting node. If there is an error, no acknowledgment is sent and the packet will be retransmitted.

o HOST INTERFACE - An important issue was the proper design of an interface between the switching node and host computers of many different types. It is important to allow a logical match between the switching node computer word length and the varying word lengths of the host computers, and also to allow the input/output routines of the switching node and the input/output routines of the host computer to service the interface "cooperatively" without placing an undue processing burden or tight timing constraints on either machine. The approach taken was to design a bit-serial interface which could logically stop after any bit, and to then require that each host computer build (obtain) a specialized small hardware unit called a "special host interface" which would be installed in the host machine itself to serve the network connection. Thus, a logical and electrical <u>standard</u> was specified by the switching network in a manner intended to minimize overall trouble to all hosts, then each host was required to meet that standard both in hardware and software.

- SWITCHING NODE PERFORMANCE - A central goal of the network was to provide resource sharing between remote installations in such a way that a local user could employ a remote resource without degradation. In particular, this required that the subnetwork be sufficiently reliable, have sufficiently low delay, have sufficiently great capacity and have sufficiently low cost, that remote use would be "as attractive" as local use. These objectives translated into a major technical problem for the switching node itself to provide low delay and high capacity at modest cost. The approach taken was to select a mini computer for the switching nodes whose I/O system was very efficient and to write very carefully tailored programs in machine language to optimize the capacity and the low delay of the data path in the switching node. Great attention was paid to minimizing the operating time of the inter loops of these programs.

o REMOTE CONTROL - A special and new problem was posed by the need to put dozens of small identical mini computers in the field, in an environment where the host connections to those computers was somewhat experimental and where the programs in the mini computers themselves were under development and were changing from month to month. It was important to be able to do debugging of software, debugging of new host connections, program modifications, and the installation of new programs without the costs and difficulties associated with having manned sites. The approach taken was to develop an entirely new technology of remote computer management. A Network Control Center was established for the subnetwork and software was developed which made it possible to examine or change the operating software in any node of the net from the central network control center. This approach made it possible to issue complete new versions of the software to each node in the network from a central place in an hour or two. In addition, each node reports the state of its health to the central place periodically and provides information on which to base debugging and maintenance activities.

o ROUTING - In a non-fully connected network, an important problem is the decision process by which each node decides how to route information to reach any particular destination. This is a difficult theoretical problem and there are many different approaches, including fixed routing, random routing, centrally controlled routing, and various forms of distributed adaptive routing. The approach taken was to use a distributed adaptive traffic routing algorithm which estimates on the basis of information from adjacent nodes the globally proper instantaneous path for each message in the face of varying input traffic loads and local line or node failures. Each IMP keeps tables containing an estimate of the output circuit corresponding to the minimum delay path to each other IMP, and a corresponding estimate of the delay. Periodically, each IMP sends its current routing estimates to its neighbors; whenever an IMP receives such a message it updates its internal estimates. Thus, information about changing conditions is regularly propagated throughout the network, and whenever a packet of traffic must be placed on the queue for an output circuit, the Imp uses its latest estimate of the best path.

o HOST PROTOCOL - In many ways a computer network is to host computers as the telephone system is to human users: a transparent communication medium in which even after the caller has learned how to insert dimes and dial, it is still necessary that he speak the same language as the person called in order for useful communication to occur. The common language is referred to as host protocol, and the problem is to design a host protocol which is sufficiently powerful for the kinds of communication that will occur and yet can be implemented in all of the various different host computer systems. The initial approach taken involved an entity called a "Network Control Program" which would typically reside in the executive of a host, such that processes within a host would communicate with the network through this Network Control Program. The primary function of the NCP is to establish connections, break connections, switch connections, and control flow. A "layered" approach was taken such that more complex procedures (such as File Transfer Procedures) were built on top of simpler procedures in the host Network Control Program.

2.3 Major Changes in Objectives and Approaches

The ARPANET development was an extremely intense activity in which contributions were made by many of the best computer scientists in the United States. Thus, almost all of the "major technical problems" already mentioned received continuing attention and the detailed approach to those problems changed several times during the early years of the ARPANET effort. However, in addition several more major changes in objectives were introduced once the initial network became operational:

o THE TIP - The initial nodal switching units, called IMPs, were intended to interconnect computers and high bandwidth phone lines. At the outset all terminal access to the network was via terminal connections to the host themselves. After a time it became clear that there was a population of users for which terminal access to the network was very desirable, but who were not conveniently able to access the network via a host computer. Thus, a new nodal switching unit, a Terminal Interface Message processor, or TIP, was defined to serve the purpose of an IMP plus an additional function of direct terminal access. This shift resulted in the design of a TIP which really was a tiny host embedded in a switching node itself and permitting the direct connection of up to 63 asynchronous character-oriented terminals to the switching node. The TIP became the nodal switching unit of choice, often even where there was a local host computer; this allowed connection of both hosts and terminals at that location directly to the network.

o PLURIBUS - As the network grew the projected network traffic and the projected fanout of hosts and terminals at a node began to exceed the capacity of the IMPs and TIPs in the Honeywell series. In order to plan for a future in which there might be circuits of 1 megabit or more, it was decided to embark on the construction of a new switching node which could handle higher bandwidths and higher fan-out. The approach taken was to design a multiprocessor switching node which would permit modular growth as the bandwidth and fanout requirements increased. This approach represented an interesting and important departure in computer technology as well as a necessary approach to achieving greater performance in network nodes.

o SIMP - The initial ARPANET circuits were all 50 kilobit land lines. As the program developed, it became desirable to consider the use of satellite links and further desirable to attempt a more sophisticated use of

3. SCIENTIFIC AND TECHNICAL RESULTS AND ACCOMPLISHMENTS

3.1 Results of the Effort in Relation to the Program Objectives

In some cases a major program can be seen to reach its objectives in a single instant; a mushroom cloud at Alamogordo, or Armstrong stepping on the moon. In other cases like the ARPANET, an equally important objective may be reached with equal success, but the event must be observed in a more complicated way and over a longer period of time. The first ARPANET objective was "to develop techniques and obtain experience on interconnecting computers in such a way that a very broad class of interactions are possible". Not just techniques, but an entire technology of packet switching has been developed, and an enormous body of experience has been developed on interconnecting computers to allow a broad class of interactions. The second objective was "to improve and increase computer research productivity through resource sharing". Computer research productivity has been improved and has been increased through resource sharing over the ARPANET. Further, computer research productivity has been improved and increased in even some unexpected ways by an ARPANET-induced social change; within the

ARPA research community, it has become more fashionable to work closely with other remote groups of researchers.

Another objective was to permit the linking of specialized computers to the many general purpose computer centers. Several major specialized computers have been linked over the ARPANET to the main general purpose computer centers in an extremely successful fashion.

At a technological level, the overall objectives were to construct a subnetwork whose reliability, delay characteristics, capacity, and cost would facilitate resource sharing between the computers in the network; and to understand, design and implement the Host protocols to permit such resource sharing. Such a subnetwork, consisting of over 50 nodes and stretching from Hawaii to Norway, was successfully constructed; and the necessary Host protocols to allow resource sharing between the connected computers were successfully understood, designed and implemented.

From a programmatic point of view, the initial IMP contract was awarded 6 months later than had been anticipated by the program plan, but the demonstration of initial net operation with 4 nodes and later the extension to 19 nodes took place on a time

The first, called "network mail", was a case where a relatively simple idea which, in rudimentary form, had been a "curiosity" at some installations for years, suddenly and quite unexpectedly became very successful. The current form of the system is called "network mail", and provides computer "mailboxes" for many individuals in their favorite Host computer. Whenever A desires to communicate a message to B, C, D, and E, he logs into a convenient Host, uses a program that sends mail, lists the addresses with an indication of their mailboxes, and types the message. No matter where B, C, D, and E might be, they occasionally enter the net at a local node, log into their (mailbox) Host, and as part of the login process are automatically told that a message exists for them. They may then obtain any recent messages by using another program to read their mail. The desire to provide mail facilities of this type became an end in itself and partially influenced network growth and modification.

The second, use of the ARPANET for digitized speech experiments, was a case where improvements in the vocoder technology (and the continuing defense need for encrypted speech) led to a serious investigation of speech transmission over packet networks. Since speech traffic posed different requirement on network performance (e.g., low delay) changes were required in the network IMPs to accommodate this traffic. This activity is still growing in importance and is now being pursued by ARPA separately from the ARPANET program.

the satellite channel than simple point-to-point connections. In particular, the approach taken was to consider multi-access use of a single up frequency and a single down frequency where all stations can hear all other transmissions including their own. This approach required the development of a new kind of switching node called a Satellite Interface Message Processor (SIMP) which could handle the complex algorithms required for multi-access use of the satellite channel. In the final years of the ARPANET project prior to the transfer of the ARPANET to DCA, the initial work on such Satellite IMPs was accomplished. However, the full utilization of this equipment is still in progress under ARPA support and is being pursued separately from the ARPANET project itself.

o PLI - The ARPANET was initially intended for use in an entirely unclassified fashion between ARPANET-supported research groups. However, a clear intention of the activity was to understand techniques which might be applied to defense department needs, and one such need implied a desire to use the ARPANET for classified traffic. Since all of the circuits and most of the nodes existed on entirely unclassified terrain, an end-to-end encryption scheme was needed. However, this posed technical problems because all network nodes require addresses in the clear, and because no available encryption units matched the formats required by the ARPANET IMPs and TIPs. The approach taken was to design an assemblage of two minicomputers and a KG-34 encryption unit to be a "Private Line Interface", or PLI, and to allow classified traffic between two secure hosts to take place as a Host-PLI-ARPANET-PLI-Host connection. This project was conducted in cooperation with NSA, and had the additional interesting attribute that it required (probably for the first time) the "certification" by NSA of minicomputer software to be used as part of encryption scheme.

In the latter years of the ARPANET project, two "applications" of the ARPANET became sufficiently important to influence the overall objectives of the program.

scale that was extremely close to the incremental time anticipated in the program plan. However, as the success of the ARPANET became obvious after about 2 years, decisions were made to grow the net to a size that had never been anticipated in the program plan; growth to more than 50 nodes over a geographic area from Hawaii to Norway was certainly not originally anticipated. In similar fashion, the cost in the first 2 fiscal years of the program were very close to anticipated costs, but as decisions were made to expand the scale of the network, the cost no longer followed the program plan.

In short, the results of the effort were eminently successful and far more than adequately met the objectives initially stated. The success of the program far exceeded even the most optimistic views at the time of inception.

3.2 Technical Aspects of the Effort Which was Successful and Aspects of the Effort Which Did Not Materialize as Originally Envisaged

Since the ARPANET program as a whole was a major success and since the whole really is the sum of the parts, it is clear that the many key technical aspects of the program were in turn successful. A modest list of technical successes is as follows:

- o powerful computer-based techniques for topological optimization were developed and the choice of ARPANET topology was made with the aid of these tools;

- o the carriers successfully provided high reliability 50K/sec circuits;

- o a subnetwork including nodal switching IMPs and TIPs was constructed whose performance, reliability, and cost did facilitate resource sharing;

- o The ARPANET provides a convincing demonstration that adaptive routing algorithms can be made to perform reliably (e.g., in a globally correct mannner in the face of local failures), efficiently (e.g., adapting to changes in the network quickly and accurately), and flexibly (e.g., accommodating a variety of circuit bandwidths and internode distances) without excessive complexity and overhead.

- o The ARPANET has demonstrated that it is possible to build a large operational network in such a way that the effects of component failures are localized rather than "crashing" or otherwise making non-operational large portions or all of the network. A node or a Host can fail in the ARPANET and network use will be prevented for only the few users directly connected to that node or using that Host.

- o The ARPANET has confirmed the theoretical result that networks which store-and-forward packets can achieve delays which are low when compared to the delays incurred in the computers (Hosts) which are using the network.

- o The ARPANET has demonstrated that a network can be constructed so that nodes, lines, traffic, and so on can be added or deleted without major upheavals with each addition.

- o The ARPANET has demonstrated that it is possible for a network to control and operate itself without explicit control from a control center.

o The ARPANET has demonstrated new techniques for monitoring, maintenance, and debugging. The nodes in the ARPANET typically operate at sites where there is no knowledgeable person available locally. The nodes automatically report their status (via the network itself) to a network monitoring center, and maintenance and debugging (of both the software and hardware) are typically carried out (again via the network) from the monitoring center.

o Possibly the most difficult task undertaken in the development of the ARPANET was the attempt -- which proved successful -- to make a number of independent Host computer systems of varying manufacture, and varying operating systems within a single manufactured type, communicate with each other despite their diverse characteristics.

o the Pluribus multiprocessor technology, important in its own right, was developed to provide an improved performance nodal switching unit;

o a set of Host protocols was hammered out between the Host organizations and resulted in a layered structure of Host protocols that did facilitate resource sharing.

Inevitability there were technical areas which proved less tractable than had been hoped and some kinds of network utilization developed more slowly than had been anticipated.

Problems of routing, flow control, and congestion control in the subnetwork turned out to be more difficult than had been originally expected. These algorithms required intensive investigation and were modified several times in the early years of the network's growth. Luckily, the improvements in the algorithms managed to stay slightly ahead of the growth in network size and traffic and, therefore, difficulties with the algorithms never represented an impediment to resource sharing on the network.

It proved more difficult to agree on the specification of adequate Host protocols than had been originally expected and this difficulty delayed wide use of the network for at least a year longer than had been anticipated. However, the eventual design of the Host protocols were eminently successful and provided a strong base for resource sharing.

The development of a widely-useful on-line Network Information Center (NIC) proved to be a difficult project. Although very interesting and important computer-based tools for information handling were developed, the actual timely collection and on-line publication of detailed descriptions of resources at the various Host sites proved nearly intractable. Further, the computer tools that were developed while rather elegant were not so clearly cost effective for wide scale remote use. Eventually, the goals of the NIC were substantially reduced and at the time of the ARPANET transfer to DCA, it was generally necessary to obtain detailed information about the resources at a particular site directly from the site.

Many different kinds of resource sharing use were anticipated at the inception of the ARPANET program. Of these, the remote use of one program by another was one of the more distant goals. While some of this kind of resource sharing has, indeed, taken place, such use has grown more slowly than other network uses.

On a less scientific and more programmatic note, the initial program plan anticipated a technology transfer of the network to a common carrier and the availability of the network to other communities of interest such as the NSF and the NIH. The transfer that did eventually take place was to the Defense Communications Agency and although some utilization has been made of the ARPANET by other agencies like the NIH, it proved rather inconvenient to encourage major utilization outside the Defense Department.

4. APPLICATIONS AND CONSIDERATIONS FOR THE FUTURE

4.1 Conclusions of Technical Feasibility

The ARPANET project proved the technical feasibility of achieving reliable high performance, cost effective digital communications by means of packet switching technology and, in turn, the technical feasibility of operating a resource sharing computer network based on this technology. The ARPANET project also proved the feasibility of achieving closely knit communities of technical interest over a wide spread geographic area; it is possible that this social feasibility demonstration is as important as the many technical feasibility demonstrations.

4.2 Recommendations on Additional R&D Requirements and Opportunities

The ARPANET program has represented a significant marriage of computers and communications and since it was the first such marriage of this magnitude, it is not surprising that the program has many different kinds of research progeny. These new requirements and opportunities fall in several different classes:

- o opportunities for research in the specific packet switching technologies inherent in the ARPANET;

- o opportunities for research concerning other kinds of packet switching networks employing related technologies;

- o opportunities for research in connecting and integrating different kinds of packet switching networks;

- o opportunities for research in new uses of networks and new techniques in Host computers for taking advantage of networks;

- o opportunities for research in related technologies.

Each of these categories will be separately discussed:

ARPANET Specific R&D

At the point in the ARPANET project where ARPA began to consider transfer of the network to the aegis of the Defense Communications Agency, direct ARPA support of research on the specific packet switching techniques in the ARPANET was necessarily reduced. However, the network itself continued to grow, both in size and utilization and some technical problems clearly required additional research and development. Perhaps the most pressing single technical problem of this type was the "routing" problem, and there is currently a clear need for improvement in the specific ARPANET routing techniques. Closely related areas like flow control and congestion control would necessarily require re-examination in connection with the detailed review of the routing issue.

<u>Other Packet Switching Technologies</u>

In the course of research on the ARPANET, two other related kinds of packet switching technology were initially investigated: packet satellite technology and packet radio technology. It is clear that these related packet switching technology areas represent major research and development opportunities with strong basic requirements flowing from the needs of the Defense Department. In fact, ARPA is currently pursuing major R&D investigations in both of these areas and significant progress is being made.

Another R&D opportunity exists in use of packet switching techniques on "buses" such as coaxial cables, or power grids; some research in this area has already taken place.

Internetting

As ARPA research leads to other networks such as packet radio networks, packet satellite networks, and as other packet switching networks within the Defense Department, in the U.S. public sector, and in other countries come into being, there is a clear and pressing need to understand how to interconnect such networks in a technically adequate and cost effective fashion. Again, clear requirements flow from defense needs and major opportunities for progress are available. ARPA is currently supporting a major R&D activity on the investigations of internetting technology and on experimental attempts at internetting.

Experimental Network Use

Perhaps not surprisingly the largest area of R&D opportunity lies in the direction of attempting to use this new packet switching technology in many different ways to the advantage of the Defense Department. Some such research and development is already under way and excellent opportunities exist for cooperative R&D programs between ARPA and the various other components of the Defense Department.

An attractive R&D opportunity exists in the investigation of speech transmission over packet switching networks. There is an extremely serious Defense requirement for encrypted speech and the confluence of packet switching technology and improved vocoder technology is providing a major R&D opportunity for achieving cost effective digital speech transmission and the related possibility for such speech transmission to be encrypted.

The surprisingly successful experimental use of network mail has probably opened a new era in communications in the Defense Department. It is expected that such mail systems built on packet switching networks will have an enormous impact in both the Defense Department and the remainder of the public and private sectors. A major research and development opportunity exists to improve the performance and the cost effectiveness of such mail systems and to experimentally deploy them in the Defense Department.

It is clear that packet switching networks have a very direct application to command and control and a significant research opportunity exists in attempting to investigate such command and control applications. ARPA is already proceeding to develop testbeds whereby packet switching technologies and other related technologies can be experimentally employed in cooperation with one or more of the military services. There is a clear requirement for improved command and control and the packet switching technology developed in the ARPANET provides a major opportunity to make progress towards this goal.

At a deep technical level, the proper design of Host operating systems for efficient and cost effective connection to networks is still in the area of significant research and development opportunity. In the ARPANET project, the network connections were "addons" and it is clearly time to mount the more detailed investigation of how best to accomplish such connections.

Related Technology

In the course of the ARPANET program other opportunities for research and development were observed, some closely related to packet switching technology and some really quite distinct technologies in their own right.

The general topic of computer operating systems security and communications security has long been a subject of research and development interest in ARPA. The advent of packet switching technology has permitted a modified viewpoint to be taken in some of these areas of security research. There is now, for example, a greater concurrence in the security community that mini-computer software can be certified and then trusted deep within the guts of encryption systems. Second, packet switching technology permits a different viewpoint to be taken on the techniques for key distribution and there are research and development opportunities specifically in this area currently being pursued by ARPA. Thus, packet switching networks have created additional requirements on security technology and at the same time provided additional opportunities.

In the course of the ARPANET development, the Pluribus multiprocessor project was initiated in order to provide a high performance nodal switching unit. However, a multiprocessor technology is, in principle, broadly useful across the spectrum of data processing. Thus, the developments in multiprocessor technology that took place within the ARPANET program provide an enticing opportunity for further research and development in the design and construction of multiprocessors for other Defense computational needs.

The techniques for remote control of computers in the field developed within the ARPANET project are probably more broadly applicable to the management of computer resources in other areas of the Defense Department. These remote management techniques represent another opportunity which has grown out of the ARPANET experience.

It was earlier indicated that it was not tractable for the Network Information Center in the ARPANET to keep current on-line detailed descriptions of program resources available within the Host community. This difficulty in turn represents a research and development opportunity for the future. Specifically, research and development is required on how to properly describe computer programs and how to create standards for such descriptions such that it will be easier to create compendia of available resources. In other words, despite the great success of the ARPANET, the basic resource sharing problem still represents a fertile area for research and development. Now that the networking technology itself is "given", attempts at description and documentation of Host resources might more easily yield to a research and development effort.

5. PROGRAM IMPACT AND ASSESSMENT OF TECHNOLOGY DEVELOPED

5.1 Service Use of Technology

Seldom has ARPA had greater success in convincing its client, the remainder of the Defense Department, to adopt a particular technology. Not only was it possible to transfer the ARPANET itself to the Defense Communications Agency (Code 535), but the Defense Communications Agency has already embarked on the procurement of its primary backbone major communications system of the future -- AUTODIN II -- based very specifically on the packet switching technology developed in the ARPANET. Even beyond this very major step all three services are actively involved in investigations of packet switching technology to their specific needs.

5.2 Impact on Non-DoD Programs

In just the very short time since the inception of the ARPANET this technology has already resulted in the formation of a new industry: a private sector development of "value added" packet switching networks. Two corporations, Telenet and Tymeshare, are currently marketing packet switched communications to the general public as common carriers under the Communications act of 1934. It is difficult to recall any other ARPA program which has had such a direct and immediate exploitation in the commercial sector. In addition, all over the world new communications systems are being designed and built to take advantage of the packet switching technology demonstrated by the ARPANET project. At least three countries -- Great Britain, France and Canada -- have major national PTT- sponsored packet switching networks either already operating or under development and many other countries are actively pursuing this technology.

5.3 Applications of the Program Results

The most specific result of the ARPANET Program has, of course, been the ARPANET itself; and the ARPANET itself is currently fully operational under the management of the Defense Communications Agency and is actively serving thousands of individuals on a daily basis.

5.4 Advance in the State-of-the-Art

The ARPANET program has represented a first-rank advance in the state-of-the-art of communications and the state-of-the-art of computer technology. The greatest advance has been in the provision of cost effective, reliable, high performance digital communications, but very significance state-of-the-art advances have also taken place in many other areas, such as topological optimization, routing, multiprocessor technology, protocols for resource sharing between programs, network mail systems, and remote computer management. Computer and communications technology will never be the same.

6. BIBLIOGRAPHY OF REPORTS

There has been an enormous outpouring of literature about the ARPANET in particular, and about resource sharing computer networks in general. Literally an industry has been formed in the space of 2/3 of a decade and the amount of literature reflects this really unusual metabolism.

The initial seminal papers on the ARPANET were presented in May of 1970 at the AFIPS Spring Joint Computer Conference in Atlantic City and are published in the Proceedings of that conference (AFIPS Conference Proceedings, Vol 36, AFIPS Press, 210 Summit Avenue, Montvale, New Jersey 07645). Another early ARPANET session was held at the 1972 Spring Joint Computer Conference in Atlantic City and these papers are published in AFIPS Conference Proceedings, Vol. 40. These two early sets of papers represent a sensible introduction to the early notions and plans for the ARPANET.

A sizeable bibliography and index to publications about the ARPANET was published in 1976 with ARPA support: "Selected Bibliography and Index to Publications About the ARPANET", Becker and Hayes Inc., February 1976, 185 pages, AD-A026900. This document represents the most complete available listing about ARPANET publications, but it is a bibliography only and does not provide help in trying to select which of the many references would be suitable to look at in what order. Another bibliography on the literature of resource sharing computer networks in general (not just the ARPANET) has been issued by the U.S. Department of Commerce, National Bureau of Standards: "Annotated Bibliography of the Literature on Resource Sharing Computer Networks", NBS Special Publication 384, revised 1976. This document is also not very useful in helping the reader decide what is important and what is not.

Several volumes of reprints have been published which deal with computer networks in general (rather than the ARPANET specifically), but which attempt to collect together the important papers themselves and provide a much easier entry to the literature than the large bibliographies: (1) "Advances in Computer Communications", Wesley W. Chu, Artech House, Inc., 610 Washington St., Dedham, Massachusetts 02026, 1974; (2) "Computer Communications", edited by Paul E. Green, Jr. and Robert W. Lucky, IEEE Press, 345 East 47th Street, New York, New York 10017, 1975; and (3) "Computer Networking", edited by Robert P.

Blanc and Ira W. Cotton, IEEE Press, 345 East 47th Street, New York, New York 10017, 1976.

A very hefty, but fairly readable compendium with a very large list of references is "Infotech State of the Art Report 24, Network Systems and Software", Infotech International Ltd., Nicholson House, Maidenhead, Berkshire, England, 1975. This document deals with far more than just the ARPANET, but it tries to put the ARPANET in context with other current work as of 1975.

There are three important reference documents which have been prepared for the Defense Communications Agency by the Network Information Center at Stanford Research Institute, Menlo Park, California
94025 which are specifically addressed to various aspects of the ARPANET: (1) "ARPANET Resource Handbook", NIC 39335, December 1976; (2) "ARPANET Protocol Handbook", NIC 7104, Revision 1, April 1976; (3)"ARPANET Directory", NIC 36437, July 1976.

A textbook on computer networks (and almost the only useful one so far) is Davies and Barber, "Communication Networks for Computers", John Wiley, 1973.

There have been several recent conference and conference proceedings dealing with computer networks and some selected recent papers from these conferences concerning the ARPANET are as follows:

***

7. DISTRIBUTION LIST

The ARPANET Completion Report should be made publically available through NITS, and copies directly sent to the following:

Former Directors of ARPA involved in the ARPANET effort:

Charles M. Herzfeld
Eberhardt Rechtin
Stephen J. Lukasik

Former Directors of ARPANET/IPTO involved in the ARPANET effort:

J.C.R. Licklider
Ivan Sutherland
Robert Taylor
Lawrence Roberts

Former ARPANET Program Managers:

Stephen Crocker
Craig Fields

Some individuals with special interest:

Paul Baran
Thomas Marill
Donald Davies
Leonard Kleinrock
Howard Frank
Douglas Engelbert
Frank Heart

Current ARPANET Sponsors Group:

Naval Ship Research and Development Center
Defense Communications Agency -- ARPANET Management Branch
NASA Headquarters

USACC Support Branch
National Security Agency
Defense Communications Engineering Center
Air Force Systems Command Headquarters
Energy Research and Development Administration
National Bureau of Standards
Advanced Research Projects Agency

Selected DoD agencies and offices:

Other selected government agencies:

Selected congressional committees:

Selected civilian institutions:

CHAPTER III: A REVIEW OF THE ARPANET PROJECT

1. HISTORY

The ARPA Computer Network, or ARPANET as it has come to be called, consists of IMPs, lines, and hosts as shown in Figure 1. The IMPs (short for Interface Message Processors) are small, special-purpose computers connected to each other by telephone lines. The hosts are a heterogeneous collection of computers

Figure 1 -- IMPs, Lines, and Hosts

used for a variety of applications. The IMPs provide a communications subnetwork through which hosts communicate with each other, much as the commercial telephone system provides a communications subnetwork through which humans communicate. In other words, the IMPs and lines make up a communications utility of which the hosts are users. Each host is connected to one IMP; one IMP may have several hosts connected to it.

When data is to be sent from one host to another, the data is broken into discrete entities called "messages" at the sending host, which is also called the "source host". Each message consists of some data and an address. The address specifies to which host the data is to be sent, that is, the "destination host". Successive messages are passed from the source host to its IMP, where they are broken into smaller entities called "packets", each of which carries the same address as the message. The packets are routed from IMP to IMP across the network until they arrive at the IMP to which the destination host is connected, where the original messages are reconstructed from the packets, and passed to the destination host.

The ARPANET was conceived in the middle to late 1960s as a project to be sponsored by the Information Processing Techniques

Office of the Advanced Research Projects Agency (ARPA) of the U.S. Department of Defense (DoD). Construction of the network began in 1969. By 1975 the network had gone through several stages of development and for all intents and purposes had become an operational Department of Defense computer network. In 1975, control of the network was transferred from ARPA to the U.S. Defense Communications Agency, an agency better suited to the administration of a working facility.

The ARPANET is a major development in the evolution of computer communications. Our purpose here is to present the history of the ARPANET: why it was built, who helped build it, the major design decisions (and some of the minor ones) associated with it, its evolutionary development, its maturity, and why it is important.

1.1 Background

Before relating the history of the ARPANET itself, it is appropriate to summarize the events and circumstances out of which the ARPANET came into being. What is ARPA? What was the state of computer communications technology prior to the ARPANET? From what other projects is the ARPANET descended? These questions we answer in the following subsections.

1.1.1 Capsule History of ARPA and Its Information Processing Techniques Office

ARPA was formed as an agency under the Secretary of Defense as part of the U.S. reaction to Russia's launch of Sputnik in the mid-1950s. Soon after ARPA's formation, the U.S. space program was moved out of the military to become NASA, and ARPA focused on the areas of ballistic missiles, solid missile propellants, and materials science. From these areas ARPA gradually expanded its interests to a variety of sometimes esoteric fields in pursuit of the kind of far-reaching research projects that could truly be called advanced. The directors of ARPA have been Roy Johnson (1958-1959), General Austin Betts (1960), Dr. J.P. Ruina (1961-1963), Dr. Robert L. Sproull (1963-1965), Dr. Charles M. Herzfeld (1965-1967), Dr. Eberhardt Rechtin (1967-1970), Dr. Stephen J. Lukasik (1971-1974), and Dr. George H. Heilmeier (1975-present). <a sentence characterizing each director>.

In 1961, Command and Control Research (CCR) was assigned to ARPA. An initial task of CCR was to capitalize on the government's investment in the Q-32 computers, software, and people at the Systems Development Corporation (SDC). Dr. Ruina chose Dr. J.C.R. Licklider to head the CCR effort and Licklider joined ARPA in October 1962. In taking the job, Licklider saw that improvements in command and control would be heavily dependent on fundamental advances in computer technology, and he was committed to seeking advances in that field, particularly in the subfield of interactive computing. Licklider changed the focus of CCR from a prior reliance on SDC to the setting up of research contracts with the best academic computer centers, including SDC. Prophetically, Licklider nicknamed the group of computer specialists he gathered together the "Intergalactic Network". By the beginning of 1964 the CCR program had developed into a far-reaching basic research program in advanced computer technology, and Licklider's office was renamed Information Processing Techniques (IPT). Following Licklider (1962-1964) as Director of IPT have been Dr. Ivan Sutherland (1977-1977), Dr. Robert Taylor (1977-1969), Dr. Lawrence Roberts (1969-1973), Dr. J.C.R. Licklider again (1977-1977), and Col. David Russell (1977-present).

The centerpiece of the new IPT program was Project MAC at MIT, and the focus of Project MAC was its work in time-sharing, in particular the development of the first timesharing system capable of supporting many users. This system was known as the Compatible Time-Sharing System or CTSS. By 1964 CTSS had proven such a success that plans were being laid for a second generation system which became known as MULTICS. By the late 1960's and early 1970's, largely because of ARPA support of timesharing research and development of several early timesharing systems, timesharing had become an accepted component of the computer industry.

Timesharing was only the most prominent of accomplishments beginning to derive from IPT programs during the 1963-1965 period. There were important developments in computer graphics, such as the RAND Tablet. Important contributions to programming techniques and languages (e.g., LISP) were made. Support was provided for research in display techniques, computer architecture, and "artificial intelligence". By 1965 ARPA had attained a reputation as the dominant source of support for truly advanced information processing research.

In the 1965-1966 period IPT began to shift its emphasis from research to development. By 1967 MULTICS was under development, ARPA had committed itself to the development of ILLIAC IV, and the investigation had begun into means for interconnecting geographically separated computers. Numerous other research efforts were also supported, especially in the software area.

The 1967-1970 period saw the ARPANET effort firmly underway. From that period to the present ARPA has continued to support a number of advanced efforts in computer research and development.

Over the years ARPA in general and IPT in particular have achieved a number of notable successes. They have had a knack for sensing when the time was ripe for a particular development and then acting quickly to undertake that development; and they have been able to support developments with sufficient money. IPT has probably contributed more than any other computer science laboratory in the country (one is tempted to say "than all other laboratories") in most advanced areas of information processing.

1.1.2 The State of the Art of Computer Communications

Circa 1967

<this section will be written for the second draft>

1.1.3 The RAND Study of Distributed Communications Networks.

One of the most important early studies of computer networks was performed by Paul Baran and his colleagues at the RAND Corporation in the early 1960s. Many concepts central to the later development of the ARPANET and other computer networks were first described in the series of reports published by RAND in 1964 (a list of these reports is given in the bibliography at the end of this subsection). These ideas include the improved reliability of a distributed network structure over a centralized or star network and over so-called decentralized networks made up of a collection of smaller star networks. Extensive studies were undertaken, including simulation of some grid networks, to determine how "survivable" a distributed network could be expected to be after heavy node and link failures. This study was particularly concerned with the question of keeping a high percentage of the network available and performing well in the face of enemy attacks on the network, from the point of view of its suitability for Department of Defense applications.

In specifying closely the engineering details of what was called the "Distributed Adaptive Message Block Network", Baran anticipated many of the developments in practical networks that came a full decade later. In the Distributed Adaptive Message Block Network, a "multiplexing station" connects up to 1024 terminals of widely differing characteristics. Automatic user-to-user cryptography is integrated into the network switching technique to ensure efficiency. Both satellite links and low-cost microwave relay systems are suggested as techniques for providing the network with very high data rate circuits. The concept of a "message block" is introduced: a packet of up to 1024 bits of header and data, which is the unit of data transferred in the network. One of the most interesting aspects of this study is that it concluded that a large-scale digital transmission network was not only feasible but also highly cost-effective, and proposed that many of the switching functions be implemented in hardware. Baran was considering ways of making extremely reliable networks, and so preferred simple solutions and reliable hardware where possible.

The following bibliography includes the entire set of eleven reports in the original RAND study as well as two published papers resulting from that set and from two later reports. An annotated version of this bibliography is included in "Adaptive Routing Algorithms for Distributed Computer Networks" (John M. McQuillan, BBN Report No. 2831, May 1974, pp. 14-17).

Bibliography

P. Baran, "On Distributed Communications: I. Introduction to Distributed Communications Networks," RAND Corp. Memo RM-3420-PR, August 1964, 37p.

S. Boehm and P. Baran, "On Distributed Communications: II. Digital Simulation of Hot-Potato Routing in a Broadband Distributed Communications Network," RAND Corp. Memo RM-3103-PR, August 1964, 49p.

J. W. Smith, "On Distributed Communications: III. Determination of Path-Lengths in a Distributed Network," RAND Corp. Memo RM-3578-PR, August 1964, 91p.

P. Baran, "On Distributed Communications: IV. Priority, Precedence, and Overload," RAND Corp. Memo RM-3638-PR, August 1964, 63p.

P. Baran, "On Distributed Communications: V. History, Alternative Approaches, and Comparisons," RAND Corp. Memo RM-3097-PR August 1964, 51p.

P. Baran, "On Distributed Communications: VI. Mini-Cost Microwave," RAND Corp. Memo RM-3762-PR, August 1964, 101p.

P. Baran, "On Distributed Communications: VII. Tentative Engineering Specifications and Preliminary Design for a High-Data-Rate Distributed Network Switching Node," RAND Corp. Memo RM-3763-PR, August 1964, 85p.

P. Baran, "On Distributed Communications: VIII. The Multiplexing Station," RAND Corp. Memo RM-3764-PR, August 1964, 103 p.

P. Baran, "On Distributed Communications: IX. Security, Secrecy, and Tamper-Free Considerations," RAND Corp. Memo RM-3765-PR, August 1964, 39p.

P. Baran, "On Distributed Communications: X. Cost Analysis," RAND Corp. Memo RM-3766-PR, August 1964, 21p.

P. Baran, "On Distributed Communications: XI. Summary Overview," RAND Corp. Memo RM-3767-PR, August 1964, 23p.

P. Baran, "On Distributed Communication Networks," IEEE Transactions on Communications Systems Vol. CS-12, March 1964, pp. 1-9.

B. W. Boehm and R. L. Mobley, "Adaptive Routing Techniques for Distributed Communications Systems," RAND Corp. Memo RM-4781-PR, February 1966, 78p.

B. W. Boehm and R. L. Mobley, "A Computer Simulation of Adaptive Routing Techniques for Distributed Communications Systems," RAND Corp. Memo RM-4782-PR, February 1966, 35p.

B. W. Boehm and R. L. Mobley, "Adaptive Routing Techniques for Distributed Communications Systems," IEEE Transactions on Communication Technology, Vol. COM-17, No. 3, June 1969, pp. 340-349.

1.1.4 The Lincoln/SDC Experiment

In 1965 Thomas Marill and his colleagues at Computer Corporation of America (in Cambridge, Massachusetts) were commissioned by M.I.T. Lincoln Laboratory to study the concept of computer networking. The primary technical contact at Lincoln Laboratory was Lawrence Roberts, who played an active role in the network study; and the contract was, in fact, a subcontract under the Laboratory's ARPA contract. The study was done in late 1965 and a report on the study was issued in 1966 ("A Cooperative Network of Time-Sharing Computers", Thomas Marill, Computer Corporation of America, Technical Report No. 11, June 1, 1966; a later paper of the same name authored by Thomas Marill and Lawrence Roberts also appeared in the Proceedings of the AFIPS 1966 Spring Joint Computer Conference, pp. 425-431). The report examined the basic idea of computer networking, considered the available communications techniques and software problems, and recommended that a three-computer experimental network be constructed. The report suggested linking three existing computers, the AN/FSQ-32 at Systems Development Corporation, the IBM 7094 at MIT's Project MAC, and Lincoln Laboratory's TX-2. Later in 1966, CCA received another contract to carry out the linking of the Q-32 and the TX-2. The Q-32 and TX-2 were in fact linked together, and the link was demonstrated. Later a small Digital Electronics Corporation machine at ARPA was added to this network, by now known as "The Experimental Network". It is noteworthy that The Experimental Network linked host computers directly, and did not use IMPs.

1.1.5 The NPL Data Network

Another early major network development was undertaken at the National Physical Laboratory in Middlesex, England, under the leadership of D. W. Davies. The broad system design of the NPL Data Network, as it was called, was first published in 1967, and bears a resemblance to the network proposed by Paul Baran at RAND, and to the ARPANET. The NPL Data Network was specified to be a packet-switching network and was to have a hierarchical structure. It was proposed that "local networks" be constructed with "interface computers" which had responsibility for multiplexing among a number of user systems and for communicating with a "high level network". The latter would be constructed with "switching nodes" connected together with megabit rate circuits.

While there was considerable technical interchange between the NPL group and those who designed and implemented the ARPANET, the NPL Data Network effort appears to have had little fundamental impact on the design of the ARPANET. Such major aspects of the NPL Data Network design as the standard network interface, the routing algorithm, and the software structure of the switching node were largely ignored by the ARPANET designers.

There is no doubt, however, that in many less fundamental ways the NPL Data Network had effect on the design and evolution of the ARPANET.

Considerable detail about the NPL Data Network may be found in the textbook, written by two of the network's designers, entitled Communications Networks for Computers (D.W. Davies and D.L.A. Barber, John Wiley & Sons publishers, 1973). A bibliography of published papers resulting from the NPL Data Network effort follows.

Bibliography

D. W. Davies, K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals," ACM Symposium on Operating Systems Problems, October 1967, 17p.

K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Data Communication Network for Real-Time Computers," IEEE International Conference on Communications, U.S.A., June 1968.

D. W. Davies, "Communication Networks to Serve Rapid Response Computers," Proceedings of the IFIP Congress 1968, Vol. 2 - Hardware Applications, pp. 650-658.

K. A. Bartlett, "Transmission Control in a Local Data Network," Proceedings of the IFIP Congress 1968, Vol. 2 - Hardware Applications, pp. 704-708.

D. W. Davies, "The Principles of a Data Communications Network for Computers and Remote Peripherals," Proceedings of the IFIP Congress 1968, Vol. 2 - Hardware Applications, pp. 709-715.

R. A. Scantlebury, P. T. Wilkinson, and K. A. Bartlett, "The Design of a Message Switching Centre for a Digital Communication Network," Proceedings of the IFIP Congress 1968, Vol. 2 - Hardware Applications, pp. 723-733.

P. T. Wilkinson and R. A. Scantlebury, "The Control Functions in a Local Data Network," Proceedings of the IFIP Congress 1968, Vol. 2 - Hardware Applications, pp. 734-738.

D. W. Davies, "A Versatile Data Communication Network," Colloque International sur la Teleinformatique, Paris, March 1969, pp. 461-467.

D. L. A. Barber, D. V. Blake, and R. A. Scantlebury, "Implementation of the British Standard Interface for Data Exchange between Sources and Acceptors of Digital Data," Colloque International sur la Teleinformatique, Paris, March 1969, pp. 807-816.

K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Note on Reliable Full-Duplex Transmission over Half-Duplex Lines," CACM, Vol. 12, No. 5, May 1969, p. 260.

D. L. A. Barber, "Experience with the Use of the British Standard Interface in Computer Peripherals and Communication Systems," ACM Symposium on Data Communication, Pine Mountain, October 1969.

R. A. Scantlebury, "A Model for the Local Area of a Data Communication Network - Objectives and Hardware Organization," ACM Symposium on Data Communication, Pine Mountain, October 1969.

P. T. Wilkinson, " Model for the Local Area of a Data Communication Network - Software Organization," ACM Symposium on Data Communication, Pine Mountain, October 1969.

D. L. A. Barber and D. W. Davies, "The NPL Data Network," Proceedings of the Conference on Laboratory Automation, Novosibirsk, October 1970.

D. W. Davies, "The Control of Congestion in Packet Switching Networks," Proceedings of the Second ACM/IEEE Symposium on Problems in the Optimization of Data Communication Systems, October 1971, pp. 46-49; also in Transactions of IEEE, Vol. COM-20, No. 3, June 1972.

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," Proceedings of the Second ACM/IEEE Symposium on Problems in the Optimization of Data Communication Systems, October 1971, pp. 160-167; also in Transactions of IEEE, Vol. COM-20, No. 3, June 1972.

W. L. Price, "Simulation of Packet-Switching Networks Controlled on Isarithmic Principles," ACM Third Data Communications Symposium, Florida, November 1973, pp. 44-49.

R. A. Scantlebury and P. T. Wilkinson, "The National Physical Laboratory Data Communication Network," Proceedings of the ICCC Conference, Stockholm, August 1974, pp. 223-238.

D. W. Davies, "Packet Switching, Message Switching and Future Data Communication Networks," IFIP Congress, Stockholm, August 1974, pp. 147-150.

W. L. Price, "Simulation Studies of an Isarithmically Controlled Store and Forward Data Communication Network, Proceedings of the IFIP Congress, Stockholm, August 1974, pp. 151-154.

1.1.6 The Events Leading Directly to the ARPANET

The ARPA program plan under which ARPANET development was undertaken was prepared by Lawrence Roberts and approved by Dr. Rechtin, Director of ARPA, during Robert Taylor's tenure as Director of IPT. There is general agreement that Roberts is the individual with the most valid claim to being the "father" of the ARPANET. In a note dated July 7, 1977, Robert Taylor recalled the events leading to the undertaking of the ARPANET development up to the time when Lawrence Roberts joined ARPA.

> The ARPANET, like most good ideas, was there at least in tenuous form from the beginning, waiting for the right time to be invoked. In 1962-63 Licklider with characteristic tongue-in-cheek used the phrase "intergalactic network" to outline a plan for coupling academic computing expertise to the SDC Q-32 in order to aid in giving the SDC effort (which Lick inherited from DDR&E) the benefit of some of the experience gained by the universities. Lick was among the first to perceive the spirit of community created among the users of the first time-sharing systems. His successor, Ivan Sutherland, responded in 1965 to a UCLA request involving small-scale networking research at one site by offering ARPA support to the creation of a three-host campus network, but the internecine behavior of the three computer center directors at UCLA made the situation hopeless and the offer was withdrawn. Early in 1966, after a year in ARPA, it seemed to me that Lick and Ivan had paved the way for the next step. In pointing out the community phenomena created, in part, by the sharing of resources in one timesharing system, Lick made it easy to think about interconnecting the communities. The interconnection of interactive, on-line communities of people was also a dream of Doug Engelbart whose work I had supported from NASA in the early 60's and was continuing to support from ARPA.

From Ivan and the UCLA experiences I had to learn once again that the toughest problems were not the technical ones. If I could not get some ARPA funded participants involved in a commitment to a purpose higher than "Who is going to steal the next ten per cent of my memory cycles?", there would be no network. By June of 1966, Ivan had moved to Massachusetts to work at Lincoln Lab in the summer and then teach at Harvard. Ivan and I had encouraged Larry Roberts, then at Lincoln Lab, to pursue a limited networking experiment involving Lincoln's TX-2 and SDC's Q-32. With Ivan's leaving, IPTO was left with myself, a secretary, and a $20 million/year budget. The Director of ARPA then was Charles Herzfeld who, although a physicist, was extremely imaginative, approachable, and like his predecessor, Robert Sproull, always interested in discussing new ideas. (It was a pleasure to work for them!) I went to Charlie for help. I told him I wanted to initiate a networking project and I outlined its objectives:

1. To enable users at one IPTO sponsored site to gain interactive access to the programs/data at other IPTO sponsored sites.

2. To face deliberately the problems of interconnecting incompatible systems in order to someday enable the DOD to eliminate this argument for single-source systems.

3. To further encourage, through successful networking, the growth of specialized areas of computing knowledge at different sponsored sites - a trend which was already developing - thus enabling a user to tap various sources and hopefully reduce future duplication of software development.

In addition I gave Charlie some simple, hoped for, performance specifications, e.g., no user should have to wait more than one second for cross-country response else the illusion of being a local user, near the computer, be destroyed. (I underestimated what I now believe is one of the most important results from the ARPANET experiment: the power, and utility of message systems.) Charlie said "Great! Do it!" and gave the project more than ample budget to get off the ground. I told him I needed some help and described Larry Roberts as an ideal manager for the project. Charlie approved that request as well. ...

Next I had to convince Larry to head the project and convince some currently funded contractors to participate as users. It may seem surprising but I think the former was the most difficult. I had tried to interest Larry in ARPA earlier in 1966, when Ivan had announced his plans to leave, but Larry was non-committal. After getting Herzfeld's approval for the project I visited Larry several times over the next few months trying to convey the extent to which I needed help, but he argued (as many have whom ARPA has asked for help) that he did not want to make a break in his research career at that time. Furthermore, he reminded me that he was helping support IPTO's goals at Lincoln, which was certainly true. I returned to Washington after one of those visits with Larry and was about to give up. I began building a list of other candidates. Then one day something reminded me that ARPA supported 51% of Lincoln Laboratory's budget. I went to see Charlie, told him I was having difficulty getting help, and asked him if he would call the Director of Lincoln Lab and suggest to him that it would be in Lincoln Lab's best interest if they could spare Dr. Roberts for an important ARPA project. Charlie placed the call in my presence. Larry accepted the job* soon after, and moved to Virginia during the Christmas holidays of 1966. That accomplishment was probably the hardest and the most important of my ARPANET involvement.

Since 1963, a recurring question about some proposed or existing ARPA/IPTO project put to IPTO Directors by ARPA Management is "Why don't we rely on the computer industry to do that?"; or, occasionally more strongly, "We should not support that effort because ABC (read, "computer industry") will do it - if it's worth doing!" On the other hand, ARPA's support of research on networking, timesharing, and interactive computing in general, apparently denies the

----
* Roberts first went to ARPA as special assistant to the Deputy Director of ARPA. His responsibility was to provide technical advice to both IPT and other ARPA offices on computer matters. He later succeeded Taylor as director of the Information Processing Techniques office.

premise. So we ask ourselves if there is some singular, central theme which helps us distinguish the ARPA emphasis vis a vis computing research from that of the computer industry.

I believe the answer is a clear, unambivalent "yes," beautifully underscored by the explicit communications objectives of the ARPANET. The ARPA/IPTO theme was set by Licklider in his first tenure ('62-'64) and continues to be supported today: it is that the promise offered by the computer (read, "software, hardware, et al.") as a communication medium between people, dwarfs into relative insignificance the historical beginnings of the computer as an arithmetic engine. The computer industry, in the main, still thinks of the computer as an arithmetic engine. Their heritage is reflected even in current designs of their "communication systems." They have an economic and psychological commitment to the arithmetic engine model, and it can die only slowly; furthermore, it is a view that is still reinforced by most of the nation's computer science programs.

Even universities, or at least parts of them, are held in the grasp of the arithmetic engine concept. ... So while it may be later than we think - it is certainly going to take longer than we think; or put more constructively, the challenge taken on by IPTO roughly fifteen years ago has unveiled a problem-domain (opportunity-domain) which can keep IPTO productive for easily another fifteen years, even under the ARPA criterion that if it doesn't offer an order of magnitude improvement it should not be supported by ARPA.

1.2 The Events of 1967 and 1968

With Roberts on board at ARPA by the end of 1966, the ARPANET development effort got under way in earnest. The years 1967 and 1968 were spent promoting interest in the ARPANET project both within the government and with IPT contractors, deciding the fundamental structure of the network, writing a request for quotation, selecting a contractor, and other related activities.

Each year until recently ARPA held a meeting of the "principal investigators" from each of its university and other contractors. Called the "PI meetings", these meetings provided a forum at which contractors described their past year's results for the benefit of the other contractors, the IPT staff called the contractor's attention to problems of DoD concern, and everyone together attempted to unearth areas of technology which needed only ARPA fertilization in order to ripen to a point of usefulness. One of the topics of the PI meeting held at the University of Michigan in early 1967 was networks. At the meeting it was agreed that work could begin on the conventions to be used for exchanging messages between any pair of computers in the proposed network, and also on consideration of the kinds of communications lines and data sets to be used. In particular, it was decided that the inter-host communication "protocol" would include conventions for character and block transmission, error checking and retransmission, and computer and user identification. Frank Westervelt, then of the University of Michigan, was picked to write a position paper on these areas of communication, an _ad hoc_ "Communication Group" was selected from among the institutions represented, and a meeting of the group scheduled.

The plan considered throughout much of the Michigan meeting was to connect all of the computers by phone lines and data sets so as to allow any computer to establish communication with any other computer using a line-switching technique (i.e., calling it up on the telephone). A small program in each computer would interface to the data set and phone line and when this small program was given a message to another computer, the small interface program would perform a "message-switching" and transmission function, deciding how to actually reach the other computer and transmitting the message to it. Wes Clark, then of Washington University, is credited with proposing at some point in the meeting that a small computer be inserted between each participant's computer and the phone line. This concept was further discussed with the ARPA staff and one or two others in a now fabled taxi ride to the airport after the meeting, and the concept of the Interface Message Processor or IMP was reported in a memo from IPT to the PIs summarizing the Michigan meeting.

The IMP would perform the functions of dial-up, error checking, retransmission, routing, and verifications on behalf of the participants' computers (which we shall hereafter call hosts). Thus the IMPs plus the telephone lines and data sets would constitute a "message-switching network". The protocols which were to be established would define the communications formats between the IMPs. The interface between a host and an IMP would be a digital interface of a much simpler sort requiring no host consideration of error checking, retransmission, and routing. It was clearly noted that the major disadvantage of inserting the IMP was the cost of installation of another computer beside each host. The major advantage of inserting the IMPs was recognized to be the fact that a unified, straightforward network design could be made and implemented without undue consideration of the variations and constraints of the host computers. Further, as the network evolved, it would be much simpler to modify the network of IMPs than to modify all the host computers. Finally, the IMPs would relieve the hosts of the communications burdens they were initially scheduled to carry. It was also noticed that if necessary IMPs could be located at strategic connection points within the network to concentrate messages over cross-country phone lines, a network of IMPs was likely to be implemented faster than a network of directly connected hosts, and the network of IMPs provided a distinct network entity which would be useful in presenting the network publicly.

The initial draft of data communications protocols was drawn and circulated by Frank Westervelt and subjected to intensive review by a number of interested parties at a meeting at ARPA in mid-May, 1967. Following that meeting further work on the conventions was carried out by several individuals from various institutions and another draft, by Westervelt and Mills, was circulated after the summer of 1967. On the physical side this draft specified full duplex binary serial transmission at a minimum rate of 2400 bits per second in each direction, and further specified that each site should be capable of automatic answering of incoming calls and automatic dialing to originate outgoing calls. On the logical side the draft specified ASCII, use of 16-bit CRC checksums, a certain communications syntax, use of either a text or binary format (the latter using DLE-doubling for transparency), and certain control conventions. This draft also noted that the concept of using IMPs was gaining support.

A two-day meeting was scheduled in early October 1967, at ARPA, to discuss the protocol paper and specifications for the IMP (which had apparently become the accepted method). The meeting announcement also included a survey to be filled out by each of nineteen potential network locations providing gross guesses of total ARPANET traffic to other network locations by mid-1969 and percentage of the total traffic to each other location. By this time the proposed network was known as the ARPA Network, rather than the vague descriptions that had been used previously.

A variety of topics was discussed at the October meeting including message formating, message protocols, dynamic routing and message propagation, queuing, error control, measurements, and IMP-to-host communication. In a memo Roberts noted that the discussion at the meeting helped him formulate many plans and make several decisions. It was decided that 50 Kb communications lines would be used because of the vastly improved response time which could be obtained with these lines as opposed to the previously proposed 2.4 Kb lines. The 50 Kb lines were to be leased, eliminating the slow dialup procedure. The nature of the telephone tariffs available to the government made use of 50 Kb lines comparatively inexpensive. With each IMP normally permanently connected to three other IMPs via the leased 50 Kb lines, the IMPs were to use store-and-forward techniques to provide fast message handling. Each IMP was to accept messages of up to 8,000 bits from its host computer and to break this into 1,000-bit packets. Each packet was to be treated independently and routed on one of the three inter-IMP lines. When a packet arrived at an IMP, it was to be stored, error checked, and routed on to a further IMP. At its destination, packets would be held until an entire message could be assembled and then delivered to the destination host. With three lines per IMP, it was expected that approximately three store-and-forward stages would be necessary to get a message from any one of twenty locations to any other. It was also decided that a 16-bit CRC would be computed for each message, possibly by a special hardware attachment to the IMP. Also, an eight-bit binary format was to be used, requiring the IMP program or special hardware to insert DLE-characters before control characters and strip them out. Finally, it was suggested that the connection between the IMP and host use a single serial line between a shift register in the IMP and a shift register in the host, with the shift registers matched to the word sizes of the IMP and host. This serial line was to be capable of operating at 8 Mb. The need for control lines between the IMP and host was also noted.

The first half of 1968 was a time of intense study by many individuals of the methods to be used in the ARPANET. For example, in January 1968, Dr. Robert Kahn wrote a memo to the "ARPA Network Committee" in which he argued that the mean time between undetected errors in the network should be at least an order of magnitude larger than the debugging time for the network, and he concluded that an error detection scheme using a B.C.H. code generated by a 24-bit shift register be used. In February, Leonard Kleinrock of UCLA circulated three working papers about the ARPANET: a note by John Stehura on a suggested network control language which Kleinrock characterized as a first effort at solving a critical problem from the users' point of view; a note by Kleinrock presenting his thoughts on a simulation of the ARPANET; and a preliminary queuing analysis by Kleinrock for store-and-forward response time of the ARPANET. Other UCLA-generated documents on the ARPANET followed. In early March Kahn distributed another memo in which he urged that an effort be made to obtain data on the nature of errors on the 50 Kb communications lines, suggested that an important objective of the networking experiment be the evaluation of the interaction between communication facilities and terminal data processing equipment, and suggested that the initial network include at least one transcontinental link.

In parallel with the individual research that was going on, ARPA moved to pull together an IMP specification which could serve as the basis for a work statement for a competitive procurement to select a contractor to build the IMP network. At the end of 1967 ARPA initiated a small contract with the Stanford Research Institute for the development of specifications for the necessary communications system. Elmer Shapiro was to be the key person on this study. Published in a final version in December of 1968 as a 71-page SRI Report entitled "A Study of Computer Network Design Parameters", an early version of this report in early 1968 served as the first draft of the IMP specification. (Incidentally, this study was the first bit of explicitly paid effort going towards the ARPANET; all effort prior to that had been provided by contractors working extra hours on their existing ARPA contracts.) In February or March a memo written by Shapiro and revised by Kleinrock entitled "Functional Description of the IMP" was circulated. After the first draft by Shapiro, it is believed that Glenn Culler wrote a second draft, and Roberts and Wessler of ARPA wrote the final version of the IMP specification. In any case, by the first of March, 1968, IPT was able to report to the Director of ARPA that specifications for the IMP were essentially complete, and that they would be discussed at the upcoming PI meeting with the goal of issuing a Request for Quotation shortly thereafter. The network was discussed at the PI meeting and by June 1968, the ARPANET procurement officially started.

To begin an ARPA program, an ARPA Program Plan is needed. This plan must be signed by the director of the ARPA office involved, in this case IPT, and the director of ARPA. No one else in the DoD had to approve the plan to procure the ARPANET. In fact, it appears there was no serious opposition to the ARPANET within DoD at large, and within ARPA the most negative word was an admonition from the Deputy Director, Dr. Lukasik, to go slow and keep it small. In fact, as the incoming program manager for the ARPANET, Roberts had only to win over Director Rechtin (and his Deputy, Lukasik) and IPT Director Taylor. It has already been noted that Taylor was instrumental in getting Roberts to ARPA in the first place for the purpose of managing a network effort, and with Roberts then serving as Special

Assistant for Information Sciences reporting directly to the Deputy Director in the Director's office, it would have been surprising if the Director had not concurred with Roberts's plans. The ARPA Program Plan for the ARPANET, entitled "Resource Sharing Computer Networks", was submitted June 3, 1968, and approved by the Director June 21, 1968. The fiscal year 1968 ARPA budget included approximately $500,000 earmarked for the ARPANET effort.

The program plan for the ARPANET is an interesting document. The stated objectives of the program were to develop experience in interconnecting computers and to improve and increase computer research productivity through resource sharing. Technical needs in scientific and military environments were cited as justification for the program objectives. Relevant prior work was described. It was noted that the computer research centers supported or partially supported by IPT provided a unique testbed for computer networking experiments, as well as providing immediate benefits to the centers and valuable research results to the military. The network planning that had gone on was described, the need for a network information center was noted, and the network design was sketched. A five year schedule for network procurement, construction, operation, and transfer out of

ARPA was presented. (It is noteworthy that IPT initially had in mind eventual transfer of the operational network to a common carrier.) Finally, a several-million-dollar, several-year budget was stated.

The Defense Supply Service - Washington (DSS-W) agreed to be the procurement agent for ARPA. At the end of July the Request for Quotation for network IMPs was mailed to 140 potential bidders who had expressed interest in receiving it. Approximately 100 people from 51 companies attended a subsequent bidders' conference.* Twelve proposals were actually received by DSS-W comprising 6.6 edge-feet of paper and presenting an awesome evaluation task for IPT, which more normally awards contracts on a sole source basis. Attempting to evaluate the proposals "strictly by the book", an ARPA-appointed evaluation committee retired to Monterey, California, to carry out their task. ARPA was pleasantly surprised that several of the respondents believed that they could construct a network which performed as much as a factor of five better than the delay constraint given in the RFQ.

----
* In parallel with the competition for the IMP contract, ARPA held a competition among the common carriers to obtain the communication lines for the initial network. The contract was awarded to AT&T by the Defense Commercial Communications Office in early September, 1968.

Four bidders were rated within the zone of contention to receive the IMP contract, and supplementary technical briefings were requested from each of these bidders. Final negotiations were carried out with two finalists, and one was chosen in the week before Christmas, 1968. The contract was awarded and work began the second day of the New Year in 1969.

1.3 Key Aspects of the RFQ

The RFQ specified both administrative and technical aspects of the contractor selection and project implementation efforts. Bidders were requested to quote on a "cost plus fixed fee" (CPFF) basis. This is an arrangement the government commonly uses when contracting for unique developments, wherein the bidders first make their best estimate of the before-profit cost of completing the proposed effort. The bidder and the government then negotiate an appropriate fixed fee taking into account the before-fee estimated cost and the risk the bidder is taking in accepting the contract. If the contractor later overruns his estimated cost, the government has the option of stopping the effort before completion or paying the additional cost to complete; however, no additional fee is paid on the additional costs to complete. A CPFF arrangement makes sense when the government wants a risky development done and would have trouble getting a contractor to undertake the development if the contractor had to bear too much of the risk himself, as would be the case with standard fixed price bids. On the other hand, the fixed fee part of the arrangement reduces the risk that the contractor might artificially run up his costs, since he will gain no additional fee.

It was specified that responses to the RFQ would be evaluated on four criteria in addition to cost:

1. Understanding and depth of analysis of technical problems involved.

2. Availability of qualified, experienced personnel for assignment to software, hardware, and installation of the system.

3. Estimated functional performance and choice of hardware.

4. General quality, responsiveness, and corporate commitment to the network concept.

These four criteria were given weighted values of 30, 25, 25, and 20.

The RFQ had provision for a bidders conference and stated that there would be no other opportunity for bidders to discuss technical issues with the government. Bidders were asked to provide a system design for a nineteen-IMP network, but to price a four-IMP network. A thirteen-month performance period was requested to include design, construction of a prototype IMP, and implementation and installation of four operational IMPs. The four IMPs were to be installed nine months after start of the contract with the contractor supporting them in the field for three months after installation. The contractor was required to take full system responsibility, although subcontracting a portion of the work was a possibility.

The statement of work included in the RFQ was subtitled "Specifications of Interface Message Processors for the ARPA Computer Network" and specified many ARPANET characteristics which some people later thought represented technical decisions on the part of contractor. While there was room within the guidelines of the RFQ for the contractor to exercise his technical judgment in many matters, in fact ARPA knew what it wanted and the basic network design should be credited to ARPA and those who helped ARPA in 1967 and 1968. The document was a Request for Quotation for the implementation of ARPA's system design, not a Request for Proposal for an alternative system design. The following figure reproduces the Table of Contents of the Statement of Work or IMP specification section of the RFQ; it indicates the extent to which ARPA understood exactly what sort of network it had in mind.

The IMP specification clearly delineated the division of responsibilities among host sites, IMP contractor, and telephone company. Each individual host site was responsible for designing and implementing for its own convenience the hardware and software necessary to attach the host to the network, and the hardware and software to utilize other hosts on the network. The telephone company was to be responsible for providing necessary

Figure 2: Table of Contents from RFQ Statement of Work

circuits, data sets, and line conditioning equipment utilized by the network. The IMP contractor was to be responsible for providing necessary hardware and software to connect IMPs to each other using the circuits supplied by the telephone company and to connect IMPs to hosts, as well as providing hardware and software necessary to implement the procedures which allowed creation of a network of IMPs capable of forwarding messages from one host to another.

The functional description of an IMP specified the use of messages not longer than 8192 bits which would be broken into packets of not more than 1024 bits. Messages were limited in size to make them manageable for the hosts. Shorter packets were used to reduce the probability of transmission error with the attendant necessity for retransmission. It was noted that IMP provision of message and packet buffer space would permit speed conversions to take place, provide queuing space in the face of delays, and permit retransmission in the event of erroneous transmission. A routing algorithm was hypothesized which would take into account the connectivity of the network, IMP and line busyness, and message priority, and use this information to forward a packet to the next IMP on a path to the ultimate destination; periodic updates based on exchange of routing and loading information with other IMPs and hosts was also hypothesized. An IMP was to coordinate its activities with other IMPs and its hosts and perhaps other special hosts. IMPs were to take messages from a local host at the IMP's convenience, but to send messages to a local host at the host's convenience. The IMPs were to be able at selected times to measure selected network parameters and to trace the movements of selected messages through the network. The data resulting from these measurements and tracings was to be capable of transmission to a host, and the measurement activity was to be capable of initiation and termination by a host or another IMP. The IMPs were to detect and recover from various IMP, host, and line failures. In particular, it was to be possible to stop, start, examine, or reload IMPs from selected network hosts. Finally, it was thought that at each IMP site, it would be possible for special host-specific code to be provided by host site programmers, and thus it was desirable to protect the rest of the IMP from the portion that the host personnel could access and program.

There were two particularly interesting aspects of the host-to-IMP interface: 1) a standard host interface was to be specified rather than a different one for each host; 2) each bidder was to consider the cost of providing interfaces to multiple hosts per IMP, although only one host interface was required; and 3) sufficient program space was to be left to do host-specific character code conversion and repacking of binary messages.

The interface between an IMP and its telephone lines was required to have hardware to sense characters, detect control characters, calculate and check the 24-bit CRC, provide a real-time clock with 25 microseconds resolution, and provide fault and status information. Further, the IMP was to be optimized to handle three lines but be capable of handling six.

Several network performance characteristics were specified. The average message delay for a short message to go from a source IMP to a destination IMP was to be less than one-half second for a fully loaded network. The probability of lost messages and message errors was to be very low. Interestingly, network capacity was considered third in order of importance and was defined to be the maximum bit rate that can be input at every node and still have the message delay remain less than one-half second; a 25 Kb network capacity was hoped for. A network model was presented.

Host-to-host traffic flows were estimated and it was hypothesized that there would be a trimodal distribution of traffic type (high rate and short length, medium rate and medium length, and low rate and long length).

In addition to considering the option of multiple hosts connected to an IMP, the bidder was also asked to consider the provision of memory protection to facilitate simultaneous IMP operation and checkout of new software, and to consider what additional hardware and software would be necessary for an IMP to provide a terminal concentration capability for its host or for the network (i.e., no host, just terminals).

1.4 Chronological Development, 1969 to 1975

Within a year after the award of the IMP contract, the first IMPs were installed in the field. Hosts were connected to these first IMPs and a series of network measurements was undertaken. Host software was written, hosts began to communicate with each other, more IMPs and hosts were added, and gradually the ARPANET became an operating entity. After six years of development and operation the network was no longer well suited to management by an agency with the charter to sponsor advanced research and development, and thus network management was transferred to the Defense Communications Agency. In the following subsections we describe the happenings of the years 1969 to 1975.

1.4.1 The Groups and the Key People

The ARPANET development was a joint effort of many individuals and institutions, all responsive to ARPA's direction. Before we describe the ARPANET development further, it is best to list the principal "players" and briefly describe their areas of responsibility.

1.4.1.1 Management and Administration of the Network: ARPA IPT, DSS-W, RML, and DECCO

Naturally, ARPA IPT played a major role in the development of the network, and Lawrence Roberts was the most important player of them all. Roberts promoted the network, made certain design decisions, led the evaluation team which selected the IMP contractor and selected other involved contractors, and was the program manager for the network program.

IPT usually does little day-to-day management of its contractors. Especially with its research contracts, IPT would not be producing faster results with such management -- research must progress at its own pace. IPT has generally adopted a mode of management which entails finding highly motivated, highly skilled contractors, giving them a task, and allowing them to proceed by themselves. Of course, IPT holds periodic reviews, and in some cases the scientists who are serving as IPT Director or program managers pitch in and themselves participate in the contractors' research. If a contractor does not produce good results in this environment of low-profile management, the contractor is dropped. If the contractor does produce good results, IPT continues to fund him. Remember that IPT generally awards contracts on a sole source basis, which makes it possible to drop or keep contractors. The IPT style of management has also been dictated by the exceedingly small staff which has been allowed for them for many years, which has severely constrained the time that could be spent supervising each contractor. Within these constraints, the IPT management style has been reasonably successful.

The ARPANET program was managed by IPT in the same manner as most of their research programs, despite the fact that the contract for the IMP development resulted from a competitive procurement and the network evolved into an operational entity. Thus, while IPT set policy for the network, made decisions about who would join the network, adjudicated squabbles between the several involved contractors, and so forth, IPT did not run the network day by day. BBN provided day-by-day operation and maintenance of the network, and BBN and the other contractors involved carried out much of the day-by-day business of the network among themselves without need for constant IPT supervision.

Initially IPT itself took care of ordering telephone lines, filling out forms which were sent to DECCO for procurement from and execution by the various telephone companies. Also, IPT initially did its own technical monitoring of the various contractors associated with the ARPANET; DSS-W was used as the procurement agency for the contractors. Eventually, attempting to rid itself of the routine, tedious aspects of ordering communications lines and providing technical monitoring of contractors, IPT shifted the DSS-W functions, routine dealings with DECCO, and some contractor technical monitoring to the Range Measurements Laboratory at Patrick Air Force Base in Florida; in addition to having a procurement capability, RML also had a technical support capability.

There were several other members of the IPT staff who have been prominent in the management of the ARPANET besides Roberts. In 1968, 1969, and part of 1970 Barry Wessler helped Roberts with the technical side of ARPANET activities. Until the shift of responsibility to RML, first Col. Bruce Dolan and later Major Dave Carlstrom took care of ordering phone lines. Eventually Mr. Stephen Crocker became ARPANET program manager. At about the time Licklider returned for his second term as IPT director, Dr. Craig Fields took over as ARPANET program manager and Fields was followed by Mr. Stephen Walker (with a brief interregnum by Mr. William Carlson). Walker served as program manager until the transfer of the network from ARPA, and he continues today to be the ARPA point of contact for ARPANET concerns. Of course, both IPT directors after Roberts, Licklider and Russell, had strong influence on IPT's ARPANET activities. Throughout IPT's involvement with the ARPANET, Mr. Al Blue has been involved in the financial aspects of the network's development and operation (Blue was acting IPT director between Roberts and Licklider and during that period was deeply involved in other aspects of the network's management).

As mentioned above, IPT often played a strong technical role in the ARPANET, and members of the IPT staff wrote a number of papers describing the ARPANET activity. Several of these papers are listed in the following bibliography.

Bibliography

T. Merill and L. G. Roberts, "Toward a Cooperative Network of Time-Shared Computers," Proceedings FJCC 1966, p.425-431.

L. G. Roberts, "Multiple Computer Networks and Intercomputer Communication," ACM Symposium on Operating System Principles, October 1967, 7p.

L. G. Roberts and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," Proceedings SJCC 1970, p.543-549.

L. G. Roberts, "A Forward Look," Signal, August 1971, p.77-81.

L. G. Roberts, "ARPA Network Implications," EDUCOM Bulletin, Fall 1971, p.4-8.

L. G. Roberts, "ARPANET: Current Status, Future Plans," EDUCOM Spring Conference, April 1972, p.7-12.

L. G. Roberts, "Network Rationale: A 5-Year Reevaluation," Proceedings COMPCON 1973, February 1973, p.3-6.

L. G. Roberts, "Dynamic Allocation of Satellite Capacity through Packet Reservation," Proceedings NCC&E 1973, p.711-716.

L. G. Roberts, "The ARPA Net," in Computer-Communication Networks, N. Abramson and F. Kuo (Eds.), Prentice-Hall, 1973.

1.4.1.2 The Network Analysis Corporation

Led by Dr. Howard Frank, the Network Analysis Corporation (NAC) of Glen Cove, Long Island, was put under contract by ARPA to specify the topological design of the ARPANET and to analyze its cost, performance, and reliability characteristics.

In the process of evaluating any of the parameters of a particular network design, such as cost, reliability, delay, or throughput, it is necessary to simulate the flow of traffic through the proposed network. Then, the design may be altered slightly to improve one of these measures. In this procedure, it is important to have a facility for specifying the routes on which traffic will flow in the network, and the procedure must not be too complex since it must be repeated so often in the iterative design process. NAC has developed some very efficient methods for incremental changes to a shortest-path routing algorithm as the network topology is changed. Further, they have discovered a faster shortest-path algorithm than was previously available, taking advantage of the low connectivities usually present in most practical communications networks.

The following bibliography lists much of the published work by NAC related to the ARPANET.

Bibliography*

NAC First Semiannual Technical Report for the Project, "Analysis and Optimization of Store-and-Forward Computer Networks," June 1970, 62p.

NAC Second Semiannual Technical Report for the Project, "Analysis and Optimization of Store-and-Forward Computer Networks," January 1971, 96p.

NAC Third Semiannual Technical Report for the Project, "Analysis and Optimization of Store-and-Forward Computer Networks," June 1971, 109p.

NAC Fourth Semiannual Technical Report for the Project, "Analysis and Optimization of Store-and-Forward Computer Networks," December 1971, 123p.

NAC Fifth Semiannual Technical Report for the Project, "Analysis and Optimization of Store-and-Forward Computer Networks," June 1972, 92p.

NAC Final Technical Report for the Project, "Analysis and Optimization of Store-and-Forward Computer Networks," October 1972, 120p.

H. Frank, I. T. Frisch, and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," Proceedings SJCC 1970, p.581-585.

H. Frank, I. T. Frisch, R. Van Slyke, and W. S. Chou, "Optimal Design of Centralized Computer Networks," Networks, Vol. 1, no. 1, 1971, p.43-58.

H. Frank and W. Chou, "Routing in Computer Networks," Networks, Vol. 1, No. 2, 1971, p.99-112.

H. Frank, "Computer Network Design," IEEE International Convention Record, March 1971, p.220-221.

----
*I have asked Howie Frank to suggest additions and deletions from this bibliography -- DCW.

H. Frank and W. Chou, "Throughput in Computer-Communications Networks," International State of the Art Report no. 6; Computer Networks, Infotech Information Ltd, Maidenhead, Berkshire, England, 1971, p.493-512.

R. Van Slyke and H. Frank, "Network Reliability Analysis: Part I," Networks, Vol. 1, No. 3, 1971, p.279-290.

R. Van Slyke and H. Frank, "Reliability of Computer-Communication Networks," Proceedings of the 1971 Winter ACM Simulation Conference, December 1971, p.71-82.

H. Frank and I. T. Frisch, "The Design of Large-Scale Networks," Proceedings of the IEEE, Vol. 60, No. 1, January 1972, p.6-11.

H. Frank, R. E. Kahn, and L. Kleinrock, "Computer Communication Network Design - Experience with Theory and Practice," Proceedings SJCC 1972, p.255-270; also in Networks, Vol. 2, No. 2, 1972, p135-166.

W. Chou and H. Frank, "Routing Strategies for Computer Network Design," Proceedings of the Microwave Research Institute International Symposium on Computer-Communications Networks and Teletraffic, April 1972, p.301-309.

A. Kershenbaum and R. Van Slyke, "Computing Minimum Spanning Trees Efficiently," Proceedings of the ACM Annual Conference, August 1972, p.518-527.

H. Frank and W. Chou, "Topological Optimization of Computer Networks," Proceedings of the IEEE, Vol. 60, No. 11, November 1972, p.1385-1397.

H. Frank, "Optimal Design of Computer Networks," in Computer Networks, R. Rustin (Ed.), Prentice-Hall, 1972, p.167-183.

M. Gerla, "Deterministic and Adaptive Routing Policies in Packet-Switched Computer Networks," ACM Third Data Communications Symposium, Florida, November 1973, p.23-28.

W. Chou, H. Frank, R. Van Slyke, "Simulation of Centralized Computer Communications Systems," ACM Third Data Communications Symposium, Florida, November 1973, p.121-130.

W. Chou and A. Kershenbaum, "A Unified Algorithm for Designing Multidrop Teleprocessing Networks," ACM Third Data Communications Symposium, Florida, November 1973, p.148-156.

H. Frank, "Providing Reliable Networks with Unreliable Components," ACM Third Data Communications Symposium, Florida, November 1973, p.161-164.

I. Gitman, R. M. Van Slyke, and H. Frank, "On Splitting Random Accessed Broadcast Communication Channels," Proceedings of the Seventh Hawaii International Conference on System Sciences, January 1974, 5p.

1.4.1.3 The Telephone Companies

Building a nationwide communications network means doing business with a number of telephone companies; further, if the network is to have overseas or foreign components, one must also deal with various international carriers and national Post Telephone Telegraphs. As mentioned in a previous section, there exists within the U.S. government an agency called DECCO, which other parts of the government can ask to procure communications services. ARPA used DECCO to procure most, if not all, of the domestic circuits for the ARPANET. In some special cases, for example, the overseas lines to Hawaii and Europe, ARPA dealt directly with the relevant communications authorities.

The procedure for using DECCO to procure communications service is to send them a Telecommunications Service Request or TSR. For instance, to obtain a circuit from UCLA to RAND, ARPA would send a TSR requesting the necessary circuit, and DECCO would then negotiate with the relevant telephone companies and obtain the specified service at the best price. In the case of a circuit from UCLA to RAND, for example, most likely the service would be procured from General Telephone, the dominant telephone company in the Los Angeles area.

In the example just given, a requested circuit fell completely within the jurisdiction of a single telephone company. To handle instances when the requested service spanned the jurisdictions of more than one telephone company, the Bell System set up another company, called AT&T Long Lines; and in the case of many of the ARPANET circuits, the company from which DECCO procured the service was Long Lines, which in turn procured the components necessary to make up the requested service from the regional telephone companies.

For the first several years of the ARPANET development, the Long Lines customer representative to ARPA was Mr. Ken Stanley. A customer representative's job is to make the customer aware of the kinds of service available and to keep him happy with the service he receives. Fortunately for the ARPANET, the Bell System understood that a customer building a nationwide network needed the assistance of some central individual with a broad grasp of the requirements rather than having to rely on the usual miscellaneous dealings with local telephone companies. Thus Stanley, who from his position in Long Lines was already in contact with his counterparts in the many regional telephone companies, was permitted to use his network of contacts to provide informal coordination of the entire Bell System service to the ARPANET.

For instance, installing a telephone circuit between two IMPs requires that the IMPs and the telephone companies at both ends all be ready simultaneously. It is useless for the IMP supplier and one of the telephone companies to strain to make a scheduled date if the other telephone company cannot make that date. Similarly, it is useless for the telephone companies to strain to make a date if the IMP supplier cannot. Stanley took it upon himself to coordinate all such interdependent events. By virtue of the Bell System's willingness to provide this critical coordination, an extremely smooth and efficient relationship has been built up between the IMP suppliers, ARPA, and the member companies of the Bell System. For a network of the size and complexity of the ARPANET, there is surprisingly little trouble with the procurement and operation of the telephone circuits.

1.4.1.4 Bolt Beranek and Newman Inc.

The contract to construct the IMP for the ARPANET was awarded to Bolt Beranek and Newman Inc. (BBN) of Cambridge, Massachusetts, where it was carried out in a group under the leadership of Mr. Frank Heart. Once the first IMPs were installed, BBN continued to play a central role in the evolution of the network, operating it and maintaining it as well as doing the development necessary for several major enhancements of its capability.

In addition to Frank Heart, a number of other individuals at BBN have been involved with the development of the ARPANET. The names of many of these individuals may be found as authors of the papers on the ARPANET which have come out of BBN. However, one individual, Robert Kahn, stands out. After several years of participation in the group working on the network at BBN, he moved to the IPT office at ARPA from which he has probably done more to promote and support the continued advance of packet-switching technology than any other individual with the possible exception of Roberts himself.

A bibliography of ARPANET-related reports and papers written by members of the BBN staff is included as Appendix A.

1.4.1.5 The Network Information Center

The accessibility of distributed resources carries with it the need for an information service (either centralized or distributed) that enables users to learn about those resources. This was recognized at the PI meeting in Michigan in the spring of 1967. At the time, Doug Engelbart and his group at the Stanford Research Institute were already involved in research and development to provide a computer-based facility to augment human interaction. Thus, it was decided that Stanford Research Institute would be a suitable place for a "Network Information Center" (NIC) to be established for the ARPANET. With the beginning of implementation of the network in 1969, construction also began on the NIC at SRI.

The NIC provided several services. It maintained a list of network participants and distribution lists for various special interest groups within the network community. An archive of various document series was maintained. Documents could be sent to the NIC with instructions for duplication and distribution to the membership of one or more of the special interest groups. A highly structured data base construction, manipulation, and display system (called NLS) was made available on-line for use over the network. A list was kept of the resources available on hosts throughout the network. The various ARPANET protocol specifications were maintained online at the NIC.

The NIC has had a hand in the production or distribution of hundreds and hundreds of documents related to the ARPANET. A few of these documents describe the activities of the NIC itself or are otherwise of special interest within the ARPANET; a bibliography of these follows.

Bibliography

D. C. Engelbart, "Network Information Center and Computer Augmented Team Interaction", Stanford Research Institute, Augmentation Research Center, January 30, 1971, 105p.

D. C. Engelbart, "Online Team Environment", Stanford Research Institute, Augmentation Research Center, June 2, 1972, 272p.

D. C. Engelbart, "Coordinated Information Service for a Discipline or Mission-oriented Community", _Timesharing: Past, Present, and Future, Second Annual Computer Communications Conference_, 1973, pp. 2.1-2.4.

Current Catalog of the NIC Collection, NIC No. 5145.

Current Directory of Network Participants, NIC No. 5150.

Network Resource Handbook, NIC No. 6740.

Current Network Protocols, NIC No. 7104.

1.4.1.6 The Network Measurement Center

One of the influential early studies of communications networks was performed by Leonard Kleinrock and is reported in his 1962 Ph.D. thesis at MIT, Communication Nets: Stochastic Message Flow and Delay (reprinted by Dover Publications, New York, 1964). This study proved convincingly that message delays in a large store-and-forward network can be made very low, and thus put to rest one of the early and persistent objections to networking. Kleinrock considered several aspects of the operation of communications networks and their underlying algorithms in developing a precise model for networks.

Kleinrock continued his work as a member of the faculty of UCLA and it was quite natural for him to participate in the study and specification phase of the ARPANET development in the two years that preceded 1970. By the time the IMP RFQ was released, it was clear to ARPA that Kleinrock and his group at UCLA were the correct people to set up and staff a Network Measurement Center (NMC) for the ARPANET. Plans were therefore laid to have UCLA be the site of the first IMP installation, to allow early connection of an SDS SIGMA 7 host which was to be used to support the NMC tasks. The NMC had the responsibility for much of the analysis and simulation of the ARPANET performance, as well as direct measurements based on statistics gathered by the IMP program. While Kleinrock himself was the guiding force at the NMC, over the years he had a series of students or staff members who supervised the day-to-day measurement work, including Gerald Cole, Vinton Cerf, Holger Opderbeck, and William Naylor (each obtained a UCLA Ph.D., although not necessarily for their NMC work).

There has been a series of doctoral dissertations written by students at the UCLA School of Engineering and related to the work of the NMC. A list of of these theses, as well as other relevant publications by the faculty and students associated with the NMC, is included in the following bibliography.

Bibliography*

L. Kleinrock, "Communication Nets: Stochastic Message Flow and Delay," Dover Publications, New York, 1964, 209p.

L. Kleinrock, "Models for Computer Networks," Proceedings of the International Communications Conference, June 1969, p.21.9-21.16.

L. Kleinrock, "Comparison of Solution Methods for Computer Network Models," Proceedings of the Computers and Communications Conference, October 1969.

----
*I have asked Len Kleinrock to suggest additions and deletions from this bibliography -- DCW.

L. Kleinrock, "Analytic and Simulation Methods in Computer Network Design," Proceedings SJCC 1970, p.569-579.

L. Kleinrock and G. L. Fultz, "Adaptive Routing Techniques for Store-and-Forward Computer Communication Networks," International State of the Art Report No. 6; Computer Networks, Infotech Information Ltd. Maidenhead, Berkshire, England, 1971, p.541-562; also in Proceedings of the International Conference on Communications, Montreal, Quebec, Canada, June 1971, p.39.1-39.8.

G. Cole, "Performance Measurements on the ARPA Computer Network," Proceedings of the Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications Systems, October 1971, p.39-45; also in IEEE Trans. on Communications , Vol COM-20, No 3, June 1972, p630-636.

G. D. Cole and L. Kleinrock, "An Analysis of the Separation between Packets in a Store-and-Forward Network," Proceedings of the Microwave Research Institute International Symposium on Computer-Communications Networks and Teletraffic, April 1972, p.1-3.

V. G. Cerf and W. E. Naylor, "Storage Considerations in Store-and-Forward Message Switching," Paper presented at WESCON Conference, September, 1972, 8p.

V. G. Cerf and W. E. Naylor, "Selected ARPA Network Measurement Experiments," Proceedings COMPCON 1972, September 1972.

L. Kleinrock, "Computer Networks," in Computer Science, A. F. Cardenas et. al. (Eds.), John Wiley and Sons, Inc. New York, 1972, p.241-284.

L. Kleinrock, "Performance Models and Measurement of the ARPA Computer Network," Proceedings of the International Symposium on the Design and Application of Interactive Computer Systems, Brunel University, Uxbridge, England, May 1972, 30p.

L. Kleinrock, "Survey of Analytical Methods in Queueing Networks," in Computer Networks, R. Rustin (Ed.), Prentice-Hall, 1972, p.185-205.

D. Cantor and M. Gerla, "The Optimal Routing of Messages in a Computer Network Via Mathematical Programming," Proceedings COMPCON 72, September 1972, p.167-170.

L. Fratta, M. Gerla, and L. Kleinrock, "The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design," Networks, Vol. 3, No. 2, 1973, p.97-134.

L. Kleinrock and S. S. Lam, "Packet-switching in a Slotted Satellite Channel," Proceedings NCC+E 1973, p.703-710.

L. Kleinrock, "Queueing Systems: Theory and Application," Wiley Interscience, New York, 1973.

L. Kleinrock and S. Lam, "On Stability of Packet Switching in a Random Multi-Access Broadcast Channel," Proceedings of the Seventh Hawaii International Conference on System Sciences, January 1974, 4p.

D. G. Cantor and M. Gerla, "Capacity Allocation in Distributed Computer Networks," Proceedings of the Seventh Hawaii International Conference on System Sciences, January 1974, p. 115-117.

G. Cole, "Computer Networks Measurements: Techniques and Experiments," PhD Thesis, UCLA-ENG-7165, Computer Science Department, School of Engineering and Applied Science, UCLA, October 1971, 350p.

G. L. Fultz, "Adaptive Routing Techniques for Message Switching Computer-Communications Networks," PhD Thesis, UCLA-ENG-7252, Computer Science Department, School of Engineering and Applied Science, UCLA, July 1972, 418p.

M. Gerla, "The Design of Store-and-Forward (S/F) Networks for Computer Communications," PhD Thesis, UCLA-ENG-7319, Computer Science Department, School of Engineering and Applied Science, UCLA, January 1973, 300p.

L. Kleinrock, "Computer Network Research," UCLA February 1970, 26p.

L. Kleinrock, "Computer Network Research," UCLA August 1970, 109p.

L. Kleinrock, "Computer Network Research," UCLA June 1971, 130p.

L. Kleinrock, "Computer Network Research," UCLA December 1971, 115p.

L. Kleinrock, "Computer Network Research," UCLA June 1972, 113p.

L. Kleinrock, "Computer Network Research," UCLA December 1972, 70p.

L. Kleinrock, "Computer Network Research," UCLA June 1973, 64p.

1.4.1.7 The Network Working Group and Host Involvement

The initial design of the ARPANET as contained in the RFQ went some way toward specifying the protocols for inter-IMP communications and for IMP-to-host communication. Less explicit attention was given to host-to-host communication, this area being left for host sites to work out among themselves.

To provide the hosts with a little impetus to work on the host-to-host problems, ARPA assigned Elmer Shapiro of SRI "to make something happen", a typically vague ARPA assignment. Shapiro called a meeting in the summer of 1968 which was attended by programmers from several of the first to hosts be connected to the network. Individuals who were present have said that it was clear from the meeting that at that time, no one had even any very clear notions of what the fundamental host-to-host protocol issues might be.

After the Santa Barbara meeting, S. Crocker, S. Carr, and J. Rulifson met again in the summer and fall of 1968 to continue discussion of host-to-host protocol issues. Their early thinking was at a very high level, e.g., the feasibility of creating a portable front-end package which could be written once and moved to all network hosts; a host desiring to send data to another host would first send a data description to the receiving host which instructed the front-end package at the receiving host how to interpret data coming from the sending host. On Valentine's Day, 1969, the first meeting of host representatives and representatives from the NMC and NAC, along with the IMP contractor, was held at BBN in the middle of an enormous snow storm.

The several individuals considering host-to-host protocol issues had never actually received an explicit charter from ARPA to do this work, and they worried that someone might think they were acting out of place. For this reason, in April 1969, they established a series of working notes called Request for Comments (RFCs), which could be circulated to let others know what they were doing and to obtain the reactions and involvement of other interested parties. They called themselves the Network Working Group (NWG).

The NWG eventually grew quite large, with representatives from almost every host site in the network participating, and mountains of paper was circulated describing and commenting on various protocols. There were also occasional mass meetings. From about the time it was decided that he would go to ARPA until near the time he left ARPA, Stephen Crocker served as chairman of the NWG. By the beginning of 1972 the NWG had grown too large, but much of its work was done -- large numbers of hosts were communicating over the network. From this point onward, meetings were limited to those of an executive protocol committee which met to discuss general protocol issues and provide guidance for Crocker, and to those of various subcommittees, e.g., the group interested in the Remote Job Entry protocol. Even after the big meetings stopped, most participant working notes were circulated to most other participants in the network.

Gradually the activities of the NWG began to diminish. Many of the host site personnel who had originally been active moved on to other tasks, and new users joining the network tended to use the defined protocols rather than becoming involved in their specification. As Crocker's time for the NWG group became increasingly limited, he appointed Alex McKenzie and Jon Postel to serve jointly in his place. McKenzie and Postel interpreted their task to be one of codification and coordination primarily, and after a few more spurts of activity the protocol definition process settled for the most part into the status of a maintenance effort. Today when one of the once-central figures in the host protocol design process circulates an RFC among his remaining contemporaries, it may be more for old times' sake than for any other reason.

Further activities of the Network Working Group will be described in the section below on the development ik host host-to-host protocol.

1.4.2 Initial Subnet Design

BBN's proposal for the IMP was for the most part compliant with the requirements of the RFQ. In some important respects, however, the BBN IMP design diverged from those requirements or added constraints that were not in the RFQ.

The BBN design took the idea of inserting a communications processor between the host and the network to a logical extreme: it specified that the IMP be used only for communications functions, and that there be maximum logical separation between IMP and host. Thus, the design did not provide for IMP repacking of binary messages on behalf of a host or for doing character conversion for a host. Further, the design precluded all host programming of the IMP and any control of the IMP by regular hosts. This portion of the IMP design undoubtedly was correct and contributed to the rapid development of the subnetwork of IMPs.

The initial subnet design also specified minimal control messages between an IMP and its hosts, many fewer than envisioned in the RFP. As the network later developed, it proved useful to add additional such control messages.

The IMP design called for a hardened IMP, which would be robust in the face of physical and electrical abuse. While statistics have shown that hardening did help, it was eventually decided not be be worth the added price of the hardware.

The capability to loop all interfaces was included in the design. This has proved to be of enormous operational importance.

A decision was made to reload IMPs initially from paper tape, and each IMP was provided with a paper tape reader. It was always intended that this procedure would sooner or later be replaced by loading through the network, and it was. The use of the paper tape reader at each IMP was probably a useful simplication in the beginning.

Program debugging was also specified to take place from a local terminal, in the interests of simplicity, with the knowledge that cross-network debugging could be added later. A cross-network debugging capability was added even before the first IMP was delivered.

A delivery schedule of one IMP per month in months eight through eleven after contract start, instead of all four at month nine, was assumed.

In keeping with the strict independence of host and IMP, the IMP was to gather statistics routinely and transmit them periodically to specified hosts rather than permitting hosts to control the IMPs' statistics-taking capabilities. This proved satisfactory.

The initial IMP design was responsive to the RFQ in one particular which was changed at the first meeting between BBN, ARPA, and host representatives. RFQ had called for one host per IMP and six circuits per IMP, although it also asked that more hosts per IMP be an option. Almost immediately it became clear that many host sites had more than one host to connect to a single IMP. Thus before the first IMP was delivered, the design was changed to permit two, three, or four hosts on an IMP along with five, four, or three lines. This was a simple change to implement.

In the area of the host-to-IMP protocol, the initial IMP design specified this protocol as required. Unfortunately, some aspects of the host-to-IMP protocol had significant detrimental effect on the design and performance of the other protocols. Particularly unfortunate have been the acknowledgment system, the retransmission system, and the message identification system initially suggested by the host-to-IMP protocol.

It is crucial for the IMPs to limit the rate of flow of host traffic into the net to the rate at which that traffic is being taken out, in order to prevent subnetwork congestion. The IMPs' first attempt, which was insufficient, took the form of suggesting that each message in a conversation should be held by the sending host until an end-to-end acknowledgment for the previous message was received. This suggestion was adopted as part of host-to-host protocol upon which all the higher standard protocols are based. As a consequence, the bandwidth of a single host-to-host protocol connection is severely limited; given the ARPANET response time using 50 Kbs lines, waiting for a destination-to-source acknowledgment between messages typically limits connection bandwidth to about 10 Kbs, in contrast to the 40 Kbs possible with a constant stream of messages.

It was originally thought that the ARPANET would lose a message so seldom that there was no point in hosts ever bothering with message retransmission. Unfortunately, resolving various possible lockups has required the subnetwork to discard a message occasionally, and the topology of the network has evolved into long series of machines and lines that increase the probability of involuntary message loss. However, the host-to-host protocol followed the initial thought and did not provide for message retransmission. Given the realities of the probability of message loss in the network and given the host-to-host protocol, which is inordinately sensitive to any abnormality, the host-to-host protocol (and protocols based on it) has proved quite unreliable, and the original host-to-IMP protocol must be held partly responsible.

As part of the host-to-IMP protocol end-to-end acknowledgement system described above, the host-to-IMP protocol specified an 8-bit message identification number and suggested that all messages in a single conversation carry this same identification number; in fact, messages with different identification numbers were not guaranteed to be delivered in the order sent. Eight bits is probably insufficient to identify uniquely (for the purpose of possibly required retransmissions) outstanding messages when successive messages in a conversation are sent without waiting for an end-to-end acknowledgement. Thus the small 8-bit message identifier prevented reliable high-bandwidth connections.

The IMP/host protocol has been changed so that it is no longer necessary to wait for the end-to-end acknowledgment; message order is now preserved except for priority considerations; cases requiring message retransmission are unambiguously reported to the sending host; and the message identifier has been expanded to a sufficient size. Unfortunately, there is great inertia in the host-to-host protocol due to the large number of implementations that would have to be changed. The host-to-host protocol is still incapable of simultaneous high performance and perfect reliability.

1.4.3 Subnet Development

The first four IMPs were developed and installed on schedule by the end of 1969. No sooner were these IMPs in the field than it became clear that some provision was needed to connect hosts relatively distant from an IMP (i.e., up to 2000 feet instead of the expected 50 feet). Thus in early 1970 a "distant" IMP/host interface was developed. Augmented simply by heftier line drivers, these distant interfaces made clear for the first time the fallacy in the assumption that had been made that no error control was needed on the host/IMP interface because there would be no errors on such a local connection.

By midyear of 1970, a series of network performance tests were being carried out. These uncovered some flaws which were quickly corrected, and some problems which looked more worrisome. Also by midyear, a rudimentary version of the network control center was established at BBN.

As the year wore on, sites continued to be added to the network, the IMP program continued to be improved, NCC development continued, the first 230.4 Kb circuit was tested between two IMPs, and design for a version of the IMP able to support direct terminal connection was begun. The latter was called the Terminal IMP.

By early in 1971 major problems with the IMP flow control and storage allocation techniques were found, and design began on improved algorithms. It is interesting that once these problems were found, and they were serious enough to completely halt network operation, the network continued to give adequate service for many, many months while improvements were designed and implemented. The hosts were simply asked to not use the network in the way that caused the subnetwork problems, and the hosts did as they were asked.

About three-quarters of the way through 1971 the first two TIPs were delivered, providing ARPANET access for the first time to users without their own hosts or access to terminals on some other organization's host.

By the beginning of 1972 it was recognized that even the distant version of the IMP/host interface was not sufficient, and design for a IMP/host interface for use over communications circuits was begun. The evolution of the IMP/host interface is worth a little additional comment. The initial bit-serial, asynchronous, non-error-controlled IMP/host interface was essentially specified in the RFQ, in an effort to simplify network connection for the hosts. This non-standard interface may have been of some benefit in simplifying the host connection. However, its greatest virtue was the separation it put between the IMP and the host. The IMP and host did not have to worry about each other's word size, and they did not have to worry about each other's timing constraints. It seems likely that having to worry about these issues would have delayed network operation. However, this interface also resulted in a hodge-podge of interface variations, each designed for more distant operation than its predecessors, and none except the first was very elegant. For any new network, which need not fear the now proven packet-switching technology, it would clearly be better to use an industry standard communications interface, e.g., HDLC, for every IMP/host connection.

In the first half of 1972 the TIP's capability was expanded to support TIP-to-TIP magnetic tape transfers. While this option was successfully used between two network sites, it was never very elegant, and almost immediately had to be partially redone. Also, a massive change in the IMP software was undertaken to correct the previously discovered flow control and storage allocation problems. In the second half of the year, the new version of the IMP program was released in many small increments, and the design of a new, ten times more powerful IMP was begun.

The beginning of 1973 brought the first satellite link in the network, from California to Hawaii. Also, with network traffic rapidly increasing, a number of subnet reliability problems developed which had to be corrected. By midyear, a pair of TIPs had been shipped to Europe, for use in Norway and London. These brought plenty of operational troubles. For the first time, circuits had to be obtained from a foreign PTT, the circuits were relatively slow at 9.6 Kb, and like Hawaii, these TIPs were on a long spur off the network rather than being connected doubly as IMPs typically are. As 1973 continued, nodes continued to be delivered, but there began to be a low level of switching of node locations, to optimize the use of various IMP configurations and as sites came on and went off the network. Certain improvements were also made to correct problems with the routing algorithm. As 1973 ended the first very distant hosts were connected to the network.

In 1974 there were major efforts to make the network more operational. Subnetwork reliability was improved as was TIP-to-host communication reliability. Methods for providing TIP access control and accounting and partitioning of logical subnetworks of hosts were developed. Methods were developed to selectively reload sections of IMP memory.

In 1975 network development slowed up and the network took on more and more of an operational appearance. Major network developments in 1975 included delivery of the first Pluribus IMP, modification of the IMP and TIP software to support more than sixty-three IMPs in the network and attachment of the first two Satellite IMPs to the edge of the network. By the end of 1975 the network was under DCA management.

Looking back, the subnet development between 1969 and 1975 appears relatively smooth, although there were many times during that period when those intimately involved felt they were trying to solve one crisis or another. The network grew slowly enough, and the basic technology and implementation was flexible and robust enough, that many problems, both major and minor, which naturally cropped up with this new development were for the most part corrected before they obstructed the work of too many users. The fact that the network was explicitly an experiment no doubt also made users more tolerant.

1.4.4 Host Protocol Development

Specifications exist for the physical and logical message transfer between a host and its IMP. These specifications are generally called the IMP-to-host protocol. This protocol is not sufficient by itself, however, to specify the methods of communication between processes running in two possibly dissimilar hosts. Rather, the processes must have some agreement as to the method of initiating communication, the interpretation of transmitted data, and so forth. Although it would be possible for such agreements to be reached by each pair of hosts (or processes) interested in communication, a more general arrangement is desirable in order to minimize the amount of implementation necessary for networkwide communication. Accordingly, the host organizations formed a group (the Network Working Group or NWG, introduced above) to facilitate an exchange of ideas and to formulate additional specifications for host-to-host communications.

The NWG adopted a "layered" approach to the specification of communications protocols, wherein the higher layers of protocol use the services of lower layers; the advantages and disadvantages of the layered approach are discussed elsewhere in this report. As shown in Figure 3, the lowest layer is the

Figure 3 -- Layered Relationship of the ARPANET Protocols

IMP-to-host protocol. The next layer (called the host-to-host layer in the figure) specifies methods of establishing communications paths between hosts, managing buffer space at each end of a communications path, etc. Next, the Initial Connection Protocol or ICP specifies a standard way for a remote user (or process) to attract the attention of a network host, preparatory to using that host. The ICP provides the analog of the user pressing the attention button at a local terminal on a host. In the next layer is the Telecommunications Network or TELNET protocol, which was designed to support terminal access to remote hosts. TELNET is a specification for a network standard terminal and the protocol for communicating between this standard terminal and a host. The next logical protocol layer consists of function oriented protocols, two of which, File Transfer Protocol (FTP) and Remote Job Entry protocol (RJE), are shown in the figure. Finally, at any point in the layering process, it is possible to superimpose ad hoc protocols.

In the following subsections we discuss in some detail the events in the evolution of the host-to-host and TELNET protocols, and the events in the evolution of a number of other protocols in somewhat less detail. Appendix B contains a chronology, supplied by Jon Postel, of the host protocol development experience.

1.4.4.1 Host-to-Host Protocol

The Network Working Group was established in early 1969. By December 1969 an initial host-to-host protocol had been specified which supported communication between a terminal on one host and a process on another host. At a meeting in Salt Lake City in December 1969, the initial protocol specification was described to Lawrence Roberts of ARPA who was unhappy with it because the initial plan would not support transmission of electronic mail over the network. He instructed the Network Working Group to "go back and get it right."

By the spring of 1970 several successive versions of a host-to-host protocol had been developed, and a relatively formal meeting of the NWG was held at UCLA before mid-year at which the latest version of the protocol was described. Reactions to the described protocol were very negative. In June of 1970 there was a series of meetings held at UCLA and Harvard at which people from these two institutions tried finally to settle upon a host-to-host protocol and specify how it should be implemented. In August of 1970 some of the more general (and some thought more exotic) aspects of the host-to-host protocol being considered were ordered dropped from the protocol by Barry Wessler of ARPA.

thus administratively clearing away some of those issues which had prevented agreement. The NWG discussion continued at the 1970 Spring Joint Computer Conference; in particular, there was discussion between Crocker and Roberts regarding the formality to be sought for the protocol, and ARPA approvals required, and so forth. Another NWG meeting was held at the Fall Joint Computer Conference in November 1970 in Houston, Texas.

By early 1971 ARPA was growing increasingly impatient with the progress being made to settle on a host-to-host protocol. At a NWG meeting held in mid-February 1971 at the University of Illinois, a subcommittee was appointed to look at the host-to-host protocol to see what changes were immediately desirable or necessary. This subcommittee went directly from Illinois to Cambridge, Massachusetts, where it met for two days, wrote an interim report, and then reconvened a month later in Los Angeles. It appears that with the efforts of this committee (known as the "host-to-host protocol glitch cleaning committee") the design of the ARPANET host-to-host protocol was finally coming close to being settled.

At about this same time ARPA was beginning to exert great pressure not only to get the host-to-host protocol settled but also to get it implemented by the hosts. Some hosts had already implemented an early version of the protocol; in fact, at least one host, with especially productive programmers, was penalized for its efficiency by implementing several successive versions of host-to-host protocol. At a NWG meeting at the Spring Joint Computer Conference in Atlantic City in May 1971, Roberts strongly admonished the NWG to "finish something." At that NWG meeting Alex McKenzie took on the task of writing a definitive specification of the host-to-host protocol -- not to invent new protocol, but to write down what had been decided.

In October 1971 the final big NWG meeting was held at M.I.T., and was preceded by a programmers' workshop at which differences in implementations were clarified and eliminated. In January 1972 a McKenzie document describing the protocol was published and the ARPANET host-to-host protocol has remained essentially unchanged since.

1.4.4.2 The Evolution of Telnet

Early in the development of the ARPANET it became clear that a major function of the network would be to provide remote use of interactive systems. To allow a user at a terminal (connected to his local host) to control and use a process in a remote host, as if he were a local user of that remote host, a special mechanism was required. The problems to be overcome are legion; for example, the typical host expects its interactive terminals to be physically attached to the individual ports of its hardware terminal scanner rather than logically attached via a multiplexed connection to the network; a given host expects to communicate only with terminals with certain characteristics (e.g., half-duplex, line-at-a-time, physical echo, EBCDIC character set, 134.5 baud) while a remote user's terminal might have completely different characteristics (e.g., full-duplex, character-at-a-time, no character echo, ASCII character set, 300 baud). The TELNET protocol was an attempt to provide the special mechanism necessary to permit such communication.

As early as 1969 a few hosts had been programmed on an __ad hoc__ basis to permit terminal access from another host. In 1971 an NWG subcommittee was formed to consider the general problem of supporting interactive use of arbitrary hosts by users at arbitrary remote terminals. There was great controversy in the committee discussions, focusing on four issues: character set, connection establishment, echoing, and interrupt capability. By late 1972 there was enough consensus so that widespread implementation of an early version of the TELNET protocol had been accomplished.

Despite widespread implementation of the early TELNET protocol, its heavy and effective use, and numerous attempts to declare it complete, discussion of it continued. There were several problems with the early version:

1. Despite the attempt to permit a minimal implementation well suited to the constraints of small hosts, there was no well-defined minimal implementation. Even if some TELNET feature was not desired for a given implementation, it had to be provided in case some other implementation commanded its use.

2. The control structure was inadequate. For example, unless some exceedingly constraining assumptions were made, it was possible for the two ends of a TELNET connection to loop, commanding each other to take opposite actions.

3. The asymmetry of TELNET connections precluded one end from initiating certain functions, such as echoing behavior. This seriously constrained the use of TELNET protocol for character communication between processes not serving terminals, a role for which it would otherwise have been well suited and for which it was already frequently used in the absence of any better protocol.

4. The issue of interfacing character-at-a-time hosts to line-at-a-time hosts was poorly handled.

By early 1973 it had become apparent that minor adjustments to the early TELNET protocol would not solve these problems and that some fundamental changes were needed. A new subcommittee met and, with the previous experience to guide them, developed several fundamental principles. These new principles, when added to the earlier principles of the Network Virtual Terminal and the remote interrupt (synch) mechanism, resulted in a revised TELNET protocol which solved most of the earlier problems that had precluded universal acceptance of the protocol.

There was such enthusiasm for the new version that a schedule for "rapid" (within the year) implementation was laid out. However, the implementation of the new TELNET protocol proceeded more slowly than expected. Only in the past year have implementations been widely available. In retrospect, there were several reasons for the delay in the implementation: 1) at the time the revised protocol implementation was scheduled, implementation of the initial version had been completed and host system managers had not budgeted resources for a second implementation; 2) about this time ARPA's research interest in the network was declining and the network was entering a period of status quo operation; 3) despite initial belief that a clean method of phasing over from the initial protocol to the revised protocol existed, none was found by most implementers and consequently most chose to provide a complete implementation of the revised protocol to operate in parallel with the initial protocol; and 4) implementation for the most prevalent user host, the TIP, proved to be very difficult (because of the TIP's limited memory) and timeconsuming, thus implicitly relieving pressure on the server hosts to implement the revised protocol.

At the time this is being written, in late 1977, the new TELNET protocol has been the accepted standard for several years, and it is widely implemented and used.

1.4.4.3 The Evolution of the Other Host Protocols

There are several other host protocols the evolution of which should be briefly mentioned.

The File Transfer Protocol started out as two protocols, a Data Transfer Protocol and a File Transfer Protocol. To over-simplify, the Data Transfer Protocol was to specify the format of data being transferred and the File Transfer Protocol was to specify how it was transferred. Eventually, the File Transfer Protocol alone was defined with a data portion and a control portion. After the final push to specify the FTP, relatively little additional work was done, consisting only of a little effort to clean up fundamental aspects of the protocol, and a good bit of work reconciling the "reply codes" that different hosts used to indicate FTP-related events.

Before a Remote Job Entry protocol could be defined by the NWG as a whole, UCLA's IBM 360/91 host, a batch oriented host, needed some RJE-like protocol with which to serve a few users who wanted early access to the computing power of that particular host. Thus, led by the UCLA group, a protocol called the Remote Job Service or RJS protocol was defined and implemented. The NWG eventually got around to working on the problem of a Remote Job

Entry protocol and undertook a relatively massive effort to define such a protocol. However, by the time the RJE protocol definition was finished, half a dozen or so hosts had already implemented to interim RJS protocol. Since these included most of the hosts on the network interested in supporting remote batch, there was little incentive for them to implement the new RJE protocol. Thus, today the RJE protocol is carefully specified but to our knowledge is not implemented anywhere, and the RJS protocol prevails.

Another protocol much discussed within the NWG for a time is the Data Reconfiguration Service protocol. The Data Reconfiguration Service was to be a facility which would come to have tables to convert from the format of any host on the network to the format of any other. For reasons which are unclear, the service was never implemented.

Moving upward in sophistication, another protocol that was the subject of early discussion was one for graphics. Several versions of a graphics protocol were specified but there was never widespread implementation of any of them.

In addition to the host-to-host protocol which was finally specified after much iteration, a number of alternative protocols were suggested by various members of the NWG. Before the host-to-host protocol was settled upon, Richard Kalina and David Walden each suggested an alternative protocol. Even after the adoption of the host-to-host protocol, there was some discussion of experiments with a protocol derived from the Walden suggestion. More recently, as part of the ARPA-sponsored National Software works project, ???, Richard Schantz, and Robert Thomas have designed and implemented a host-to-host protocol known as MSG. Another protocol, known as TCP, deserves special mention.

Near the time of the formation of the International Network Working Group, as network interconnection began to be of great interest, discussions began on a standard inter-network protocol, particularly one which would correct some of the shortcomings of the ARPANET host-to-host protocol. At the AFIPS 1973 NCC in New York City a meeting was held at which certain ideas for a new host-to-host protocol were discussed. After some additional correspondence, Robert Kahn of ARPA and Vinton Cerf, then of Stanford, got together and designed a protocol known as TCP. Other members of INWG, perhaps not satisfied that TCP represented an international standard, continued developing still another host-to-host protocol (Cerf also participated in this later effort). TCP quickly became ARPA's choice of the host-to-host protocol to be used in situations where the ARPANET host-to-host protocol was insufficient or where inter-networking was required. With ARPA support, several TCP implementations were done and the protocol has come into relatively widespread use within the ARPANET, and its use is still spreading. It appears that ARPA envisions the day when TCP will replace the ARPANET host-to-host protocol. Meanwhile the host-to-host protocol that the rest of INWG was working on was finished, and documented. Just as the PTTs and North American common carriers submitted the X.25 standard to CCITT, so the INWG consensus protocol will most likely play little operational role in the ARPANET or elsewhere.

1.4.5 Network Growth -- A Summary

In the following three subsections we consider three aspects of the growth of the network: the traffic growth, the growth of the network topology, and the increase in the number and type of hosts on the network.

1.4.5.1 Traffic Growth

In early 1973, Roberts presented a curve of average host internode traffic growth for the network* which showed the level of internode network traffic to be increasing at a rate of a factor of ten every ten months. Internode traffic means traffic sent from a host on one node to a host on a different node; i.e., it does not include traffic sent between hosts on the same node. Based on this rapid rate of growth, Roberts predicted the network would run out of capacity in nine months. As shown in the following figure, shortly after Roberts' prediction the rate of internode traffic growth decreased sharply to roughly a factor of two every twenty months. It is interesting to speculate on the reason for this sharp decrease.

----

* L.G. Roberts, "Network Rationale: A 5-Year Reevaluation," Proceedings COMPCON 1973, February 1973, pp. 3-6.

Figure 4 -- Growth in Average Host Internode Traffic

Before the network existed, ARPA appears to have had a tendency to buy a computer for each of its research contractors. Once the existing computers were put on the network as hosts, ARPA appears to have shown greater reluctance to provide each of its contractors with a host, preferring instead that new contractors use existing hosts; in fact, some groups acquired all of their computing over the network. Thus it can be hypothesized that the existing hosts (and the few new hosts that were added) were used remotely more and more and network traffic increased more and more, until the hosts (at least the popular timesharing hosts) began to run out of capacity; this made it painless for new remote users to attempt to get service, and resulted, in turn, in a leveling off of network traffic growth. Therefore, instead of the network running out of capacity as predicted by Roberts, it seems that the hosts ran out of capacity while the network still has capacity left.

As already stated, the traffic shown in Roberts' curve and in the figure above includes only internode traffic. There are two reasons for excluding intranode traffic. First, intranode traffic puts a burden on only one node rather than on the network as a whole. Thus when Roberts, for instance, was attempting to calculate the effects of host traffic on network capacity, he naturally excluded intranode traffic. Second, the available intranode traffic statistics include some amount of test traffic being looped from a host through its node and back to the same host, and there is no convenient way to separate this looped test traffic from actual data traffic between two hosts on the same node. It is believed, however, that there is actually a significant amount of real traffic between hosts on the same node. For instance, Kleinrock reports* that during a week-long measurement, the level of intranode traffic amounted to a daily average of twenty percent of the level of internode traffic, and in some one-hour intervals the intranode traffic level was as much as eighty percent of the internode traffic level. A scan of available long term statistics on inter- and intranode traffic shows that intranode traffic levels have averaged between twenty and forty percent of internode traffic levels. Thus the traffic curve given in the figure above should be scaled up by this factor if all traffic is to be included.

That intranode traffic is a significant portion of all network traffic is interesting and probably indicative of four

----
* L. Kleinrock and W. Naylor, "On Measured Behavior of the ARPA Network," AFIPS Conference Proceedings, Volume 43, May 1974, pp. 767-780.

phenomena. First, the IMP is a handy interhost interface, and once one is installed in a computer center to connect some host onto the network, there is very soon pressure to connect other computers in the computer center to the IMP so that desired communication between the computers is possible. Second, when two computers are connected to the same IMP so they may both communicate with other computers in the network, communication between the two computers themselves comes free and begins to happen even if it was not initially thought to be desired. Third, the TIP (a host) has been chosen by several sites as the most flexible available terminal multiplexor and TIP-to-host traffic at these sites is likely to be intranode. Fourth, there is a (as yet still weak, but definite) tendency for hosts to be concentrated at a certain site and therefore often on the same IMP. The reason for this tendency is that, while some cynics would have guessed that every computer center manager is trying to build his empire as large as possible, in fact the world of computer center managers appears to include not only managers whose inclinations are as the cynics guessed but also many who dislike running computer centers but do so because they need the service supplied by the computer center. Once the network became available, some sites have arranged with some other sites that one site's computer was moved to a second site, and the second site managed it for the first site which used the computer over the network via a simple terminal concentrator. A further reason for this tendency (for hosts to be clustered) is the economy of scale possible when only one facility and staff is required for the operation of several computers.

1.4.5.2 Topology

The first ARPANET node was installed at the University of California at Los Angeles in late 1969 and the next three nodes were installed in California and Utah.

Figure 5: The ARPANET in December 1969

By June 1970 three East Coast and two more West Coast nodes were added, as well as two cross-country lines.

Figure 6: June 1970

IMPs continued to be delivered to the field at an average rate of approximately one per month, so that by late 1970 there were thirteen IMPs installed in the network. The IMPs were all entirely compatible, all being based on the Honeywell 516 computer.

Figure 7: December 1970

By two-thirds of the way through 1971, two additional 516 IMPs had been installed, the prototype TIP was running at BBN, and two TIPs were operational within the network, at MITRE and AMES.

Figure 8: September 1971

The TIPs were based on Honeywell 316 (instead of 516 computers and had as a component a 316-based IMP which was completely compatible with the 516 IMP but half as expensive. By early 1972 several additional IMPs and TIPs had been installed and the central part of the network between the East and West Coast clusters was beginning to fill out.

Figure 9: March 1972

By August 1972 a third cross-country line had been added and it was clear that in addition to the IMPs scattered throughout the center of the country, there were actually clusters of IMPs in four geographic areas, Boston, Washington, D.C., San Francisco, and Los Angeles.

Figure 10: August 1972

There follow once-yearly maps for the years 1973 to 1977 with which the reader can follow the continued growth of the ARPANET topology.

Figure 11: September 1973

Figure 121 June 1974

Figure 13: July 1975

Figure 14: July 1976

Figure 15: July 1977

Having shown the growth of the ARPANET topology through a series of geographic maps, it is interesting to consider the evolution of the topology on a more quantitative basis. We use the topologies given in the eleven maps shown above as the basis for the quantitative data shown in the Figure 16 chart. Columns 6 and 7 are missing the first six entries because the HAWAII, NORSAR, and LONDON nodes did not exist at the times the six earlier maps were made. The first three entries in columns 8 through 11 are missing because that information was not kept in the early days of the network. There are several interesting facts that should be noted from the chart. First, the number of network nodes has stayed about the same since 1975. Despite this, the network throughput has continued to increase. Next, note that intranode throughput actually is a significant fraction of total network throughput. Also, note the peak in average path length reached in 1974-75, and the subsequent decrease in average path length; selected lines were added to the network in 1975-76 as a direct response to network delay problems which occured in 1974-75. Finally, note that in July 1975 and July 1977, the path from HAWAII to LONDON is equaled by other paths in the network; the NCC (in the Boston area) was 15 hops from both UCSB (Santa Barbara, Californa) and FNWC (Monterey, California) in 1975, and

1	2	3	4	5	6	7	8	9	10	11
DEC69	4	2.00	1.33	2	---	--	---	----	----	----
JUN70	9	2.22	2.31	4	---	--	---	----	----	----
DEC70	13	2.46	2.76	6	---	--	---	----	----	----
SEP71	18	2.44	3.32	7	---	--	3.27%	2,892	3,121	6,013
MAR72	23	2.35	5.04	11	---	--	4.00%	11,633	21,073	32,706
AUG72	29	2.21	4.68	9	---	--	1.79%	682,502	287,953	970,455
SEP73	40	2.20	5.61	13	5.40	11	3.53%	2,893,130	742,746	3,635,876
JUN74	46	2.17	6.14	13	5.98	12	1.19%	3,125,955	1,513,777	4,639,732
JUL75	57	2.28	6.79	15	6.68	15	.67%	5,179,361	1,918,538	7,097,899
JUL76	58	2.45	5.26	11	5.13	10	.58%	6,627,968	1,961,726	8,589,694
JUL77	58	2.41	5.37	11	5.27	11	.94%	7,051,922	2,706,054	9,757,976

where the columns contain the following information:

Column 1: Date of Map
Column 2: Number of Nodes
Column 3: Average Connectivity
Column 4: Average Path Length
Column 5: Maximum Path Length
Column 6: Average Path Length, Minus HAWAII, NORSAR, LONDON
Column 7: Maximum Path Length, Minus HAWAII, NORSAR, LONDON
Column 8: Percentage of Node Unavailability
Column 9: Internode Throughput
Column 10: Intranode Throughput
Column 11: Sum of Internode and Intranode Throughput

Figure 16: Some Quantitative Data

In 1977 SRI2 (near San Francisco, California) is eleven hops from NRL (near Washington, D.C.).

There has been a good deal of actual measurement of the behavior of the ARPANET, and the most detailed discussion of this is presented in the paper entitled "On Measured Behavior of the ARPA Network" (L. Kleinrock and W.E. Naylor, AFIPS 1975 Conference Proceedings, Vol. 43, pp. 767-780).

1.4.5.3 Hosts

The first four hosts connected to the ARPANET were an SDS SIGMA-7 at UCLA, an SDS-940 at SRI, an IBM 360/75 at UCSB, and a DEC PDP-10 at the University of Utah. This beginning gave a good indication of the diversity of host manufacturer types and operating systems which would use the ARPANET. There follows a recent* schematic map of the ARPANET (Figure 17) showing the various hosts. Notice that the hosts range from small PDP-11s to large IBM systems, with a scattering of very special hosts such as ILLIAC-IV, running a variety of operating systems from relatively standard ones provided by manufacturers to very special ones constructed by university researchers.

Figure 18 charts the increase in number of hosts on the network against time.

Figure 19 provides a breakdown of the numbers and kinds of hosts on the network. It is based on information compiled by the NIC for inclusion in the ARPANET directory. Some of these hosts share a single host port on an IMP. Also, each of the twenty-three TIPs in the ARPANET logically provides a host

----
* A sampling of such schematic maps covering the entire history of the ARPANET is provided in Appendix 1.C.

Figure 17 -- ARPANET Logical Map -- June 1977

function although no physical host separate from the network node is required.

Figure 18 -- Number of ARPANET Hosts Vs. Time

Figure 19 -- Summary of ARPANET Hosts

1.5 The Impact of the ARPANET

The ARPANET has served well its original function as a testbed for a new computer communications technology. More recently the ARPANET has also given good operational service to a number of users who have come to depend on it for their computer communications service. However, a part of the original program plan for the ARPANET was technology transfer. The original program plan stated, "The transfer of interactive computer network technology will occur in three forms: (1) Dissemination of techniques and experimental results through the open scientific and technical literature, (2) Through the common carriers or other commercial organizations concerned with data transfer and dissemination, and (3) Through the military command and control centers for which the national Military Command System Support Center in the Pentagon serves as the focal point." The ARPANET's greatest success has perhaps been in this area of technology transfer.

Being an unclassified effort, implemented for the most part by individuals with an academic or research leaning, there have naturally been numerous papers written on the ARPANET. Two key sets of papers written relatively early in the ARPANET development were a set of five presented in a session at the AFIPS 1970 Spring Joint Computer Conference and another set of five presented in a session at the AFIPS 1972 Spring Joint Computer Conference. Both these sessions were organized by IPT and the two sets of five papers were specially bound together under ARPA-provided covers and distributed widely. We have already listed these and a number of other papers in the bibliographies provided elsewhere in this history. In 1975 IPT commissioned Becker and Hayes, Inc., of Los Angeles, California, to prepare a bibliography of publications related to the ARPANET (Selected Bibliography and Index of Publications about ARPANET, Becker and Hayes, Inc., February 1976, 185p.). This bibliography lists 561 relevant documents and includes a subject index. The document is available from NTIS under accession number AD-A026900. A complete set of the papers in the bibliography was also collected on microfiche. There was also a large number of informal working papers distributed amoung the various groups and individuals working on the ARPANET development. Many of these are also covered in the Becker and Hayes biblioграhy. The National Bureau of Standards has also constructed a bibliography on computer communications which includes hundreds of ARPANET-related publications.

In addition to the many papers and presentations that have been given on the ARPANET, there have been numerous demonstrations of the technology. Of course many of the demonstrations have been informal or relatively low key or relatively short, but in several instances the demonstrations have been truly major productions. Recently for example, IPT, with help from several members of the ARPANET community, took a demonstration team to Europe where a series of demonstrations was given for members of the NATO staff over a two-week period. Earlier, for the First International Conference on Computer Communications held in Washington, D.C., in 1972, 50 Kb phone lines were leased from existing network sites to the conference site at the Washington Hilton Hotel, and an ARPANET TIP was set up in a demonstration room in the hotel for the duration of the conference. This event was the idea of Lawrence Roberts and was orchestrated by Robert Kahn. Dozens of members of the ARPANET community were involved. Manufacturers of all manner of computer terminals were invited to connect their terminals to the demonstration TIP. Throughout the conference hours, each day of the conference, there were individuals available to demonstrate the use of programs on ARPANET hosts from the manufacturer-provided terminals. A relatively thick booklet was written, and many copies made available at the conference, by means of which visitors to the demonstration area could follow "do it yourself" directions to use programs on many network hosts. A twenty or thirty minute motion picture about the ARPANET and the promise of computer communications was produced and shown at the conference. Coming at a time when the TIP had not been availble for a very long time, when only a limited number of terminals had been tried with the ARPANET, and when many hosts had completed the initial implementation of the necessary host software but few had had it running for very long, the ICCC demonstration provided an important stimulus for the ARPANET community to pull together and get the network in true operational shape. The demonstration itself was a spectacular success; with everything working amazingly well, many visitors remarked that the ARPANET technology "really is real" and carried this impression back home with them. The assurance with which Roberts promised the demonstration and the routine way in which he spoke of it while it was happening no doubt enhanced the impression taken home by the visitors, and belied the crash efforts and feelings of panic of the members of the ARPANET community who were called upon to execute the demonstration.

There has been good success in transferring the ARPANET technology to other parts of the Department of Defense. The Defense Communications Agency procured two small networks, essentially identical to the ARPANET in function, for the purpose of gaining experience with the ARPANET technology. Called the PWIN and EDCN networks, the first of these was used in the WWMCCS effort and the second was used in connection with the successor to AUTODIN I. NSA also procured two smaller networks essentially identical to the ARPANET. Called COINS and PLATFORM, the first is used to connect the computers of a number of intelligence agencies and the second is used for internal NSA computer communications.

Where direct copies of the ARPANET have not been procured, the ARPANET technology has nonetheless affected the characteristics of new DoD networks being built. The new SATIN IV, the Strategic Air Command network presently under construction, has a strong component of packet switching in its makeup. The new DoD common user network, AUTODIN II, being constructed by DCA, is explicitly a second generation ARPANET.

Outside the U.S. military, the commercial world has begun to use the ARPANET technology or variations on it. Several companies have filed with the F.C.C. or already been licenced to offer communications services based more or less directly on the packet-switching technology developed in ARPANET. Among these are Telenet Communications Corporation (for which BBN arranged the financing and Lawrence Roberts was President and is now Chairman of the Board), Tymnet, Graphnet, IT&T, and AT&T. Because of its access to substantial ARPANET expertise, of the several common carriers Telenet has used the ARPANET technology most directly. Telenet now serves about eighty cities in the continental U.S. and has made arrangements to connect to several foreign networks and serve several foreign cities. Tymnet initially developed in parallel to ARPANET as a means of making the time-sharing services of Tymshare available to a wide geographic area, and used a substantially different although related communications tehcnology; however, a new version of Tymnet being built uses techniques closer to those developed for ARPANET. Graphnet, IT&T, and AT&T all have announced their intention to provide public packet-switching services.

A number of U.S. companies have also procured or are procuring private corporate networks utilizing many of the techniques developed for ARPANET. For instance, it was recently announced that Citibank of New York City has constructed (by contract to BBN) a private network very similar to the ARPANET. An increasing number of commercial RFPs call for packet-switching or for functions which can only be provided using packet switching. A number of companies have taken advantage of the fact that the ARPANET technology is in the public domain to obtain the listings of the ARPANET software. There has even been a "pop" reaction to the technology. The December 7, 1972, issue of Rolling Stone features an article which "approves" the popular usefulness of the ARPANET and other ARPA-developed technology, and more recently work has begun on networks of personal computers.

While technology transfer to foreign institutions is farther from the intent of the ARPA charter, the wide foreign acceptance and use of the technology confirms the fundamental correctness and importance of the technology ARPA has developed. Several nations' PTTs (the foreign national Postal, Telephone, and Telegraph authorities) have made a commitment to the development of packet-switching networks, there have been several foreign research networks, and several international networks are being developed or are under consideration. There follows a list of some of these networks:*

CIGALE -- an operational network developed by a French government research agency.

RCP -- built by the French PTT and operational as a testbed.

TRANSPAC -- under construction by the French PTT and scheduled to become operational for public use in 1978.

EPSS -- an experimental packet-switching service built by the UK PTT which became operational in 1976.

CTNE -- a packet-switching service operated by the Spanish PTT.

Datapac -- a packet-switching network built by the Trans-Canada Telephone System which is in the early phases of operation.

JIPNET -- practically a copy of the ARPANET built in Japan.

----
* More detail on this list may be found in "Planned New Public Data Networks", P.T. Kirstein, Computer Networks, Volume 1, Number 2, September 1976, pp. 79-94; Kirstein is himself a member of the ARPANET community and was instrumental in arranging the installation of the London node of the ARPANET.

EIN -- the European Information Network, built jointly by the U.K., Switzerland, France, and Italy and strongly influenced by CIGALE.

EURONET -- a network under discussion by the European Economic Community, initially to use the EIN technology.

No doubt there are other networks besides those mentioned above in the planning phase or under construction. With so many networks coming into being, technology exchange and standards become important issues. From the time of the 1972 ICCC, representatives of various countries and institutions interested in computer networks met informally to discuss their experience and to consider possible standards. In 1977 the International Network Working Group (INWG) was formed. Modeled on the ARPANET Network Working Group, ARPA IPT essentialy hand-picked Vinton Cerf to be INWG's first chairman and offered the services of the the ARPANET NIC to coordinate and distribute INWG working notes. Later INWG became associated with the International Federation for Information Processing, ARPA cut back its NIC support, and ARPA's influence in INWG dwindled. From its beginning, INWG was a forum at which techniques other than those used in the ARPANET were considered; nonetheless, the ARPANET for a long time remained the one big, existing network against which new ideas were compared.

The international packet-switching standardization effort has been especially effective. With the urging of Datapac, Transpac, Telenet, and <UK?>, CCITT, the international communications standards organization at which all of the world's communications authorities are represented, has with remarkable speed adopted standards for connecting hosts to packet-switching networks and packet-switching networks to each other. Known as X.25 and X.92, these standards clearly address, for purposes of international communication, issues which were first seen to be of importance in the ARPANET development, perhaps where the ARPANET was seen to be deficient.

Even now the development sequence begun with the ARPANET is far from complete. ARPA is currently pushing onward in such related areas as use of ground radio transmission media in the development of a mobile packet-switching network, integration of packet-switching technology with satellite broadcast and multi-access technology, development of security mechanisms adequate for use with a packet-switching network, inter-network connection, improved host-to-host protocols, use of packet-switching technology in intra-fleet and fleet-to-shore communication, packet transmission of speech, and so forth. Other institutions, notably Xerox, have developed a high-speed multi-access bus using packet-switching technology; this effort is under the direction of Dr. Robert Metcalfe, earlier a member of the ARPANET community.

The mention of Metcalfe calls to mind one of the most effective routes for ARPANET technology transfer -- by the movement of people. Metcalfe is but one of hundreds of individuals who have worked on the ARPANET or used one of its many sites, later moved to an institution removed from direct contact with the ARPANET, and helped convert the new institution to use of the ARPANET technology.

A final thought on the impact of the ARPANET. While the ARPANET continues to provide useful service, and while ARPA is pursuing a number of interesting advanced programs, the ARPANET is now largly out of the mainstream of packet-switching network development. There are second-generation networks being built by the U.S. government, commercial industry, and many foreign institutions. It is with these other, newer networks that new protocols are being developed, new ideas are being tried, and there are large groups of active researchers. The ARPANET has been passed by and is no longer the center of the state of the art.

1.6 Maturity and Handover to DCA

From the very beginning it was assumed that ARPA would eventually give up operation of the network. The program plan for the network project written in the middle of 1968 noted that once the experiment of building the network was complete, a common carrier could be requested to take over the management of the network of IMPs and to provide "digital message service directly to the individual users on a tariff basis" which would permit ARPA to terminate its system responsibility. It was hoped that this transfer could take place three or four years after the beginning of the network development. ARPA still had in mind attempting to transfer the network to private industry in 1972 when the Office of Telecommunications Processing wrote a letter to ARPA strongly urging them to divest the network to private industry.

For help in figuring out how the transfer of the network to a common carrier should take place, IPT contracted with Paul Baran and his colleagues at Cabledata Associates, Inc. of Palo Alto, California, to study the issue. This study took place between April 1973 and January 1974, and the final result of the study was to serve as the basis for an RFP for a common carrier to take over the network. A couple of lengthy interim reports and a massive final report resulted from the study ("ARPANET Management Study", Final Technical Report, Cabledata Associates, Inc., January 14, 1974).

By the time the Cabledata study was done, the idea of spinning off the network to a private company had lost its appeal. Possibly a government auditor had something to do with turning things toward internal government divestiture rather than private when he worried out loud about the propriety of turning over a government-developed network to a common carrier which would make a profit selling network service back to the government. In any case, after a good bit of searching for a proper home for the network, it appeared to ARPA that DCA might be the place.

At this point DTAACS stepped in and acted as matchmaker between DCA and ARPA. It is noteworthy that at the time, Dr. Rechtin was head of DTAACS, the same Dr. Rechtin who as Director of ARPA had signed the original program plan for the ARPANET, and who had become a real believer in the network.

The general guidelines of transfer of the network from ARPA to DCA were worked out by Col. Russell, then Deputy Director of

IPT, and Dr. Estil Heversten, then Deputy Technical Director of DCA. Their negotiations resulted in a memorandum of agreement which was signed in the first days of March 1975 by Lt. Gen. Lee Paschall, Director, Defense Communications Agency, and Dr. George Heilmeier, then Acting Director, Defense Advanced Research Projects Agency.

The memorandum of agreement called for management of the ARPANET to be transferred to DCA as of July 1, 1975, with a six-month phaseover period from July 1 until December 31, during which ARPA would continue to help DCA with ARPANET management while DCA acclimated itself to the job. The memorandum also called for a detailed transition plan to be written, which Stephen Walker of IPT succeeded in completing (with assistance from Science Applications Inc.) by June 1975.

The management of the network was officially transferred to DCA on July 1, 1975. However, speaking realistically, Stephen Walker of IPT continued as manager for the network for several more months, with help from Mr. Robert Brownfield of DCA. As Brownfield learned the job, he took more and more responsibility for management of the network until by January 1976 he was carrying out the day-to-day management of the network with help from Walker. Walker then detached himself from day to day involvement in the network and continued to serve only as ARPA representative regarding ARPANET affairs. Along with the transfer of ARPANET management from ARPA to DCA, the technical functions that had been being performed at RML were also transferred to DCA and the procurement functions were transferred from RML to DECCO, the procurement agency with which DCA was most used to doing business.

There are several aspects of the memorandum of agreement and the transition plan which are worthy of mention here. The network was to be an operational DoD facility, to be used solely for government business. The concept of "ARPANET sponsors" was invented with sponsors being those users or collections of users (e.g., ARPA, NSS) who originally owned ARPANET equipment before management of it was turned over to DCA. Ownership of the equipment was to remain with the sponsors. DCA was to finance the operation and maintenance of the network through use of the DCA managed Communications Industrial Fund, which would recover its costs by a prorated allocation to sponsors based on the equipment used by the sponsors. DCA was to contract initially with BBN and SRI to perform the ARPANET operations and maintenance and NIC functions, and with NAC initially if topological consulting was needed; it was clearly implied that DCA could retain other contractors to perform these functions eventually and certainly after the first year. DCA was to operate the network for a period of three years and thereafter if necessary until equivalent service could be provided.

The transfer of the network to DCA ends the period covered by this history; however, it may be of interest to the reader that the network has now been under DCA management for over two years and continues to serve its users well. Plans are being made regarding the disposition of the ARPANET once the new common user network now being built by DCA is finished, but the ARPANET will remain in operation for at least another couple of years, and possibly for many more.

1.A BBN ARPANET Bibliography

1.A.1 Technical Reports

AD682905 BBN Report No. 1763, "Initial Design for Interface Message Processors for the ARPA Computer Network", January 1969.

*ADA019160 BBN Report No. 1822, "Specifications for the Interconnection of a Host and an IMP", revision of January 1976.

AD730725 BBN Report No. 2161, "A Study of the ARPA Network Design and Performance", Kahn and Crowther, August 1971.

*ADA014398 BBN Report No. 2183, "Terminal Interface Message Processor - User's Guide," updated version of August 1975.

*ADA002481 BBN Report No. 2184, "TIP Hardware Manual," revision of November 1974.

*AD776995 BBN Report No. 2277, "Specifications for the Interconnection of Terminals and the Terminal IMP," Rettberg, revision of June 1973.

BBN Report No. 2491, "Throughput in the ARPA Network -- Analysis and Measurement," McQuillan, January 1973 (text also contained in QTR No. 16, January 1973).

ADA025358 BBN Technical Information Report (TIR) No. 89, "The Interface Message Processor Program," updated version of June 1976 (Version 3231).

ADA033351 BBN TIR No. 90, "The Network Control Center Program," updated version of November 1976 (Version 131).

ADA031617 BBN TIR No. 91, "The Terminal Interface Message Processor Program," updated version of August 1976 (Version 400).

AD781467 BBN Report No. 2831, "Adaptive Routing Algorithms for Distributed Computer Networks," McQuillan, May 1974.

BBN TIR No. 93, "The Remote Job Entry Mini-Host," August 1974 (text also contained in QTR No. 6, July 1974).

BBN Report No. 2891, "A Proposed Experiment in Packet Broadcast Satellite Communications," Rettberg and Walden, September 1974.

BBN Report No. 2918, "Network Design Issues," November 1974 (text also contained in QTR No. 7, October 1974).

ADA022040 BBN Report No. 2999, "Pluribus Document 1: Overview," May 1975.

BBN Report No. 2930, "Pluribus Document 2: System Handbook," January 1975.

BBN Report No. 3000, "Pluribus Document 3: Configurator," in production.

BBN Report No. 3001, "Pluribus Document 4: Basic Software," December 1975.

BBN Report No. 2931, "Pluribus Document 5: Advanced Software," April 1975.

BBN Report No. 3002, "Pluribus Document 6: Functional Specifications," February 1976.

BBN Report No. 3004, "Pluribus Document 7: Maintenance," September 1976.

BBN Report No. 3056, "The Atlantic Satellite Packet Broadcast and Gateway Experiments," R. Binder, R. Rettberg, and D. Walden, April 1975.

ADA018341 BBN Report No. 3126, "A Multiprocessor Design," W. Barker, September 1975.

1.A.2 Published Papers*

W. Teitleman and R.E. Kahn, "A Network Simulation and Display Program," Proceedings of the Third Annual Princeton Conference on Information Sciences and Systems, March 1969.

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," AFIPS Conference Proceedings 36, June 1970, pp. 551-567; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc., 1974, pp. 300-316; also in Computer Communications, P.E. Green and R.W. Lucky (eds.), IEEE Press, 1975, pp. 375-391; also in Computer Networking, R.P. Blanc and I.W. Cotton (eds.), IEEE Press, 1976, pp. 60-76.

F.E. Heart and S.M. Ornstein, "Software and Logic Design Interaction in Computer Networks," International Computer State of the Art Report No. 6: Computer Networks, Infotech Information Ltd., Maidenhead, Berkshire, England, pp. 423-462.

R.E. Kahn, "Terminal Access to the ARPA Computer Network," in Courant Computer Science Symposium 3 -- Computer Networks, R. Rustin (ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 147-166.

R.E. Kahn and W.R. Crowther, "Flow Control in a Resource-Sharing Computer Network," Proceedings of the Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications Systems, Palo Alto, California, October 1971, pp. 108-116; also in IEEE Transactions on Communications, Vol. COM-20, No. 3, Part II, June 1972, pp. 539-546; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc., 1974, pp. 230-237; also in Computer Networking, R.P. Blanc and I.W. Cotton (eds.), IEEE Press, 1976, pp. 117-125.

D.C. Walden, "A System for Interprocess Communication in a Resource-Sharing Computer Network," Communications of the ACM, Vol. 15, No. 4, April 1972, pp. 221-230; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc., 1974, pp. 340-349.

----
* Ordered according to date of writing.

R.H. Thomas and D.A. Henderson, "McROSS -- A Multi-Computer Programming System," AFIPS Conference Proceedings, Vol. 40, June 1972, pp. 281-293; also in Computer Networking, R.P. Blanc and I.W. Cotton (eds.), IEEE Press, 1976, pp. 246-258.

S.M. Ornstein, F.E. Heart, W.R. Crowther, S.B. Russell, H.K. Rising, and A. Michel, "The Terminal IMP for the ARPA Computer Network," AFIPS Conference Proceedings 40, June 1972, pp. 243-254; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc., 1974, pp. 317-328; also in Computer Communications, P.E. Green and R.W. Lucky (eds.), IEEE Press, 1975, pp. 354-365.

H. Frank*, R.E. Kahn, and L. Kleinrock**, "Computer Communications Network Design -- Experience with Theory and Practice," AFIPS Conference Proceedings, Vol. 40, June 1972, pp. 255-270; also in Networks, Vol. 2, No. 2, 1972, pp. 135-166; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc., 1974, pp. 254-269.

A.A. McKenzie, B.P. Cosell, J.M. McQuillan, and M.J. Thrope, "The Network Control Center for the ARPA Network," Proceedings of the First International Conference on Computer Communication, Washington, D.C., October 1972, pp. 185-191; also in Computer Networking, R.P. Blanc and I.W. Cotton (eds.), IEEE Press, 1976, pp. 319-325.

R.E. Kahn, "Resource-Sharing Computer Communications Networks," Proceedings of the IEEE, Vol. 60, No. 11, November 1972, pp. 1397-1407; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc., 1974, pp. 208-218; also in Computer Communications, P.E. Green and R.W. Lucky (eds.), IEEE Press, 1975, pp. 537-547.

J.M. McQuillan, W.R. Crowther, B.P. Cosell, D.C. Walden, and F.E. Heart, "Improvements in the Design and Performance of the ARPA Network," AFIPS Conference Proceedings 41, December 1972, pp. 741-754.

----
*Network Analysis Corporation.
**University of California at Los Angeles.

W.R. Crowther, R.D. Rettberg, D.C. Walden, S.M. Ornstein, and F.E. Heart, "A System for Broadcast Communication: Reservation-ALOHA," Proceedings of the Sixth Hawaii International Conference on System Sciences, January 1973, pp. 371-374.

D.C. Walden, "Host-to-Host Protocols," International Computer State of the Art Report No. 24: Network Systems and Software, Infotech, Maidenhead, England, pp. 287-316.

N.W. Mimno, B.P. Cosell, D.C. Walden, S.C. Butterfield, and J.B. Levin, "Terminal Access to the ARPA Network -- Experience and Improvements," Proceedings of the Seventh Annual IEEE Computer Society International Conference, San Francisco, California, February 1973, pp. 39-43; also in Computer Networking, R.P. Blanc and I.W. Cotton (eds.), IEEE Press, 1976, pp. 287-291.

E.W. Wolf, "An Advanced Computer Communication Network," AIAA Computer Network Systems Conference, April 1973, AIAA Paper No. 73-414.

F.E. Heart, S.M. Ornstein, W.R. Crowther, and W.B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," AFIPS Conference Proceedings 42, June 1973, pp. 529-537; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc., 1974, pp. 329-337; also in Computer Communication Networks, R.L. Grimsdale and F.F. Kuo (eds.), Proceedings of the NATO Advanced Study Institute of September 1973, Sussex, England, published by Noordhoff International Publishing, Leyden, The Netherlands, 1975, pp. 159-180; also in Computer Communications, P.E. Green and R.W. Lucky (eds), IEEE Press, 1975, pp. 366-374.

R.H. Thomas, "A Resource Sharing Executive for the ARPANET," AFIPS Conference Proceedings 42, June 1973, pp. 155-163; also in Advances in Computer Communications, W.W. Chu (ed.), Artech House Inc., 1974, pp. 359-367.

F.E. Heart, "The ARPA Network," in Computer Communication Networks, R.L. Grimsdale and F.F. Kuo (eds.), Proceedings of the NATO Advanced Study Institute of September 1973, Sussex, England, published by Noordhoff International Publishing, Leyden, The Netherlands, 1975, pp. 19-33.

W.R. Crowther, J.M. McQuillan, and D.C. Walden, "Reliability Issues in the ARPA Network," Proceedings of the ACM/IEEE Third Data Communications Symposium, November 1973, pp. 159-160; also in Computer Networking, R.P. Blanc and I.W. Cotton (eds.), IEEE Press, 1976, pp. 142-143.

J.M. McQuillan, "Design Considerations for Routing Algorithms in Computer Networks," Proceedings of the Seventh Annual Hawaii International Conference on System Sciences, Honolulu, Hawaii, January 1974, pp. 22-24; also in Computer Networking, R.P. Blanc and I.W. Cotton (eds.), IEEE Press, 1976, pp. 108-110.

S.C. Butterfield, R.D. Rettberg, and D.C. Walden, "The Satellite IMP for the ARPA Network," Proceedings of the Seventh Annual Hawaii International Conference on System Sciences, Honolulu, Hawaii, January 1974, Computer Nets Supplement, pp. 70-73.

S.M. Ornstein, W.B. Barker, R.D. Bressler, W.R. Crowther, F.E. Heart, M.F. Kraley, A. Michel, and M.J. Thrope, "The BBN Multiprocessor," Proceedings of the Seventh Annual Hawaii International Conference on System Sciences, Honolulu, Hawaii, January 1974, Computer Nets Supplement, pp. 92-95.

A.A. McKenzie, "Some Computer Network Interconnection Issues," AFIPS Conference Proceedings 43, May 1974, pp. 857-859.

E.A. Akkoyunlu, A.J. Bernstein*, and R.E. Schantz, "Interprocess Communication Facilities for Network Operating Systems," IEEE Computer, Vol. 7, No. 6, June 1974, pp. 46-55.

F.E. Heart, "Implications of the Computer-Communication Partnership," in Preprints of Papers, MEDINFO 74; The First World Conference on Medical Informatics, Stockholm, Sweden, August 1974, pp. 21-27; proceedings to be published by North-Holland, The Netherlands.

----
*State University of New York at Stony Brook.

F.E. Heart, "Networks and the Life Sciences: the ARPA Network and Telenet," Federation Proceedings, Federation of American Societies for Experimental Biology (FASEB), Vol. 33, No. 12, December 1974, pp. 2399-2402; also in Computers in Life Science Research, W. Siler and D.A.B. Lindberg (eds), FASEB and Plenum Press, 1975, pp. 209-215.

J.M. McQuillan, "The Evolution of Message Processing Techniques in the ARPA Network," International Computer State of the Art Report No. 24: Network Systems and Software, Infotech, Maidenhead, England, pp. 541-578.

R.H. Thomas, "JSYS Traps - A TENEX Mechanism for Encapsulation of User Processes," AFIPS Conference Proceedings 44, May 1975, pp. 351-360.

S.M. Ornstein, W.R. Crowther, M.F. Kraley, R.D. Bressler, A. Michel, and F.E. Heart, "Pluribus -- A Reliable Multiprocessor," AFIPS Conference Proceedings 44, May 1975, pp. 551-559.

F.E. Heart, S.M. Ornstein, W.R. Crowther, W.B. Barker, M.F. Kraley, R.D. Bressler, and A. Michel, "The Pluribus Multiprocessor System," in Multiprocessor Systems: Infotech State of the Art Report, Infotech International Ltd., Maidenhead, Berkshire, England, 1976, pp. 307-330.

W.R. Crowther, F.E. Heart, A.A. McKenzie, J.M. McQuillan, and D.C. Walden, "Issues in Packet-Switching Network Design," AFIPS Conference Proceedings 44, May 1975, pp. 161-175; also in Computer Networking, R.P. Blanc and I.W. Cotton (eds.), IEEE Press, 1976, pp. 182-196.

J. Burchfiel, R. Tomlinson, and M. Beeler, "Functions and Structure of a Packet Radio Station," AFIPS Conference Proceedings 44, May 1975, pp. 245-251.

R.S. Tomlinson, "Selecting Sequence Numbers," Proceedings of the ACM SIGCOMM-SIGOPS Interface Workshop on Interprocess Communications, March 1975, pp. 11-23.

J.M. McQuillan and D.C. Walden, "Some Considerations for a High Performance Message-Based Interprocess Communication System," Proceedings of the ACM SIGCOMM-SIGOPS Interface Workshop on Interprocess Communications, March 1975, pp. 77-86.

S.M. Ornstein and D.C. Walden, "The Evolution of a High Performance Modular Packet-Switch," Conference Record of the 1975 International Conference on Communications, June 1975, Vol. I, pp. 6-17 to 6-21.

R.Binder, "A Dynamic Packet-Switching System for Satellite Broadcast Channels," Conference Record of the 1975 International Conference on Communications, June 1975, Vol. III, pp. 41-1 to 41-5.

R.D. Bressler, M.F. Kraley, and A. Michel, "Pluribus: a Multiprocessor for Communications Networks," Fourteenth Annual ACM/NBS Technical Symposium -- Computing in the mid-70's: an Assessment, June 1975, pp. 13-19.

D.C. Walden, "Experiences in Building, Operating, and Using the ARPA Network," Proceedings of the 2nd USA-Japan Computer Conference, Tokyo, Japan, August 1975, pp. 453-458.

R.D. Rettberg and D.C. Walden, "Gateway Design for Computer Network Interconnection," in: Communications Networks: Proceedings of the European Computing Conference on Communications Networks, London, England, September 1975, published by Online Conferences Limited, Uxbridge, England, pp. 113-128.

M.F. Kraley, "The Pluribus Multiprocessor," Digest of the 1975 International Symposium on Fault-Tolerant Computing, Paris, France, June 18-20, 1975, p. 251 (abstract only).

B.P. Cosell, J.M. McQuillan, and D.C. Walden, "Techniques for Detecting and Preventing Multiprogramming Bugs," in: Minicomputer Software, J.R. Bell and C.G. Bell (eds.), North-Holland Publishing Co., 1976, pp. 381-388.

A.A. McKenzie, "The ARPA Network Control Center," Proceedings of the Fourth ACM Data Communications Symposium, Quebec City, Canada, October 1975, pp. 5-1 to 5-6.

R. Binder, J.M. McQuillan, and R.D. Rettberg, "The Impact of Multi-Access Satellites on Packet Switching Networks," EASCON '75 Record, IEEE Electronics and Aerospace Systems Conference, Washington, D.C., October 1975, pp. 63-A to 63-F.

B.P. Cosell, P.R. Johnson, J.H. Malman, R.E. Schantz, J. Sussman, R.H. Thomas, and D.C. Walden, "An Operational System for Computer Resource Sharing," Proceedings of the Fifth Symposium on Operating System Principles, Austin, Texas, November 1975, pp. 75-81.

R.D. Bressler and R.H. Thomas, "Design Issues in Distributed Computing," in Distributed Systems: Infotech State of the Art Report, Infotech International Ltd., Maidenhead, Berkshire, England, 1976, pp. 215-222.

V. Cerf*, A. McKenzie, R. Scantlebury**, and H. Zimmermann***, "Proposal for an International End to End Protocol," ACM Computer Communication Review, Vol. 6, No. 1, January 1976, pp. 63-89.

W.F. Mann, S.M. Ornstein, and M.F. Kraley, "A Network-Oriented Multiprocessor Front-End Handling Many Hosts and Hundreds of Terminals", AFIPS Conference Proceedings 45, June 1976, pp. 533-540.

P.J. Santos, "Software Instrumentation for Maintainability of Distributed Computer Networks," Fifteenth Annual ACM/NBS Technical Symposium--Directions and Challenges, June 1976, pp. 143-148.

R. Binder, "A Reservation-TDMA Protocol for the Atlantic Packet Satellite Experiment," Workshop on Packet-Switching and Communication Satellites, Institut de Recherche d'Informatique et d'Automatique, Rennes, France, June 1976 (Abstract only).

----
*Defense Advanced Research Projects Agency
**Institut de Recherche d'Informatique et d'Automatique (France)
***National Physical Laboratories (U.K.)

R. Binder, Nai-Ting Hsu, R.D. Rettberg, D.C. Walden, and R.C. Weissler, "Current Status of the Satellite Interface Message Processor," Workshop on Packet-Switching and Communication Satellites, I.R.I.A., Rennes, France, June 1976 (Abstract only).

J.M. McQuillan, "A Status Report on the ARPANET," Australian Computer Society/International Federation for Information Processing Joint International Symposium on Data Communications Technology and Practice, August 1976, pp. 13.1-13.7.

J.M. McQuillan, "Strategies for Implementation of Multi-Host Computer Networks," ACS/IFIP Joint International Symposium on Data Communications Technology and Practice, August 1976, pp. 26.1-26.6; also in Computer Communication Review, Vol. 6, No. 4, October 1976, pp. 19-24.

F.E. Heart and D.C. Walden, "Communications Applications of the Pluribus Computer," Conference Record of the 1976 IEEE National Telecommunications Conference, Dallas, Texas, 29 November-1 December 1976, pp. 7.1-1 to 7.1-5.

A. Evans and C.R. Morgan, "The Tinman and Communications Applications," presented at the Workshop on the Design and Implementation of Programming Languages, Cornell University, October 1, 1976.

R. Levine, J.M. McQuillan, and R. Schantz, "Distributed Systems," a section of "New Directions for Operating Systems: A Workshop Report," J.C. Browne, Operating Systems Review, Vol. 11, No. 1, January 1977, pp. 14-19.

J.M. McQuillan and D.C. Walden, "Issues in Packet Switching Network Design and the ARPANET Design Decisions," submitted to Computer Networks.

J. Davidson, W. Hathaway+, N. Mimno, J. Postel++, R. Thomas, and D. Walden, "The ARPANET TELNET Protocol: Its Purpose, Principles, Implementation, and Impact on Host Operating System Design," submitted for presentation at the ACM/IEEE Fifth Data

----
*Carnegie-Mellon University
+NASA Ames Research Center
++USC Information Sciences Institute

Communications Symposium, Snowbird, Utah, September 27-29, 1977.

J. Jacobs***, L. Lee***, A. Viterbi***, R. Binder, R. Bressler, N. Hsu, and R. Weissler, "CPODA -- a Demand Assignment Protocol for SATNET," submitted for presentation at the ACM/IEEE Fifth Data Communications Symposium, Snowbird, Utah, September 27-29, 1977.

***Linkabit Corporation

1.B Host Protocol Chronology

April 1969
Network Working Group formed, First RFC published,
S. Crocker "Host Software," RFC 1, NIC 4687, 7-Apr-69,
May 1969
First host level Protocol Specified,
G. Deloche "Host Software," RFC 9, NIC 4695, 1-May-69,
February 1970
Second host level protocol proposed,
S. Crocker "New HOST-HOST Protocol," RFC 33, NIC 4735, 12-Feb-70,
June 1970
Version 2 of second host level protocol proposed,
S. Crocker, "An Official Protocol Proffering," RFC 54, NIC 4756, 18-Jun-70,
August 1970
Host level protocol published as official specification,
S Crocker, "Official Host to Host Protocol," Document 1, NIC 7149, 3-Aug-70,
The Message Switching host level protocol is proposed,
D. Walden "A System for Interprocess Communication in a Resource Sharing Computer Network," RFC 62, NIC 4962, 3-Aug-70, Also published in Communications of the ACM, 15(4):221-230, April 1972,
Network Graphics Meeting
Where , Who ???
Initial Connection Protocol proposed,
S. Crocker "3rd Level Ideas and other Noise," RFC 66, NIC 5409, 26-Aug-70,
November 1970
Network Working Group Meeting at FJCC in Houston,
J. Postel "Network Meeting Report," RFC 77, NIC 5604, 20-Nov-70,
E. Meyer "Network Meeting Notes," RFC 82, NIC 5619, 9-Dec-70,
Initial network use at UCSB and RAND,
E. Harslem "NCP Status Report: UCSB/RAND," RFC 78, NIC 5199, undated,
December 1971
Data Reconfiguration Service first suggested,
E. Harslem "Protocols and Data Formats," RFC 80, NIC 5608, 1-Dec-70,
January 1971

Graphics Protocol first proposed.
S. Crocker "Proposal for a Network Standard Format for a Data Stream to Control Graphics Display," RFC 86, NIC 5631, 5-Jan-71.
Remote Job Entry protocol first proposed.
R. Braden "NETRJS - A Third Level Protocol for Remote Job Entry," RFC 88, NIC 5668, 13-Jan-71.
Initial use of the network at MIT and Harvard.
R. Metcalfe "Some Historic Moments in Networking," RFC 89, NIC 5697, 19-Jan-71.
February 1971
Telnet protocol first proposed.
J. Melvin "A First Cut at a Proposed Telnet Protocol," RFC 97, NIC 5740, 15-Feb-71.
E. Meyer "Logger Protocol Proposed," RFC 98, NIC 5744, 11-Feb-71.
Network Working Group held at University of Illinois.
R. Watson "Notes on the Network Working Group Meeting," RFC 101, NIC 5762, 23-Feb-71.
R. Watson, "Addendum to NWG Meeting Notes," RFC 108, NIC 5807, 25-Mar-71.
Host level protocol modified.
S. Crocker "Output of the Host/Host Protocol Glitch Cleaning Committee," RFC 102, NIC 5763, 22-Feb-71.
March 1971
Host level protocol modified.
R. Bressler "Output of the Host-Host Protocol Glitch Cleaning Committee," RFC 107, NIC 5806, 23-Mar-71.
April 1971
File Transfer Protocol first proposed.
A. Bhushan "A File Transfer Protocol," RFC 114, NIC 5823, 16-Apr-71.
May 1971
Network Working Group meeting held at SJCC in Atlantic City.
J. Heafner, "Minutes of the Network Working Group Meeting," RFC 164, NIC 6778, 25-May-71.
Telnet protocol specified.
T. O'Sullivan "TELNET Protocol," RFC 158, NIC 6768, 19-May-71.
Data Reconfiguration Service specified.
B. Anderson "Data Reconfiguration Service -- An Implementation Specification," RFC 166, NIC 6780, 25-May-71.
June 1971
File Transfer Protocol design initiated.

Bhushan, A. "The Data Transfer Protocol," RFC 171, NIC 6793, 23-JUN-71.
Bhushan, A. "The File Transfer Protocol," RFC 172, NIC 6794, 23-JUN-71.
July 1971
Mail Protocol first discussed.
Watson, R. "A Mail Box Protocol," RFC 196, NIC 7141, 20-JUL-71.
Graphics meeting held by MIT.
August 1971
Terminal IMP Protocol Implementation
McKenzie, A. "NCP, ICP, and Telnet: The Terminal IMP Implementation," RFC 215, NIC 7545, 30-Aug-71.
October 1971
Network Working Group Meeting and Programmers Workshop
Vezza, A. "Network Working Group Meeting Schedule," RFC 234, NIC 7651, 5-OCT-71.
Postel, J. "Report of the Protocol Workshop," RFC 295, NIC 8335, 2-JAN-72.
November 1971
Network Graphics Meeting
Padlipsky, M. "Graphics Meeting Report," RFC 282, NIC 8164, 8-DEC-71.
January 1972
Protocol proposed by Graphics Committee.
Michener, J. "Graphics Protocol - Level 0 only," RFC 292, NIC 8302, 12-JAN-72.
April 1972
File Transfer protocol Meeting
Bhushan, A. "Data and File Transfer Workshop Notes," RFC 327, NIC 9630, 17-MAR-72.
Graphics protocol meeting
Telnet Protocol specification published
Postel, J. "Telnet Protocol," RFC 318, NIC 9348, 3-APR-72.
Remote Job Entry protocol meeting
June 1972
Remote Job Entry protocol published
Holland, C. "Proposed Remote Job Enter Protocol," RFC 360, NIC 10602, 24-JUN-72.
August 1972
Mail commands included in evolving File Transfer protocol specification
Bhushan, A. "Comments on the File Transfer Protocol (RFC 354)," RFC 385, NIC 11357, 18-AUG-72.

October 1972
  Demonstration of ARPANET at International Computer Communications Conference (ICCC) in Washington D.C.
January 1973
  The principle of negotiated options for telnet first enunciated.
    Cosell, B. "TELNET Issues," RFC 435, NIC 13675, 5-JAN-73.
February 1973
  A common user command language proposed.
    Padlipsky, M. "Tentative Proposal for a Unified User Level Protocol," RFC 451, NIC 14135, 22-FEB-73.
  Mail protocol meeting
    Kudlick, M. "Network Mail Meeting Summary," RFC 469, NIC 14798, 8-MAR-73.
    Bhushan, A. "FTP and Network Mail System," RFC 475, NIC 14919, 6-MAR-73.
March 1973
  Telnet Protocol Meeting
    McKenzie, A. "Telnet Protocol Meeting Announcement," RFC 461, NIC 14416, 14-FEB-73.
  File Transfer protocol Meeting
    McKenzie, A. "File Transfer Protocol; Meeting Announcement and a Proposed Document," RFC 454, NIC 14333, 16-FEB-74.
May 1973
  Telnet protocol specified, a thorough revision of the control signal mechanisms and the adoption of the negotiated option concept.
    McKenzie, A. "Telnet Protocol Specification," RFC 495, NIC 15371, 1-MAY-73
  Resource Sharing Executive Workshop meeting
  Users Interest group meeting
    Crocker, D. "Arpanet Users Interest Working Group Meeting," RFC 585, NIC 20050, 6-NOV-73.
June 1973
  A file access protocol is proposed.
    Day, J. "File Access Protocol," RFC 520, NIC 16819, 25-JUNE-73.
July 1973
  Network Graphics Meeting
    Bunch, S. "Minutes of Network Graphics Group Meeting," RFC 549, NIC 17795, 21-AUG-73.
October 1973
  A common text editing subsystem proposed.
    Padlipsky, M. "Neted: A Common Editor for the ARPA Network,"

RFC 569, NIC 18972, 15-OCT-73.

July 1974

A cross-network debugging program is proposed.
Mader, E. "Network Debugging Protocol," RFC643, NIC 30873, JUL-74.

November 1974

A Host to Front End protocol proposed.
Padlipsky, M. "A Proposed Protocol for Connecting Host Computers to ARPA-Like Networks via Directly-Connected Front End Processors," RFC 647, NIC 31117, 12-NOV-74.

December 1974

A procedure oriented protocol is proposed.
Postel, J. "Procedure Call Protocol Documents," RFC 674, NIC 31484, 12-DEC-74.

June 1975

Major extensions to the IMP/Host interface proposed
Walden, D.C. "IMP/Host and Host/IMP Protocol Change," RFC 687, NIC 32654, 2-JUN-75.

1.C Selection of ARPANET Logical Maps

We can select from the following:

December 1970
February 1971
April 1971
August 1971
December 1971
March 1972
August 1972
February 1973
August 1973
September 1973
January 1974
June 1974
November 1974
January 1975
April 1975
June 1975
July 1975
October 1975
July 1976
August 1976
October 1976
November 1976
December 1976
March 1977
June 1977

2. DESIGN AND IMPLEMENTATION

This chapter of our report describes the technical basis for the ARPANET. We begin with an overview of some of the main technical concepts of the ARPANET and of packet-switching. Then, we go on to treat in turn the topological design of the system, the nature of the communications equipment within it, the subnetwork protocols, and the host level protocols, and finally, a treatment of the performance of the overall ARPANET system.

2.1 Overview and Technical Basis

The ARPANET represents a major step forward in computer communications. As such, it introduced a number of new terms into the technical literature. We will discuss these terms and present the design point of view which underlies much of the ARPANET technology. We present this perspective in two parts: first, a discussion of the design goals of the system, and secondly, a brief overview of the choices made in each of the important design areas in the ARPANET.

2.1.1 Terminology

__Nodes.__ The nodes of the network are real-time computers, with limited storage and processing resources, which perform the basic packet-switching functions.

__Hosts.__ The Hosts of the network are the computers, connected to nodes, which are the providers and users of the network services.

__Lines.__ The lines of the network are some type of communications circuit of relatively high bandwidth and reasonably low error rate.

__Connectivity.__ We assume a general, distributed topology in which each node can have multiple paths to other nodes, but not necessarily to all other nodes. Simple networks such as stars or rings are degenerate cases of the general topology we consider.

__Message.__ The unit of data exchanged between source Host and destination Host.

__Packet.__ The unit of data exchanged between adjacent nodes.

__Acknowledgment.__ A piece of control information returned to a source to indicate successful receipt of a packet or message. A packet acknowledgment may be returned from an adjacent node to indicate successful receipt of a packet; a message acknowledgment may be returned from the destination to the source to indicate successful receipt of a message.

Store and Forward Subnetwork. The node stores a copy of a packet when it receives one, forwards it to an adjacent node, and discards its copy only on receipt of an acknowledgment from the adjacent node, a total storage interval of much less than a second.

Packet Switching. The nodes forward packets from many sources to many destinations along the same line, multiplexing the use of the line at a high rate.

Routing Algorithm. The procedure which the nodes use to determine which of the several possible paths through the network will be taken by a packet.

Node-Node Transmission Procedures. The set of procedures governing the flow of packets between adjacent nodes.

Source-Destination Transmission Procedures. The set of procedures governing the flow of messages between source node and destination node.

Host-Node Transmission Procedures. The set of procedures governing the flow of information between a Host and the node to which that Host is directly connected.

Host-Host Transmission Procedures. The set of procedures governing the flow of information between the source Host and the destination Host.

Some of the more obvious differences among packet-switching networks can be cited briefly. The ARPANET splits messages into packets up to 1000 bits long; some of the other networks have 2000-bit packets and no multipacket messages. Hosts connect to a single node in the ARPA Network and SITA; multiple connections are possible in Cyclades and EIN. Dynamic routing is used in the ARPANET and EIN; a different adaptive method is used in SITA; fixed routing is presently used in Cyclades. The ARPANET delivers messages to the destination Host in the same sequence as it accepts them from the source Host; Cyclades does not; in EIN the sequence is optional. Clearly, many of the design choices made in these networks are in conflict with each other. The resolution of these conflicts is essential to the planning and building of balanced, high performance networks, particularly since many future designs will be intended for networks which are larger, less experimental, and more complex.

We next summarize how the IMP in the ARPANET performs its functions as a message switching center and interface between host computers. Figure 2-1 shows a diagram of message flow in

Figure 2-1 Packets and Messages

the ARPANET and illustrates some of the terminology. The host sends the IMP a message with up to 8063 data bits. The source IMP breaks this up into packets with up to 1008 data bits. When the packet is successfully received at each IMP, an acknowledgment or ack is sent back to the previous IMP. When the message arrives at the destination IMP it is reassembled, that is, the packets are combined into a message again. The message is sent to the destination host and when it has been accepted, a Ready For Next Message which we abbreviate as RFNM is sent back to the source host. The RFNM is also a packet and it is acknowledged. Several points are worth noting. First, acks are not actually separate transmissions, but are piggy-backed in packets to cut down on overhead. Next, packets on the links between IMPs are checksummed in the modem interface hardware and the IMP employs a positive acknowledgment retransmission scheme; that is, if a packet is in error, it is not acknowledged. It is then retransmitted until an acknowledge is received. Further, an IMP may send the several packets of a message out on different links. Because of retransmission (out of order) of a packet on a link and transmission of packets on alternate links, the packets of a message may arrive at the destination IMP out of order and must be reassembled into the correct order for transmission into the host.

2.1.2 Design Goals: Performance and Functions

In this section we define what we believe are fundamental properties and requirements of packet-switching networks and what we believe are the fundamental criteria for measuring network performance.

2.1.2.1 Basic Issues

We begin by giving the properties central to packet-switching network design. The key assumption here is that the packet processing algorithms (routing, acknowledgment/retransmission strategies used to control transmission over noisy circuits, etc.) result in a virtual network path between the hosts with the following characteristics:

a. Finite, fluctuating delay -- A result of the basic line bandwidth, speed of light delays, queueing in the nodes, line errors, etc.

b. Finite, fluctuating bandwidth -- A result of network overhead, line errors, use of the network by many sources, etc.

c. Finite packet error rate (duplicate or lost packets) -- A result of the acknowledgment system in any store-and-forward discipline (this is a different use of the term "error rate" than in traditional telephony).

Duplicate packets are caused when a node goes down after receiving a packet and forwarding it without having sent the acknowledgment. The previous node then generates a duplicate with its retransmission of the packet. Packets are lost when a node goes down after receiving a packet and acknowledging it before the successful transmission of the packet to the next node. An attempt to prevent lost and duplicate packets must fail as there is a tradeoff between minimizing duplicate packets and minimizing lost packets. If the nodes avoid duplication of packets whenever possible, more packets are lost. Conversely, if the nodes retransmit whenever packets may be lost, more packets are duplicated.

d. Disordering of packets -- A property of the acknowledgment and routing algorithms.

These four properties describe what we term the store-and-forward subnetwork.

There are also two basic problems to be solved by the source and destination in the virtual path described above:

e. Finite storage -- A property of the nodes.

f. Differing source and destination bandwidths -- Largely a property of the hosts.

(Note: the question is frequently raised whether the source and destination nodes or the source and destination hosts should solve these problems. This question is addressed in a later section.)

The fundamental requirements for packet-switching networks are dictated by the six properties enumerated above. These requirements include:

a. Buffering -- Buffering is required because it is generally necessary to send multiple data units on a communications path before receiving an acknowledgment. Because of the finite delay of the network, it may be desirable to have buffering for multiple packets in flight between source and destination in order to increase throughput. That is, a system without adequate buffering may have unacceptably low throughput due to long delays waiting for acknowledgment between transmissions. Buffering is also required when input traffic momentarily exceeds output capacity.

b. Pipelining -- The finite bandwidth of the network may necessitate the pipelining of each message flowing through the network by breaking it up into packets in order to decrease delay. The bandwidth of the circuits may be low enough so that forwarding the entire message at each node in the path results in excessive delay. By breaking the message into packets, the nodes are able to forward the first packet of the message through the network ahead of the later ones. For a message of P packets and a path of H hops (with small propagation delays), the minimum delay is proportional to P + H - 1 instead of P * H, where the proportionality constant is the packet length divided by the transmission rate. (See section 2.6.3 below for a derivation and more exact result.)

c. Error Control -- The node-to-node packet processing algorithm must exercise error control, with an acknowledgment system in order to deal with the finite packet error rate of the circuits. It must also detect when a circuit becomes unusable, and when to begin to use it again. In the source-to-destination message processing algorithm, the destination may need to exercise some controls to detect missing and duplicated messages or portions of messages, which would appear as incorrect data to the end user. Further, acknowledgments of message delivery or non-delivery may be useful, possibly to trigger retransmission. This mechanism in turn requires error control and retransmission itself, since the delivery reports can be lost or duplicated. The usual technique is to assign some unique number to identify each data unit and to time out unanswered units. The error correction mechanism is invoked infrequently, as it is needed only to recover from node or line failures.

d. Sequencing -- Since packet sequences can be received out of order, the destination must use a sequence number technique of some form to deliver messages in correct order, and packets in order within messages, despite any scrambling effect that may take place while several messages are in transit. The sequencing mechanism is frequently invoked since it is needed to recover from line errors.

e. Storage allocation -- The fact that storage in the nodes is finite means that both the packet processing and the message processing algorithms must exercise control over its use. The storage may be allocated at either the sender or the receiver.

6. Flow Control -- The different source and destination data rates may necessitate implicit or explicit flow control rules to prevent the network from becoming congested when the destination is slower than the source. These rules can be tied to the sequencing mechanism, with no more messages (packets) accepted after a certain number, or tied to the storage allocation technique, with no more messages (packets) accepted until a certain amount of storage is free, or the rules can be independent of these features.

In satisfying the above six requirements, the algorithm often exercises contention resolution rules to allocate resources among several users. The twin problems of any such facility are:

fairness -- resources should be used by all users fairly (perhaps in accordance with appropriate priority rules);

deadlock prevention -- resources must be allocated so as to avoid deadlocks.

Notice that each of these algorithms is implemented by a <u>distributed computation</u>. IMP-to-IMP transmission control involves cooperation between every pair of neighboring IMPs. Source-to-destination transmission control is a distributed process between the pair of IMPs exchanging a message. Finally, flow control and routing are distributed algorithms which involve all the IMPs in the network. Each IMP makes local decisions about global functions. The process of routing messages from source to destination involves all the IMPs in the network, in order that the best path for the message be chosen and agreed upon. Such distributed computations are quite different from conventional algorithms. They are not initialized, nor do they run to completion and halt. In a real sense, the ARPA routing calculation, flow control techniques, and so on, have been in progress ever since 1969, because some part of the Network has always been running since then. These algorithms are continuously active processes on a large number of different processors. In fact, the number of processors and the interconnection between them is subject to change at any moment. They must run completely without human intervention. They perform contention resolution among bidders for shared resources, and they must do so without races or deadlocks. (We have also come to believe that it is essential to have a reset mechanism to unlock "impossible" deadlocks and other conditions that may result from hardware or software failures.)

2.1.2.2 Network Performance Goals

Packet-switching communications systems have two fundamental goals in the processing of data -- low delay and high throughput. Each message should be handled with a minimum of waiting time, and the total flow of data should be as large as possible. The difference between low delay and high throughput is important. What the network user wants is the completion of his data transmission in the shortest possible time. The time between transmission of the first bit and delivery of the first bit is a function of network delay, while the time between delivery of the first bit and delivery of the last bit is a function of network throughput. For interactive users with short messages, low delay is more important.

There is a fundamental tradeoff between low delay and high throughput, as is readily apparent in considering some of the mechanisms used to accomplish each goal. For low delay, a small packet size is necessary to cut transmission time, to improve the pipelining characteristics, and to shorten queueing latency at each node; furthermore, short queues are desirable. For high throughput, a large packet size is necessary to decrease the circuit overhead in bits per second and the processing overhead per bit. That is, long packets increase the effective circuit bandwidth and nodal processing bandwidth. Also, long queues may be necessary to provide sufficient buffering for full circuit utilization. Therefore, the network may need to employ seperate mechanisms if it is to provide low delay for some users and high throughput for others.

To these two goals one must add two other equally important goals, which apply to message processing and to the operation of the network as a whole. First, the network should be cost-effective. Individual message service should have a reasonable cost as measured in terms of utilization of network resources; further, the network facilities, primarily the node computers and the circuits, should be utilized in a cost-effective way. Secondly, the network should be reliable. Messages accepted by the network should be delivered to the destination with a high probability of success. And the network as a whole should be a robust computer communications service, fault-tolerant, and able to function in the face of node or circuit failures.

In summary, delay, throughput, reliability, and cost are the four criteria upon which packet-switching network designs should be evaluated and compared. Further, it is the combined performance in all four areas which counts. For instance, poor delay and throughput characteristics may be too big a price to pay for "perfect" reliability. What are the primary issues regarding the cost of computer networks? There are two kinds of costs considered here: the cost of the actual network components, and the cost of the use of the network. The first cost is a measure of the expense of connecting some component to the network, and the second is a measure of the expense of utilizing the resources of that component. We are concerned here with outlining the effects of these costs on the balance among the various system parameters of the network.

<u>Low Cost for Network Connectivity</u>. A major consideration in the cost of the ARPANET is the cost of connectivity. The basic variables are circuit costs and node costs; the values for the ARPANET are shown in Tables 2-1 and 2-2.

Bandwidth Kbs/sec	Termination Cost $/Month	Line Cost $/Month/Mile
9.6	650	.40
19.2	850	2.50
50	850	5.00

230.4	1350	30.00

Table 2-1 ARPANET Line Costs

Machine Type	Cost	Configuration (Kbs Total)
516 IMP	$100K	up to 4 hosts (400)
		up to 5 lines (800)
		up to 7 devices total
316 IMP	$50K	up to 4 hosts (300)
		up to 5 lines (600)
		up to 7 devices total
316 TIP	$100K	up to 2 hosts (300)
		up to 3 lines (600)
		up to 64 terminals (100)

Table 2-2 ARPANET Node Costs

Several implications follow from these figures:

1. The layout problem for networks is an important one, since reductions in the total number of circuit miles in a network represents substantial dollar savings.

2. Low cost comes at the expense of both low delay and high throughput if lower line speeds are chosen.

3. Low cost also comes at the expense of low delay and high throughput if fewer lines are used, since more traffic is forced to use each line.

4. As a consequence of these points, some network designers, notably the Network Analysis Corporation, attempt to find a relatively low cost network layout and then test it by simulation to determine if average message delay is below some threshold and the throughput obtained (when all nodes send to all other nodes) is above some threshold.

<u>Low Cost for Network Use.</u> A second aspect of the cost of networks is the cost of the use of the network, which may be counted in one of several ways:

1. $/bit, $/character, $/packet, or $/message costs for the shipping of data through the network.

2. These charges may be rated per mile or per hop or may be distance-independent.

3. There may be different grades of service at different costs.

These costs are the translation into dollars of the utilization rates for various network resources. These resources can be catalogued as follows:

1. line bandwidth
2. node processor bandwidth
3. node storage

Next we present some of the primary issues regarding the reliability of computer networks. In parallel with the discussion of network cost above, there are two basic topics examined here: the reliability of the network connections themselves, and the reliability of the network services. In the first instance, we are interested in how reliable the network components are. In the second case, the subject is the reliability of the use of the network facilities for data transmission. Again, we are focussing in this section on the broad design issues which affect network performance as a whole. In section 2.3 on routing, we examine closely the effects that routing algorithms can have on network reliability.

High Reliability of Network Connectivity. One can attempt to minimize the probability that a line or node will be inoperative, and thus to reduce the probability that a node cannot communicate with the rest of the network. In practice, this is a matter of maintaining a high mean time between failures and a low mean time to repair. One can also measure reliability in terms of the number of nodes and/or lines necessary to disconnect the network, taking into consideration the probability of the various events. Alternatively, one can consider the size of the components of the disconnected network, or the fraction of node pairs not connected by any network path.

A different level of solution to the problem of network reliability is redundant network design, primarily in the layout of the circuits connecting the nodes. In this way, a network can be constructed which is much more reliable as a whole than any one component. In the ARPANET, a design constraint has been that an IMP must be connected to the network by at least two circuits, so that the probability that an IMP cannot communicate with the network is very small. Of course, this principle can be applied to other components in the network as well. The node computers can be backed up with alternate computers, or may be equipped with redundant interfaces and processors. The host computers also may wish to be connected to the network at more than one point by means of separate communications facilities.

High Reliability of Network Use. The reliability of the message processing can be measured in terms of the percentage of messages delivered, or the detected error rate, or the undetected error rate. Measures to improve the reliability of message processing range from error detecting and correcting hardware to redundancy in software to backup message storage. The costs of these approaches are multiple: they add to the complexity of the system, and may degrade its performance, in addition to representing a dollars cost. In general, the communications subnetwork becomes more costly as these measures of reliability are improved. At some point, it becomes appropriate to pass the cost of these improvements on to the user. However, a minimum level of reliability is necessary for the operation of the communications subnetwork. Guarantees concerning a grade of service better than this minimum level might reasonably cost more.

<u>The Tradeoff between Low Cost and High Reliability.</u> It is clear that there is a natural tradeoff between low-cost networks and high-reliability networks. This tradeoff exists in building either sparse networks or highly-connected networks, and in providing special mechanisms to ensure the reliable transmission of data or choosing not to implement such safeguards. In short, the price for reliability must be paid somewhere, either in the actual cost of constructing and maintaining the network, or in

the cost to the user of unreliable network service. It is difficult to generalize, but it may often be the case that a low-cost network without sufficient measures for reliability may prove more costly to use in the long run than a network with a higher cost for higher network reliability. Stated differently, it may be cheaper to build a network to be fault-tolerant, redundant, and error-detecting than to build these measures into each user process that communicates with the network.

2.1.2.3 Intended Functions

One of the most important factors in determining the overall design of any computer communications system is the set of functions and applications which must be supported by the system. In the case of the ARPA computer network, many different factors combined to make the design of a pioneering new approach to computer communications a necessary step. Foremost among these requirements was the need to connect terminals to remote computers with a very low response time. This was necessary in order to permit remote terminals to use timesharing computers as if they were connected directly to them. Other aspects of the computer environment are also worth noting. At the time the ARPANET was first built in 1969, there were no standard computer communications protocols of any kind. Furthermore, connections between computer mainframes were extremely rare and usually special purpose. Very few interconnections of computers had ever been attempted, practically none between computers of different manufacturers.

Thus, the ARPANET had to be created with brand new technical approaches. The concept of a subnetwork of packet-switching nodes, separate from the host computers, was a new idea. These computers were dedicated to the job of communications, freeing the host computers for applications processing. Furthermore, these node computers needed to be very fast so that packets could be routed through the network through many intermediate packet-switching nodes in less than a second. Finally, it was envisioned that the nodes should be able to support relatively high amounts of traffic since the intended uses of the network included remote job entry and file transfers, as well as interactive traffic. These high bandwidth applications were necessary in order to meet ARPA's goal of resource sharing among the various ARPA-sponsored research centers.

Another important function to be performed by the communications subnetwork was interfacing between computers and terminals of widely varying types. Thus, 16-bit minicomputers should be able to communicate with 32-bit and 36-bit large-scale computers and every computer should be able to communicate with every kind of terminal in use at the various sites. This formidable problem necessitated the design of a set of standard interfaces from host computers and terminals to the network and a set of hierarchical communications protocols for communication among the various types of equipment.

The intended functions of the ARPANET can be summarized as follows: general purpose intercommunication between a wide variety of host computers and terminals with a minimum impact on those equipments and users and providing for a high level of performance.

2.1.3 Fundamental Design Choices

We believe there are six major areas in which the key choices must be made in designing a packet-switching network:

--Transmission Facilities

--Switching Technology

--Topological Structure

--Communications Equipment

--Communications Protocols

--Interface Characteristics

2.1.3.1 Transmission Facilities

We next consider some of the important characteristics of the circuits used in the ARPANET.

The bandwidth of the network circuits is their most important characteristic. It defines the traffic-carrying capacity of the network, both in the aggregate and between any given source and destination. What is less obvious is that the bandwidth (and hence the time to clock a packet out onto the line) may be the main factor determining the transit delays in the network. The minimum delay through the network depends mainly on circuit rates and lengths, and additional delays are largely accounted for by queueing delay, which is directly proportional to circuit bandwidth. These two factors lead to the general observation that the faster the network lines, the longer the packet can be, since long packets have less overhead and permit higher throughput, while the added delay due to length is less important at high circuit rates. In addition, more packet and message buffering is required when higher speed circuits are used.

The major effect of circuits with appreciable delay in a system requiring packet acknowledgment is that they require more buffering in the nodes to keep them fully loaded. That is, the node must maintain more packets in flight at once over a circuit with longer delay. This effect may be so large (a circuit using a satellite has a delay of a quarter of a second) as to require significantly more memory in the nodes. This memory is needed at the nodes connected to the circuit to permit sufficient packet buffering for node-to-node transmission using the circuit. The subtle point is that additional buffering is also required at all nodes in the network that may need to maintain high source-to-destination rates over network paths which include this circuit. If they are to provide maximum throughput, they need sufficient message buffering to keep the entire network path fully loaded.

Traditionally, the telephone carriers have quoted error rates in the following manner: "No more than an average of 1 bit in 10 to the 6th bits in error." This definition is not entirely adequate for packet switching, though it may be for continuous transmission. For packet switching, the average bit error rate is less interesting than the average packet error rate (packets with one or more bits in error). For example, ten bits in error in every tenth packet is a 10% packet error rate, while one bit in error in every packet is a 100% packet error rate, yet the two cases have the same bit error rate.

An example of an acceptable statement of error performance would be as follows:

> The circuit operates in two modes. Mode 1: no continuous sequence of packet errors longer than

two seconds, with the average packet error rate less than one in a thousand. Mode 2: a continuous sequence of errors longer than two seconds with the following frequency distribution:

> 2 seconds no more often than once per day
> 1 minute no more often than once per week
> 15 minutes no more often than once per month
> 1 hour no more often than once per 3 months
> 6 hours no more often than once per year
> 1 day

While the figures above may seem too stringent in practice, the mode 1 bit error rate is actually quite lax compared to conventional standards. In any case, these are the kinds of behavior descriptions needed for intelligent design of packet-switching network error control procedures. Therefore, it is important that the carriers begin to provide such descriptions.

The packet error rate of a circuit has two main effects. First, if the rate is high enough, it can degrade the effective circuit bandwidth by forcing the retransmission of many packets.

While this is basically a problem for the carrier to repair, the network nodes must recognize this condition and decide whether or not to continue to use the circuit. This is a tradeoff between reduced throughput with the circuit and increased delay and less network connectivity without it. Before the circuit can be used, it must be working in both directions for packets and for control information like routing and acknowledgments, and with a sufficiently low packet error rate.

The second effect of the error rate is present even for relatively low error rates. It is necessary to build a very good error-detection system so that the users of the network do not see errors more often than some specified extremely low frequency. That is, the network should detect enough errors so that the effective network error rate is at least an order of magnitude less than the host error or failure rate. A usual technique here is a cyclic redundancy check on each packet. This checksum should be chosen carefully; to first order, its size does not depend on packet length and it should be quite large, for example 24 bits for 50-Kbs lines and 32 bits for multi-megabit lines or lines with high error rates. (Note: assuming that the probability of packet error is proportional to the product of packet length and bit error rate, the checksum length should be proportional to the log of the product of the desired time between undetected errors, the bit error rate, and the total bandwidth of all network circuits.)

2.1.3.2 Switching Technology

In designing the ARPANET one of the key questions was how the subscriber equipment (hosts and terminals) would share the network transmission facilities. Various possibilities exist. There are two general kinds of switching methods: point-to-point methods, such as multiplexing and switching, and multipoint or broadcast methods, such as polling and contention. In the first class, multiplexing, there is frequency division multiplexing, time division multiplexing, and statistical time division multiplexing. At the time the ARPANET was designed, no adequate off-the-shelf technology for multiplexing such diverse sets of subscribers into a common network was available. The key problem here is incompatibility of electrical and functional characteristics of the various devices. Since multiplexing is inherently a transparent technique, no conversion would have been possible. With various switching technologies such as circuit switching, message switching, or packet switching, code conversion, data conversion, and speed conversion are possible. Circuit switching involves a dedicated physical path between the communicating subscribers, such as in the telephone system. This would have been inappropriate for the ARPANET due to the long setup delays. Message switching usually involves a store-and-forward process in which each node forwards the message to the next node which stores the message on the disk for later forward transmission. Such systems often have delays of many minutes. Packet switching was invented for use on the ARPANET and can be seen as a logical outgrowth of message switching since it does not use secondary storage. The packets of each message are routed independently on a store-and-forward basis which allows pipelining of traffic through the network giving high efficiencies while permitting low delays. Multipoint techniques including polling were not used on the ARPANET primarily because of the perponderance of asynchronous character-at-a-time terminals at the various ARPANET sites. Also, most of the ARPANET sites did not have computers which accessed their terminals on a polled basis. In fact, the absence of synchronous line-at-a-time polled terminals from the ARPANET environment is quite an important aspect of the overall ARPANET design.

2.1.3.3 Topological Structure

The subject of network topology is a complex one, and we limit ourselves here to a few general observations. A distributed topology was chosen for the ARPANET as the least-cost network design which would meet delay and throughput requirements.

The problem of designing a general distributed network represented an important research problem in its own right and ARPA contracted with Network Analysis Corporation (NAC) to develop tools for designing such networks. There are many subproblems which can be identified in topological design, including node location, link capacity assignment, traffic flow assignment to links, combined capacity and flow assignment, and finally, the most general problem, topology capacity and flow assignment. At the time that the ARPANET was designed, the only topological algorithms in existence were for centralized algorithms. Thus, NAC had to develop algorithms which found the minimum-cost network layout satisfying a certain level of traffic capacity and minimizing delay given subscriber locations, traffic flows, network element costs, and possible switch locations. This they did by a variety of different kinds of algorithms, which cannot be discussed in detail here. A family of procedures has been developed over the years which have been used to design optimal and near optimal connections for the ARPANET subscribers and which have been used to measure how close the actual ARPANET comes to cost effectiveness. The actual ARPANET configuration cannot be restructured every time a new subscriber needs to be connected. Therefore, it is always somewhat less than optimal. The studies by NAC have shown ARPANET topologies to be within a few percent of the best cost which could be expected.

The connectivity of the ARPANET nodes was chosen to be relatively uniform, usually two or three. It is obvious that nodes with only a single line are to be avoided for reliability considerations. Nodes with many circuits also present a reliability problem since they remove so much network connectivity when they are down. We also feel that the direction for future evolution of network geometries will be towards a "central office" kind of layout with relatively fewer nodes and with a high fan-in of nearby hosts and terminals. This tendency will become more pronounced as higher reliability in the node computer becomes possible, even for large systems. One reason that we favor this approach is that a large node computer presents an increased opportunity for shared use of the node resources (processor and memory) among many different devices leading to a much more efficient and cost-effective implementation. This trend will mean that in the future, even more than now, a key cost of network topology will be the ultimate data connection to the user (host or terminal), who may be far from the central office. Concentrators and multiplexors have been the traditional solution; in packet-switching networks, a small node computer should fill this function. In conclusion, we see flexibility and extensibility as two key requirements for the node computer. These factors together with increasing performance and fan-in requirements imply a very high reliability standard as well.

2.1.3.4 Communications Equipment

The design of a large-scale computer communications network such as the ARPANET requires a set of communications equipment of several different types. In the case of the ARPANET which was designed at a time when the communications equipment industry was in its infancy, there were very few off-the-shelf communications processors to choose from. Today, the list is much longer:

--front-end processors

-multiplexors

-line concentrators

--terminal controllers (remote concentrators)

--circuit switches

--message switches

--packet switches

--network monitoring centers

A decision was made to construct the ARPANET using packet-switching nodes. These nodes were designed from a standard minicomputer base with additions for high speed interfaces to the wide band circuits and to host computers. Later, the network grew to include terminal controllers which were merged with these packet switches. At an early stage in the network's development, the need for a network control center was clear and the development of the NCC host was also initiated. The decision was made in most cases to locate the IMPs and TIPs near to the ARPA-sponsored facilities. This eliminated the need in most cases for local access networks of any great complexity. Therefore, multiplexors and concentrators are not used in the ARPANET; all terminals and hosts connect directly to switching nodes. One exception to this rule has been the connection of some host computers over relatively long distances using the very distant host (VDH) interface which is simply a provision for wide band circuit connection between host computers and the ARPANET.

Another noteworthy development over the past several years has been the proliferation of frontend processors in order to connect host computers to the ARPANET. This step was not the original intention of the ARPANET designers who felt that host computers should connect directly to the ARPANET to take full advantage of the power and flexibility of the network. However, subsequent experience has shown that the complexity of connecting some host computers directly to the ARPANET resulted in very expensive programming jobs with reduced efficiency within the host computer. At the same time as the trend towards frontend processors has become evident in the ARPANET, these devices have also achieved widespread acceptance in the commercial market place. It is during this period of time that IBM introduced the 3704 and 3705 communications processors. (Of course, as we have noted earlier, the influence of the ARPANET has been seen in the introduction in the commercial market of packet-switching nodes, network monitoring centers, and network-oriented terminal controllers).

2.1.3.5 Communications Protocols

Communications protocols are designed to provide for the orderly exchange of data between computing equipment. They make possible logical communications over a given physical path. Protocols operate in three basic modes:

1. establishing necessary conventions, such as speeds and formats

2. establishing standard communications paths, including addressing, sequencing, error control, etc.

3. establishing the standard data element to exchange, such as a character, a message, a file, etc.

The ARPANET protocols fit into this definition and have many other characteristics in common with other communications protocols. For instance, they are built on the principle of hand-shaking, in which the sender and receiver exchange messages in pairs to insure the correct receipt of each message. The ARPANET protocols were also developed using the concept of a standard so that n senders and m receivers each convert to a standard protocol which requires n + m implementations instead of adopting pairwise ad hoc communications conventions which would require n X m protocol implementations. In the ARPANET a protocol has been defined for communication between every pair of similar processes. Likewise, an interface has been defined for communication between each adjacent pair of dissimilar processes. Thus, there is a protocol between adjacent IMPs, between the source IMP and the destination IMP, between the source host and the destination host, between the sending terminal and the destination host, and so on. Also, there are interfaces between the IMP and the host, between the host operating system and the user program, etc. As is common practice in many computer systems, these different protocols have been designed modularly and in a hierarchical structure. This makes possible an independent design effort at the several levels and facilitates growth and change within each level on an independent basis. Finally, it simplifies the specification and implementation of the higher level protocols since they can use the functions of the lower level protocols instead of re-implementing them at the higher level.

2.1.3.6 Interface Characteristics

Each IMP will service up to four hosts whose cable distances from the IMP are less than 2000 feet. For greater distances, a modem channel must be used. This latter type of host connection is termed a Very Distant Host (VDH).

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

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

The bandwidth and reliability of the host connection to the ARPANET are key parameters. The issues in choosing the bandwidth of the host connections are similar to those for the network circuits. In addition to establishing an upper bound on the hosts' throughput, the rate is also an important factor in delay. The delay to send or receive a long message over a relatively slow host connection may be comparable in magnitude to the network round trip time. To eliminate this problem, and also to allow high peak throughput rates, the host connection bandwidth should be as high as possible (within the limits of cost-effectiveness), even higher than the average host throughput would indicate. By the same argument given above for packet size, a higher speed host connection allows the use of a longer message with less overhead and host processing per bit and therefore greater efficiency.

The reliability of the host connection is an important aspect of the network design; several points are worth noting. First, the connection should have a packet error rate which is at least as low as the network circuits. This can be accomplished by a highly reliable direct connection locally or by error-detection and retransmission. The use of error control procedures implies that the host-node transmission procedures resemble the node-node transmission procedures which are discussed in a later section. Second, if the host application requires extremely high reliability, a host-to-host data checksum and message sequence check are both useful for detecting infrequent network failures. Third, if the host requires uninterrupted network service, and the host is reliable enough itself to justify such service, multiple connections of the host to various nodes can improve the availability of the network. This option complicates matters for the source-to-destination transmission procedures in the nodes (e.g., sequencing) since there may be more than one possible destination node serving the host.

2.2 Topological Design

We next address the topic of topological design of the ARPANET.

2.2.1 Introduction

The design of the topological layout of a computer communications system is an important design problem. There are many different possibilities for the topological structure of a network ranging from the simple star configuration through various forms of hierarchical networks to fully connected systems and distributed topologies. The topological design problem is to find the minimum cost network layout given various requirements such as subscriber locations, traffic flows, and performance constraints. These constraints are as variable as the computer communications systems involved. Typical performance constraints include average delay, peak throughput, and average reliability. These performance constraints affect a number of the choices of the topological design including the circuit speed, block size, number of links in the connections, equipment choices, and various protocol design choices as well.

2.2.2 Centralized Network Design

We begin with the problem of centralized network design. Here one is trying to obtain the least cost connection of a set of terminals to a center. The constraints require that particular circuits have particular reliabilities or throughputs and that an overall delay requirement be met. Without such constraints this problem reduces to the minimum spanning tree problem. There are two kinds of centralized networks, the so-called star network in which all terminals connect directly to the center, and the multipoint network in which particular communications circuits may be shared by more than one terminal. Two-level networks are also used in centralized designs; here the terminals connect to some form of concentrator.

One-level centralized network design can be solved in two different ways. The optimal approach usually involves some kind of branch and bound technique in which the set of feasible solutions is partitioned into successively smaller subsets until a lower cost bound is achieved. Such techniques are often too expensive to use for large scale networks. Heuristic approaches exist for the design of such networks. The Esau-Williams algorithm proceeds by finding the node which is farthest from the center, connecting it to the adjacent node which gives the greatest cost saving, and then repeating the process. Prim's algorithm selects the node closest to the center, connects the node closest to it, and repeats growing the centralized network node by node. Finally, the Kruskal algorithm works by picking the least cost link not yet used and inserting it into the centralized network if it does not make a loop with some of the other links in the tree, It then repeats this process until a complete tree is formed,

The design of two-level centralized networks is more complicated since it is a hierarchical problem to determine how many concentrators to use, where they should be located, and which terminals should be connected to each concentrator, The connection of the terminals to the concentrators may be achieved in a star or multipoint fashion as may be the connection of the concentrators to the center, Here optimal solutions are not often employed and heuristics similar to those for one-level networks are used, For instance, the Add algorithm, which is a form of "steepest descent", proceeds by connecting all the terminals to the center and then adding the concentrator which saves the most cost and repeating the process, The so-called Drop algorithm is the inverse process, It connects all terminals to the least cost concentrator for that terminal and then removes the most costly concentrator, reconnecting the terminals to other concentrators, The process is then repeated until a lower bound is achieved,

Centralized networks which meet performance constraints as well as cost constraints can be designed using these techniques, Furthermore, these algorithms can be used in the design of the access area network in large scale distributed systems as well. To consider such systems, we begin by describing the design of the backbone net for a large decentralized network.

2.2.3 Backbone Network Design

The general problem of designing a distributed backbone for a computer communications network can be considered to be one or more of the following five subproblems in optimization:

a. Node Location - Locate the switching nodes in the computer network.

b. Capacity Assignment - Given the traffic flows and network topology, minimize average message delay with respect to channel capacity.

c. Flow Assignment - Given link capacities and network topology, minimize average delay with respect to traffic.

d. Capacity and Flow Assignment Problem - Given the network topology, minimize the average delay with respect to channel capacities and traffic.

e. The Topology, Capability and Flow Assignment Problem - Minimize average delay with respect to network topology, capacity, and traffic.

The capacity assignment problem can be solved exactly when channel costs are a linear function of bandwidth. Other solutions can be obtained for channel costs which are logarithmic or power law costs. For step-function costs, which are more common in practice, heuristic solutions are necessary.

The flow assignment problem can be solved by the flow deviation method which is an optimal procedure resulting in an alternate routing flow. It begins by establishing a feasible starting flow of traffic through the network. It then solves the shortest route flow problem by assigning lengths to each of the branches associated with the increase in delay due to additional traffic on that link. The algorithm iterates by assigning more traffic to paths which represent the least additional delay paths to which some flow can be deviated.

When these two problems are combined to form the capacity flow assignment problem, globally optimal solutions are no longer possible. Instead, local optimum solutions are obtained by beginning with a feasible starting flow calculating optimum capacity assignment under linearized costs, carrying out the flow deviation algorithm to find the optimum flows, repeating the capacity assignment problem for these new flows and continuing to iterate between these two solutions until a local minimum is found.

The final problem is to determine the topology as well as the capacity and flow assignment. In this case an iterative algorithm known as concave branch elimination can be used. It works by selecting an initial topology and then carrying out the capacity flow assignment algorithm. From a suboptimal solution this algorithm picks some specific capacities and conducts a final flow optimization by an application of the flow deviation algorithm. Then it repeats these steps for a number of feasible starting flows. Finally, it repeats this entire algorithm for a number of initial topologies.

2.2.4 General Network Design

The overall design of a general computer communications network incorporates elements of each of the procedures that we have described so far. One begins by locating backbone nodes in the network. Then the backbone switching network is laid out using techniques such as the concave branch elimination algorithm and subalgorithms for determining capacities traffic and topology. A third step is to identify access area clusters of subscribers and to connect these subscribers to backbone nodes by means of one-level or two-level local centralized networks. Then one can repeat the design of the backbone and access area network with different starting assumptions or constraints until a satisfactory network design is achieved. The final two steps are to identify the optimal carrier offerings with which to construct the communications network and the optimal communications equipment with which to build the network.

2.3 Communications Equipment

In this section of our report we discuss the many different types of equipment which are contained in the ARPANET including the circuits themselves, the IMPs, TIPs, the NCC, and various other pieces of equipment such as the PLI, the RJE, etc.

2.3.1 The Network Circuits

Each IMP in the ARPANET can connect to up to four telephone lines. Normally, these are 50 Kb per second circuits. These lines are provided by the Bell System as part of their wideband data services. A 303-type data set is available as part of the terminal equipment for this facility. The physical connection to the local toll office is a twisted pair which resides in a cable containing about 1000 other twisted pairs. The line is equalized. Long haul circuits are used to connect distant locations. In that case, incoming data at the local toll office is frequency-multiplexed and transmitted wideband over the long haul circuits as an analog signal with no coding. Approximately two thirds of the long haul connections in the Bell System use microwave relay stations and the remainder use repeatered coaxial cables. The 50 Kb per second group facility can be operated in a synchronous or a nonsynchronous mode. The ARPANET uses the wideband service in the synchronous mode.

The 303-type wideband data station has been engineered according to the Bell System specifications to achieve a bit error probability on the order of one in 100,000. Errors occur for two basic reasons, impulse noise and momentary dropouts. Impulse noise seems to arise from strong field interferences such as those caused by large currents which flow in switching centers. An even more important cause of errors is momentary dropouts. These appear primarily on the long haul circuits which are known to be subject to conditions which result in the occasional presence of a low signal-to-noise ratio circuit. These conditions are generally referred to as "fades." The Bell System monitors their channels for fading conditions and provides for diversity in the event of a fade. Both types of errors tend to cause bit errors which occur in bursts.

Published experimental data on the nature of errors on leased private lines in the telephone plants indicates that an average of one block of 511 bits in a few thousand blocks will not contain errors at all. Each block which contains errors will typically have errors occurring in bursts, and a block containing errors is also likely to have neighboring blocks containing errors. The following assumptions were made for the purpose of obtaining estimates on line errors in the design of the ARPANET: On a telephone line approximately one block is in error every three minutes on the average. This corresponds to approximately one block of 511 bits in error every 1000 blocks. To first order, the design of the ARPANET assumed that the error occurrences in time on the wideband facilities were approximately the same as those on the telephone lines. This means that an error every three minutes on the wideband service corresponds to one block of 1000 bits in error approximately every 10,000 blocks on the average at 50 Kbs. This also means that the distribution of error bursts over the wideband service differs from that on the telephone lines, whereas the median burst length on the telephone line is approximately 4 bits. The median burst length over the wideband service is larger by a factor between two and ten.

One of the important design objectives for the ARPANET was to establish essentially error-free communications so that any system difficulties could be effectively and rapidly isolated. The error control scheme was designed to provide an average time between undetected errors in the communications subnetwork on the order of at least one or two years. The time between undetected errors is a random variable and therefore can have values which are lower than the average. An average time between errors of one or two years allows a margin of safety which constitutes a reasonable design specification. The estimates of the network designers were that a 16-bit parity check would give an average time between undetected errors on <u>each line</u> in the ARPANET of between half a year and five years. With a 24-bit parity check, the estimate was that the time between undetected errors on each line would be between 10 and 100 years. In a network with between 10 and 100 lines this gives an average time between undetected error for the entire network of at least one year.

There are two basic kinds of error detection mechanisms, cyclic parity checks and lateral and longitudinal parity checks, also called iterative checks. Cyclic parity checksums have been employed in the detection of errors on telephone lines, whereas iterative checksums have been used on magnetic tapes. The ARPANET design was based on the use of a BCH code generated by a 24-bit shift register corresponding to the generator polynomial

The reason for selecting BCH codes was that the distant structure of these codes is better understood than most and they exhibit uniformly wide code word separation. This means they have proved to be satisfactory choices in detecting errors in many applications.

After the first four IMPs were delivered in 1969, a series of phone line tests were run on the initial ARPANET. The first test program continuously transmitted short packets on each phone line using looped telephone circuits in order to perform two-way tests. A 27-hour test indicated approximately one packet per

20,000 in error. The distribution of error bursts was concentrated in the range between one and seven packets. Subsequent tests were performed with much larger packets and over other lines for long periods of time. No errors were ever detected in the data by the phone line test program which had not already been detected by the cyclic checksum hardware.

In the third quarter of 1970 when the ARPANET included 11 IMPs, the network was first tested with a 230.4 Kb per second circuit. Subsequently, a number of such high speed circuits have been introduced in the network primarily for short distance links. These circuits are controlled by the IMP program without any special changes in software or hardware.

In the summer of 1973 the Norway TIP was connected by a 9.6 Kb per second satellite circuit to the rest of the ARPANET in the U.S. The low speed circuit required some changes to the IMP software to allow for slower propagation times for routing and to allow for more packets in flight over the satellite channel.

The use of a satellite channel in a packet-switching environment introduces some important new constraints to the method used to allocate satellite capacity. For example, in the SPADE system, channels are allocated to voice conversations which are usually several minutes long. The minimum time required to allocate a channel pair is approximately one quarter of a second, which is also the minimum time required to deallocate the channels. Allowing for additional delays due to allocation conflicts, the total allocation-deallocation time is on the order of one second, an insignificant fraction of the conversation time.

In contrast, the duration of a data packet in the ARPANET is comparable to the duration of an SPADE allocation request burst. Thus if SPADE data channels were switched on a packet basis, each channel would be idle for at least the minimum .5 second allocation-deallocation time and utilized for only about .02 seconds assuming maximum length packets. This is a utilization of only 4%.

An important additional consideration for packet-switching traffic is the different delay requirements for different traffic classes. From these considerations it is clear that if only single-packet messages are to be sent, it is best simply to send each directly with a destination address in an unused slot rather than sending a call request and release before and after it. On the other hand, for multi-packet traffic it becomes more efficient to first send requests for slots.

Several different approaches to the allocation of a satellite channel on a dynamic basis have been developed. These have been based on multi-access and broadcast modes of packet delivery via a destination address in each burst making use of the broadcast feature in varying degrees to accomplish dynamic allocation of the channel.

The random access method also known as ALOHA is a pure contention scheme. Other systems have been developed combining TDMA with reservation-based dynamic allocations. Both of these systems have been tested in the ARPANET satellite IMPs using an INTELSAT-IV 56 Kb per second digital channel.

The ramifications of the use of satellite channels for network trunks are considerable. Any satellite channel represents a path of higher delay and also potentially much higher bandwidth. If the channel is used in broadcast mode, then the delay and bandwidth characteristics of the link are more difficult to estimate precisely. In general, it is best to send short interactive traffic over terrestrial lines and long bulk data transfers over satellite paths.

The introduction of satellites into the system has a direct impact on nodal buffering. First of all, the IMPs connected to the satellite need much more packet buffering than those connected to land lines since many more packets can be in flight. Another important requirement and one which is less obvious is that all IMPs in the network need more memory for end-to-end message buffering when a satellite is added not just those IMPs directly directed connected to it. The IMPs may also require special acknowledgment schemes to keep track of the large number of packets in flight as well as changes in the host-to-host protocol and in other procedures.

In summary, the ARPANET was developed initially on the basis of 50 Kb per second terrestrial circuits. In subsequent years, it has incorporated lower speed and higher speed circuits and satellite channels all without major change to the original design.

2.3.2 The IMP Hardware and Software

The first packet-switch in the ARPANET series was based on the Honeywell 516 computer. The second packet-switch in the series was developed in 1971, based on the Honeywell 316 (the successor to the 516). This second machine was developed primarily to provide a less expensive version of the original switch. The choice of the 316 as successor to the 516 permitted this second version to be developed with minimal reprogramming, although hardware "specials" in the system had to be reengineered.

The experience of operating the ARPANET produced many insights into just what characteristics make a machine more or less suitable for use in a packet-switch. On the basis of this experience, a new machine called the Pluribus, was designed and built with the specific requirements of a packet-switching node in mind from the outset.

In Section 2.3.2.1, we discuss a number of packet-switching network considerations and their influence on the design of the packet-switching computer itself. In the remainder of Section 2.3.2, we describe the hardware and software structures of the 516/316 packet-switch. The Pluribus-based switch is described in Section 2.3.5.

2.3.2.1 Network Considerations

The speed of the packet-switch's processor is an important determinant of the throughput rates possible in the network (the effect of processor speed on delay is of much less concern since processing delays are typically less than a millisecond). The store-and-forward processing bandwidth of the processor can be computed by counting instructions in the inner program loop. The source-to-destination processing bandwidth can be calculated in a similar fashion. These rates should be high enough so that the entire bandwidth of the network lines can be used, i.e., so that the node is not a bottleneck. Experience indicates that the instruction cycle time is the main factor in this bandwidth calculation; complex or specialized instruction sets would not be particularly valuable because simple instructions make up most inner loops--at least this is true in the systems we have built.

A different aspect of the packet-switch's processor which can also affect throughput is its responsiveness. Because circuits are synchronous devices, they require service with very tight time constraints. If the switch does not notice that input has completed on a given circuit, and does not prepare for a new input within a given time, the next input arriving on that circuit will be lost. Similarly, on output the switch must be responsive in order to keep the circuits fully loaded. This requirement suggests that some form of interrupt system or high-speed polling device is necessary to keep response latency low, and that the overhead of an operating system and task scheduler and dispatcher may be prohibitive. Finally, we note that the amount of time required by the switch to process input and output is most critical in determining the minimum packet size, since it is with packets of this size that the highest packet arrival and departure rates (and thus processing requirements) can be observed. Of course, data buffering in the device interfaces can partially alleviate these problems.

The speed of memory may be a major determinant of processor speed, thus affecting the switch bandwidth. An equally important consideration is memory speed for I/O transfers, since the switch's overall bandwidth results from a division of total memory bandwidth based on some processing time for a given amount of I/O time. First, there is the question of whether the I/O transfers act in a cycle-stealing fashion to slow the processor or whether memory is effectively multi-ported to allow concurrent use. Second, there is the issue of contention for memory among the various synchronous I/O devices. In a worst-case scenario, it is possible for all the I/O devices to request a memory transfer at the same instant, which keeps memory continuously busy for some time interval. A key design parameter is the ratio of this time to the available data buffering time of the least tolerant I/O device. This ratio should be less than one, and may therefore determine how much I/O can be connected to the node.

The size of the memory, naturally, is another key parameter. In our experience, with a terrestrial network the switch program and associated data structures take up the majority of the switch's storage; this may change with the introduction of high-speed satellite circuits. The remainder of the switch's memory is devoted to buffering of two kinds: packet buffering between adjacent nodes, and message buffering between source and destination nodes. These requirements can be calculated quite simply in each case as the product of the maximum data rate to be supported times the round trip time (for a returning acknowledgment). In large networks it may be necessary to rely on sophisticated compression techniques to ensure that tables for the routing algorithm, the source-to-destination transmission procedures, and so on, do not require excessive storage.

The speed of the I/O system has been touched upon above in relation to processor and memory bandwidth. Other factors worth noting are the internal constraints imposed by the I/O system itself--its delay and bandwidth. A different dimension, and one that we have found to be inadequately designed by most manufacturers, is the flexibility and extensibility of the I/O system. Most manufacturers supply only a limited range of I/O options (some of which may be too slow or too expensive to use). Further, only a limited number of each type can be connected. A packet-switch requires high performance from the I/O system, both in the number of connections and in their data rates.

There are other factors to consider in evaluating or designing a computer for the packet-switch function apart from performance in terms of bandwidth and delay. As we mentioned, extensibility in I/O is very important and was comparatively rare until recently; it is more common to find memory systems which can be expanded. Processor systems which can be expanded are not at all common, and yet processor bandwidth may be the limiting factor in some switch configurations. Without a modular approach allowing processing, memory, and I/O growth, the cost of the switch can be driven quite high by large step functions in component cost.

A final aspect of switch computer architecture is its reliability, particularly for large systems with many lines and hosts. A failure of such a system has a large impact on network performance. Computer manufacturers tend to cut corners in order to compete on price and delivery schedules. The penalty for this practice is usually paid in the coin of lowered reliability. These issues of performance, cost, and reliability become critically important in large networks serving thousands of hosts and terminals on a 24-hour-a-day basis.

2.3.2.2 The 516/316 IMP Hardware

The 516 and 316 IMPs can be considered to be the same machine. The 316 derivative is less expensive, smaller, effectively 30% slower, and less tightly engineered than the original ruggedized 516s. Architecturally, however, the machines are very similar. They share the following characteristics: 1) simple processor with 16-bit word length, 512-word pages, single accumulator, approximately 1 microsecond cycle time, one index register, indefinite indirect addressing, power-fail, and auto-restart; 2) 16K words of memory which can be conveniently addressed; 3) a multiplexed 16-channel direct memory access unit for Input/Output; 4) a 16-level priority interrupt system; and 5) a Teletype for maintenance and local debugging.

To the basic machine are added a number of special hardware features which suit it to the IMP job, specifically:

Full-duplex, synchronous interfaces are provided for connecting the IMP to the communication lines. These interfaces are clocked by the modems at the bit level. They are treated by the program at the packet level (that is, they are direct access devices which send and receive blocks of words from storage, and use interrupts to notify the program of the completion of each packet transfer). The jobs of generating inter-packet (idle) "sync" characters, providing packet framing characters, fetching successive words of a packet from memory via a memory channel, serializing and deserializing all characters to and from the modem, formulating and checking a 24-bit cyclic redundancy check, detecting overflow and format errors, and signaling the completion interrupt are all handled by the interface.

Full-duplex, asynchronous interfaces connect the IMP to one or more host computers. Connecting an IMP to a wide variety of different hosts requires a hardware interface, some part of which must be custom tailored to each host. A standard portion of the interface is built into the IMP, identical for all hosts (it is a direct memory channel device like the modem interface), while a special portion must be uniquely designed for each different host type. A bit serial interface is used partly because it is less expensive and partly to accommodate conveniently the variety of word lengths in the different host computers. The interface operates asynchronously, each data bit being passed across the interface via a Ready For Next Bit/There's Your Bit handshake procedure. This technique permits the bit rate to adjust to that of either member of the pair and allows for pauses when words must be stored into or retrieved from memory.

A relative time clock provides an interrupt every 25.6 microseconds for performing all time-dependent tasks such as timing out packets for retransmission. Programmed counters are run off this clock to perform still lower frequency periodic jobs.

A watchdog timer is provided which is normally held off (i.e., restarted) by the program. The timer is restarted every time a correct response to a (periodic) "trouble report" is returned from the net indicating that most of the basic system is operating properly. If the timer ever runs out, a failure of some sort is indicated and a reload-and-restart program is activated.

A program-generatable interrupt, known as the task interrupt, permits program-generated tasks to be queued and handled by the same priority-ordered, interrupt-driven program structure as hardware (I/O) generated tasks.

A program-readable hardware-held register identifies the number and configuration of the machine.

It is illuminating to consider the impact of some detailed differences between the 516 and 316 machines. First, although the 516 basic cycle is .96 microsecond and the 316 cycle is 1.6 microseconds, the real traffic-handling capabilities of the IMP are not in this simple ratio since other factors effect it. In the 516, I/O channel pointers are kept in memory, while in the 316 they are in hardware. The result is that each I/O word transferred in the 516 takes about 4 microseconds while in the 316 it takes 3.2 microseconds. Thus for the same amount of I/O traffic, the 516 has less real time left for program execution, but it does instruction faster. In fact, for the IMP task, although one must consider the memory access time used for I/O transfers, the processing bandwidth tends to predominate. The relative importance of processing time to I/O time, however, will vary for a number of reasons. For instance, packet processing is less costly per bit for longer packets than short, whereas I/O transfer time per bit does not change. Second, in the 516 the watchdog timer is a hardware timer; in the 316 the timer is a software counter run off the relative time clock interrupt routine. Neither timer can really verify that all features of the program are operating properly. The 316 timer is vulnerable to a break in a tiny piece of the clock code but, by using the relative time clock, it eliminates the need for a seperate hardware timer. We make up for this apparent deficiency by checking in other places in the code to see if the clock code is running, and if it is not, a reload is initiated. Several other key data and control structures are tested in a similar fashion.

There are several other features which differ between the two machines but which have not turned out to be significant: 1) the 516 has a 512-word block of protected memory, originally built in for protecting recovery and startup programs, while the 316 does not have this feature; 2) the halt instruction in the 516 can be inhibited (turned into a NOP) so that interrupts will always be able to force control to interrupt serving routines while the 316 simply halts in all cases; and 3) the 516 has the option of halting or interrupting on power fail while the 316 is only able to interrupt.

2.3.2.3 The 516/316 IMP Software

Implementation of the IMPs required the development of a sophisticated operational computer program and the development of several auxiliary programs for hardware tests, program construction, and debugging. This section discusses the design of the operational program and describes the auxiliary software.

2.3.2.3.1 General Descriptions

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

The entire program is composed of fifteen functionally distinct routines; each piece occupies no more than two or three pages of core (512 words per page). These routines communicate primarily through common registers residing in page zero of the machine which are directly addressable from all pages of memory. A typical map of core storage is shown in Figure 2-2. By use of

Figure 2-21 Typical Map of Core Storage

a macro, code is "centered" on each physical page at assembly time such that there is an integral number of buffers between the last word of code on one page and the first word of code on the next. This technique eliminates all breakage (unusable core) except for part of one buffer on the very last page of core. Seven of the fifteen programs are directly involved in the flow of packets through the IMP: the task program performs the major portion of the packet processing, including the reassembly of Host messages; the modem programs (IMP-to-Modem and Modem-to-IMP) handle interrupts and the resetting of buffers for the Modem channels; the Host programs (IMP-to-Host and Host-to-IMP) handle interrupts and resetting of buffers for the Host channels, build packet headers during input, and construct allocation requests sent to the destination IMPs; the timeout program maintains a software clock, times out unused buffer allocations, reinitiates programs which have paused, and initiates routing computations and other relatively infrequent events. A background loop contains the remaining major programs and deals with initialization, debugging, testing, statistics gathering, and tracing. Background programs also initiate RFNM allocation and other sequencing and control messages. After a brief description of data structures, we will discuss packet processing in some detail.

2.3.2.3.2 Data Structures

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

Buffer Storage. The buffer storage space consists of about 52 fixed length buffers, each of which is used for storing a single packet. An unused buffer is chained onto a free buffer queue and is removed from this list when it is needed to store an incoming packet. A packet, once stored in a buffer, is never moved. After a packet has been successfully passed along to its Host or to another IMP, its buffer is returned to the free list. The buffer space is partitioned in such a way that each process (store and forward traffic, Host traffic, etc.) is always guaranteed some buffers. For the sake of program speed and simplicity, no attempt is made to retrieve the space wasted by partially filled buffers.

In handling store and forward traffic, all processing is on a per-packet basis. Further, although traffic to and from Hosts is composed of messages, the IMP converts to dealing with packets; the Host transmits a message as a single unit but the IMP takes it one buffer at a time. As each buffer is filled, the program selects another buffer for input until the entire message has been provided for. These successive buffers will, in general, be scattered throughout the IMP's memory. An equivalent inverse process occurs on output to the Host after all packets of the message have arrived at the destination IMP. No attempt is made to collect the packets of a message into a contiguous portion of IMP memory. A typical allocation of buffer space in core storage is shown in Figure 2-2, as mentioned previously. IMPs with no Very Distant Host use the space on pages 35, 36 and 37 for buffer storage. All IMPs reserve the last 71 words for saving local data over reloads and for linkage to satellite and Terminal IMP programs.

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

-- Each line must be able to get its share of buffers for input. Double buffering is provided for input on each line, which permits all input traffic to be examined by the program. Thus, acknowledgements can always be processed, which frees buffers.

-- An attempt is made to provide enough store-and-forward buffers so that some lines may operate at full capacity. The number of buffers needed depends directly on line distance and line speed. The current limit is 1-1/2 times the number of channels per line, thus permitting 1-1/2 lines on the average to be operating at full capacity. Furthermore, each output line is guaranteed at least one buffer, thus permitting a low level of traffic on any line independent of congestion on other lines.

-- All remaining free buffers may be claimed for reassembly storage, including an overlap into the store-and-forward allocation dependent upon the number of lines. All IMP processes (except for modem input) must share this storage using a priority scheme to resolve contention and preclude "deadly embrace" - type storage lockups.

Buffers currently in use are either dedicated to an incoming or outgoing packet, chained on a queue (or pointed to within a table) awaiting processing by the program, or being processed. Occasionally, a buffer may be simultaneously in several of these states, due to parallelism in the program. A "use count" is maintained within each buffer, and only when this count goes to zero is the buffer put back on the free queue.

<u>Queues</u>. There are three principal types of queues:

-- Task: All routing packets, all packets from the modems and all packets received on Host channels are placed on the task queue.

-- Output: A separate output queue is constructed for each inter-IMP modem circuit and each Host. Each modem output queue is subdivided into a priority queue and a regular message queue, which are serviced in that order. Each Host output queue is subdivided into a control message queue, a priority queue, and a regular message queue, which are also serviced in the indicated order.

-- Reassembly: The reassembly queue contains those packets being reassembled into messages for the Host.

Tables. Tables in core are identical for all IMPs, and their size is determined either by fixed IMP parameters or processing capability and network performance considerations. In the former category, there are per-IMP (67 word) tables for routing and statistics data; per-Host (8 word) tables for queue pointer, status, and local data required by reentrant Host code; and per-line (5 word) tables similar in function to the Host tables. In the latter category, all tables are pooled resources which are acquired and relinquished by IMP processes depending on the activity required of them. These include transmit and receive message blocks, reassembly blocks, trace blocks, transaction blocks, and initialization data.

The size of the initialization code and the associated tables deserves mention. This was originally quite small. However, as the network has grown and the IMP's capabilities have been expanded, the amount of memory dedicated to initialization has steadily grown. This is mainly due to the fact that the IMPs are no longer identically configured. An IMP may be required to handle a Very Distant Host, or TIP hardware, or five lines and two Hosts, or four Hosts and three lines, or a very high speed line, or a satellite link. As the physical permutations of the IMP have continued to increase, the criterion followed has been that the program should be identical in all IMPs, allowing an IMP to reload its program from a neighboring IMP and providing other considerable advantages. However, maintaining only one version of the program means that the program must rebuild itself during initialization to be the proper program to handle the particular physical configuration of the IMP. Furthermore, it must be able to turn itself back into its nominal form when it is reloaded into a neighbor. All of this takes tables and code. Unfortunately, the proliferation of IMP configurations which has taken place was not foreseen; therefore, the program differences currently cannot be conveniently computed from a simple configuration key. Instead, the configuration irregularities must be explicitly tabled.

2.3.2.3.3 Packet Flow Through Major IMP Routines

Figure 2-3 is a schematic drawing of packet processing. The processing programs are described below. Packet flow may be followed by referring to Figure 2-3.

Figure 2-3: Packet Flow and Processing

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

The Modem-to-IMP routine (M-I) handles inputs from the modems. This routine first sets up a new input buffer, normally obtained from the free list. If a buffer cannot be obtained, the

received buffer is not acknowledged and is reused immediately. The discarded packet will be retransmitted by the distant IMP. The routine processes returning acknowledgements for previously transmitted packets and either releases the packets to the free list or signals their subsequent release to the IMP-to-Modem routine. The M-I routine then places the buffer on the end of the task queue and triggers the programmable task interrupt.

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

If a packet from a modem line is addressed for local delivery, its message number is checked to see whether a duplicate packet has been received. As mentioned previously, each IMP maintains for each connection a window of contiguous numbers which it will accept from the other side of the connection. Packets with out-of-range numbers are considered duplicate and are discarded. The receipt of a RFNM for the oldest message by the source Host permits the window to be moved up by one number.

Replies such as RFNMs or Dead Host messages are placed in transaction blocks. TASK then pokes the IMP-to-Host routine to initiate output to the Host.

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

Incoming routing messages are processed so that outgoing routing messages and the routing directory immediately reflect any new information received. The task routine generates I-heard-you messages in response as necessary to indicate to the neighbor receipt of the routing message.

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

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

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

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

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

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

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

The PACKET CORE program loads and dumps portions of its own IMP's core memory, or acts as an intermediary in loading and dumping portions of the core memory belonging to a neighbor who is unable to communicate via the normal IMP-IMP protocol. The PACKET CORE facility allows for dissimilar machines to coexist as IMPs on the network; reloading and diagnostic dumping of a malfunctioning IMP can be done without the requirement that one of its neighbors be of the same machine type.

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

Other routines, which send connection protocol messages, send incomplete transmission messages, send allocations, return givebacks, send RFNMs, and retransmit single-packet messages also reside in the background program. These routines are called Back Hosts. However, these programs run in a slightly different manner than the fake Hosts in that they do not simulate the

Host/IMP channel hardware. They do not go through the Host/IMP code at all, but rather put their messages directly on the task queue. Nonetheless, the principle is the same.

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

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

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

2.3.2.3.4 Control Organization

It is characteristic of the IMP system that many of the main programs are entered both as subroutine calls from other programs and as interrupt calls from the hardware. The resulting control structure is shown in Figure 2-4. The programs are arranged in a priority order; control passes upward in the chain whenever a hardware interrupt occurs or the current program decides that the time has come to run a higher priority program, and control passes downward only when the higher priority programs are finished. No program may execute either itself or a lower priority program; however, a program may freely execute a higher

Figure 2-4: IMP Program Control Structure priority program. This rule is similar to the usual rules concerning priority interrupt routines.

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

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

All interrupt levels (except modem to IMP) make use of interrupt entry and exit routines that save the interrupted state variables on a stack. The six variables saved are: the index register, the accumulator, the keys (overflow bit, memory mode, etc.), the checksum routine return address, the interrupt mask, and the interrupt return address. The checksum routine is therefore reentrant at each separate interrupt level and can be called with interrupts enabled. For a slight improvement in efficiency, the Modem-to-IMP routine does its own saving and restoring of state variables.

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

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

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

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

-- The Modem-to-IMP routine can deliver its current packet to the task routine before the next packet arrives and can always prepare for successive packet inputs on each line. This timing is critical because a slight delay here might require retransmission of the entire packet.

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

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

2.3.3 The TIP Hardware and Software

The Terminal Interface Message Processor (TIP) provides a means for connecting up to 63 terminal devices to the ARPA Network. The terminal interface specification conforms to the EIA standard RS232C, which permits direct connection to most data modems. In addition to full duplex, serial data transmission, each of the 64 ports provides 4 program-settable control lines and monitors 6 external status lines; these lines are useful in dealing with modems or other compatible I/O devices. Data format is Teletype compatible, that is, character oriented with start and stop bits. The TIP handles all routine operations of timing and sequencing. All line parameters, such as speed and character size, are program settable.

2.3.3.1 The TIP Hardware

The TIP is built around a Honeywell H-316 computer with 28K of core. It embodies a standard 16-port multiplexed memory channel with priority interrupts and includes a Teletype for debugging and program reloading. Other features of the standard IMP also present are a real-time clock, power-fail and auto-restart mechanisms, and a program-generated interrupt feature. As in the standard IMP, interfaces are provided for connecting to high-speed (50-kilobit, 230.4-kilobit, etc.) modems as well as to Hosts.

Aside from the additional 12K of core memory, the primary hardware feature which distinguishes the TIP from a standard IMP is a Multi-Line Controller (MLC) which allows for connection of terminals to the IMP. Any of the MLC lines may go to local terminals or via modems to remote terminals. As shown in Figure 2-5 the MLC consists of two portions, one a piece of central logic which handles the assembly and disassembly of characters and transfers them to and from memory, and the other a set of Line Interface Units (all identical except for a small number of option jumpers) which synchronize reporting to individual data bits between the central logic and the terminal devices and provide for control status information to and from the modem or device. Line Interface Units may be physically incorporated one at a time as required.

Figure 2-5 shows the hardware configuration of the TIP. The upper part of the figure shows the essentially standard 316 IMP contained in a TIP which runs the standard IMP program. This program includes a test for the presence of a Multi-Line Controller. If an MLC is attached, the program activates a terminal-handling program which controls the MLC.

The lower portion of Figure 2-5 shows the Multi-Line Controller. The MLC provides interfaces for up to sixty-four full duplex connections. These lines may be clocked either

Figure 2-5 TIP Hardware Configuration

externally to the TIP, in which case they are called externally clocked, or internally by the TIP, and called asynchronous. Any of these connections or ports (as they will be referred to hereafter) may transmit and receive up to 19.2 Kbps when operating externally clocked. The bit rates for the internal clocks range from 75 baud to 2400 baud and are shown in Table

0	ILLEGAL
1	75
2	110
3	134
4	150
5	300
6	600
7	1200
10	1800
11	2400
12	4800 (OUTPUT ONLY)
13	9600 (OUTPUT ONLY)
14	19200 (OUTPUT ONLY)
15	ILLEGAL
16	ILLEGAL
17	SYNCHRONOUS

MLC CLOCK RATES

Table 2-3

2-3.

All of the circuitry specific to a single port is contained on a plug-in module called a Line Interface Unit or LIU. Although the input and the output for a given port are usually assigned to the same device, they may go to completely separate devices such as a card reader and a line printer. All such parameters as bit rate and character size are individually settable under program control for each port and are independent for the input and the output of a given port.

In addition to circuitry on the LIU, each port uses logic which is shared by all lines. This is called the common logic. The common logic contains the DMC interfaces, the timing circuitry, and a 75-bit by 64-word memory. Each of the 64 device ports is assigned a unique memory location called the STATEWORD for that line. STATEWORD records the instantaneous state of data and control transmission on a line. Part of the memory is realized as a serial, pseudo-drum memory built from circulating shift registers. This 57-bit portion is shown schematically in Figure 2-6. The remaining 18 bits of the STATEWORD are stored on a per-line basis on the individual LIUs. The pseudo-drum rotates once per 51.2 microseconds which permits a complete revolution per bit at 19.2 kilobits per second. Thus the MLC handles one bit per revolution at 19.2 Kbps, one bit per 2 revolutions at 9.6 Kbps and so forth.

All external circuitry is located on the Line Interface Unit. The LIU contains circuits which generate signals which may not be time multiplexed, i.e., turned on and off as the pseudo-drum rotates. Such signals are the output data line and

Figure 2-6 Stateword Pseudo-drum Memory

modem control lines. The LIU contains a portion of the STATEWORD, including the output data synchronizers, the output clock synchronizers, control signals for an I/O device or an optional modem, the input data synchronizer, and the input clock synchronizers.

The format for all data which will be used as input for a given port of the MLC must be either five, six, seven or eight-bit character. Each character must also be preceded by a start bit and followed by a stop bit. Essentially, characters must be compatible with standard Teletype format. The interface voltage levels conform to EIA Specification RS232 and the interface connector is the EIA standard dataphone connector. In the standard configuration the TIP is available with up to sixteen internally mounted modems of the 102, 201, or 202 variety.

For externally clocked data (that is, clock signal provided by the modem or the attached device), when the receive clock makes a negative transition, the input side of the device port will read the state of the device data line. For externally clocked output, the device port must change data in response to positive transitions of the external clock line. For data input where there is no external clock line to tell exactly when to sample the data line, the eight-phase bit-sampling technique, which employs one of the internally generated clock frequencies, is used. On internally clocked output, no constraint is imposed on when a character may start; however, once a character has started and generated a start bit on the outgoing data line, it will proceed at the specified and well-defined rate. Onset of the next character may bear no relationship to the present character. It is for this reason that this mode is normally called asynchronous.

For operation of the device control and status monitoring, ten lines are available. There are four control lines which are set under program control; these conform to EIA Spec. RS232, wherein a positive voltage is a function-true operation. There are also six status lines from the modem into the Multi-Line Controller. They may be read at any time under program control, and they are defined as for the control lines: positive voltage equals a function-true condition.

This control status information is communicated to and from the 316 computer via the accumulator input-output bus, which has an average delay of 25 microseconds and a worst-case delay of 50 microseconds when used through the Multi-Line Controller. To these delays must be added any delays due to the operation of the TIP program. Consequently, this is not a high-speed data path and is used only for control information at a low rate.

The next section describes the manner in which serial bit streams from a device port are assembled into 8-bit characters and subsequently loaded into the 316 computer memory. The inverse operation of accepting 8-bit characters from the memory and disassembling them into bit streams for the various ports is also described.

The Multi-Line Controller communicates with the 316 via memory (i.e., DMC) channels, program-generated control signals, accumulator I/O instructions and priority interrupts. The program exchanges characters with the MLC through buffer tables in the memory, accessed via the DMC. One DMC channel is used for input and two for output. Two priority interrupt lines are used: one to indicate that the output buffer table has been emptied, and one to give a periodic interrupt which causes the program to check for input and for output requests (see below for details).

The program can set certain bits in the MLC for each line. For every line's input section, the program sets character size, line rate and whether or not the line is of high input priority. These bits are set through an instruction which is addressed to the controller and whose data word designates the input section and includes the line number as well as the specific parameter values. This is used to initialize each input line on startup.

The same instruction with the data word designating the output section is used to initialize the output lines. This command sets the output line speed to the appropriate value.

Instructions are provided for starting and stopping transfers by the controller on the DMC channels. For example, when the program wishes to look at the input table, it executes an Input Disable causing the controller to inhibit further requests on the input channel. After providing a new buffer it executes an Input Enable which restarts the input transfers. Similar instructions are provided for controlling MLC access to a memory table where per-line requests for output characters are reported to the program.

The OUTPUT table, which contains outgoing characters to terminals, is reestablished only when the table is completely empty (program-controlled overall stopping of output is not provided), so only the Output Enable is required.

Figure 2-7 shows the Input Data Flow. For each line, the MLC provides hardware storage for the character presently being shifted in and for the previous character. The serial bit stream enters a shift register called INDAT. When a character is fully assembled, it is moved to a buffer, BINDAT, which is then marked as full. Each line has a private INDAT shifter and BINDAT buffer. During the period that a character is being shifted into

INDAT, the previous character in BINDAT must be unloaded into the 316 memory. Because of variations in character length and particularly in line speeds, the amount of urgency in emptying BINDAT varies from line to line. The input DMC channel is used to pump the characters from the BINDATs into the memory. If BINDATs were serviced solely on a first-come-first-served basis, an urgent high speed line could get locked out by less urgent low speed lines. Worst case memory accessing is further complicated by higher priority DMC channels and by the fact that the 64 BINDATs are not randomly available but are scanned in sequence (one every 800 ns). Lines operating at under about 3000 bits/sec, even in the worst case with the lockout by all other lines and the expected interference from Hosts (assumed 100 Kb) and modems (assumed 50 Kb), will get their BINDATs empty in time for the next character. Faster lines can be locked out until too late (i.e., until after their BINDAT is needed for the ensuing character). To avoid this, two buffer registers are provided for entering characters into the single input DMC channel. "Fast" lines use one of the buffers (FINBUF) and normal lines use the other (INBUF). Lines are marked in the MLC by the program as FAST or not.

Input characters are dumped into a tumble table. Each entry consists of the character, the line number (0-63), and a bit that indicates whether the input side of the line is in BREAK mode.

Figure 2-7 Input Data Flow

When the program is ready to start accepting input, it establishes an input buffer (sets pointers) for the DMC input channel and executes an Input Enable. It then sets up character size, line rate, and "Fast" priority for each line in turn. The lines are now active; characters come into the controller and are stored by it into the input table in core.

Periodically (every 3.3 milliseconds), a clock interrupt rouses a routine which checks to see whether or not any characters have been received. The buffer provided is sufficiently long that it should never be filled by the hardware between periodic looks. In order to check for input, the program blocks input on that DMC channel and causes the controller to rewrite the DMC pointers in the memory where the program can look at them. The program saves the pointer values and then resets the pointer and executes another Input Enable, which reopens the door for further input.

Figure 2-8 shows the Output Data Flow. The MLC contains a shift register and a character buffer for each output line. One of these, OUTDAT, contains the character presently being shifted onto the line; the other, BOUTDAT, contains the next character to go. As soon as the character in OUTDAT has been completely shifted onto the line, the character in BOUTDAT is moved into OUTDAT and in turn starts to be shifted out. A full character time is thus available for fetching the next character from memory while ensuring that the line is kept busy. If a character is not available for output when needed, output ceases until a character becomes available. Since each character has a START and a STOP bit this does not interfere with character synchronization.

For output, the only line parameter set by the program is line speed. Character size is directly determined by the program for each character put out. There are 10 bits available for holding an output character (in BOUTDAT and also in OUTDAT). The START bit is automatically created by the hardware for each character sent to the line and thus the program can ignore this format requirement.

Output is handled by the program through tables accessed by two DMC channels. One of these, the OUTPUT table, contains the outgoing characters. Since the hardware takes the characters from the table in sequence, care must be exercised to prevent trying to feed a character to a port which cannot presently take it, thereby possibly blocking characters to more needly lines. (A character can jam the OUTPUT channel, sitting in OUTBUF until the appropriate line's BOUTDAT can accept it.) The program must thus cleverly feed characters to lines at the proper rates depending upon individual line speeds. To assist in this process

Figure 2-8 Output Data Flow

a second table, the OUTIN table, is used. Each time a line's BOUTDAT buffer empties, a request is entered into the OUTIN table which effectively says, "I can take another character now." (The entry in the table consists simply of the line number.) The program periodically checks the OUTIN table (handling it just like the INPUT table discussed above) and feeds characters into the OUTPUT table for needy lines. Since several lines may need to enter requests at about the same time and the DMC channel may not be able to suck up the requests as fast as the controller wants to generate them, a bit is provided for each port in the MLC to remember that a request needs to be entered. The bit is cleared when the port succeeds in entering its request.

A line will be kept busy if a new character can go from the OUTPUT table into the BOUTDAT between the time BOUTDAT empties and the time the character has been shifted out to the line (complete with stop bit(s)). For slow speed lines there is adequate time for the request to be entered, the program to note the request and put a character into the OUTPUT table, and the character to get into BOUTDAT.

The program looks at the OUTIN table every 3.3 milliseconds. Within that period of time only one character can have been transmitted on the lines whose character period (including START and STOP bits) is greater than 3.3 ms. For an 8-bit character with one START and one STOP bit, this means a bit rate of about 3000 bits/second. Thus, lines of 3000 bits/second and under using characters with 8 data bits and 1 STOP bit can only enter one request into the OUTIN table between program looks, and a new character will be put into the OUTPUT table, in response to this request, before the line "runs dry". Blockage of the output table could, it would appear, prevent the character from going out but, if we ignore higher speed lines for the moment, any line with a request in the OUTIN table has an empty BOUTDAT and thus can accept the character "immediately." Actually, all lines could come due for characters at once and since DMC access requires something over 3.2 microseconds and access to the BOUTDATS is sequential, it can take some time to get a character out to a given line. At worst, we can get out a character to some line about every 50 microseconds and thus to get a character for any line requires at worst about 3 ms (64 x 50 microseconds). For 8 bit/1 STOP bit characters, this means that the bit rate is about 3000 bits/second. Thus, lines operating at 3000 bits/second or less can always get a character from the output table in time, even if all other lines are queued up ahead of it waiting for characters. As we have seen, the program will process requests at about that rate and thus lines of less than 3000 bits/second should operate with no degradation.

Lines of greater than 3000 bits/second may not always be able to get out a character in time and thus for these lines, the characters must be planted carefully into the output table by the program so that they come out frequently enough to keep the line going. By cleverly watching the OUTIN requests, the program can sprinkle the output character for high speed lines into the OUTPUT table so that they do not hold up these lines.

Note that all of these calculations are dependent on character length (including START and STOP bits) shrinks, the problem becomes more acute as the character rate increases for a constant line speed in bits/second.

2.3.3.2 The TIP Software

Because the terminals connected to a TIP communicate with Hosts at remote sites, the TIP, in addition to performing the IMP function, also acts as intermediary between the terminal and the distant Host. This means that network standards for format and protocol must be implemented in the TIP. One can thus think of the TIP software as containing both a very simple-minded mini-Host and a regular IMP program.

Figure 2-9 gives a simplified diagrammatic view of the program. The lower block marked "IMP" represents the usual IMP program. The two lines into and out of that block are logically equivalent to input and output from a host. The code conversion blocks are in fact surprisingly complex and include all of the material for dealing with diverse types of terminals.

Once connection to a remote Host is established, regular messages flow directly through the Input block and on through the IMP program. Returning responses come in through the IMP, into the OUTPUT program where they are fed through the OUTPUT Code Conversion block to the terminal itself.

As the user types of the keyboard, characters go, via input code conversion, to the input block. Information for remote sites is formed into regular network messages and passes through the OR switch to the IMP program for transmittal. Command characters are fed off to the side to the command block where commands are decoded. The commands are then "performed" in that they either set some appropriate parameter or a flag which calls for later action. An example of this is the LOGIN command. Such a command in fact triggers a complex network protocol procedure, the various steps of which are performed by the PROTOCOL block working in conjunction with the remote Host through the IMPs. As part of this process an appropriate special message will be sent to the terminal via the Special Messages block indicating the status (success, failure, etc.) of the procedure.

Figure 2-9 Block Diagram of TIP Program

2.3.3.3. THE TIP Command Format

The user at a terminal will at various times be talking directly to his TIP instead of to the remote Host. A typical message of this sort might look like:

@ OPEN 15

Such a command always starts with symbol @ and ends with either a linefeed or a rubout, depending on whether the user is satisfied with the command or wishes to abort it. The only exception to this rule is the specific command

@@

which inserts an @ in the data stream to the Host. Commands may occur anywhere, and need not start on a new line. Upper and lower case may be freely intermixed in the command.

Between the @ and the linefeed there will typically be one or more words to identify the command, perhaps followed by a single parameter. The TIP is not very sophisticated, and thinks the only important thing about a word is its first letter. This permits the user to abbreviate a bit; the more usual rendering of the first example might be:

@O 15

Once the user has started typing the parameter of a command the old value of the command will have been destroyed, and cannot be recovered by aborting the command.

Almost without exception the effect of a TIP command is to set a parameter or mode for the terminal. Even apparently direct commands like

@ OPEN 15

(which initiates an elaborate exchange of messages resulting in a connection to the remote Host system) actually set a mode flag to request the appropriate action when the TIP is free to undertake it. To understand the TIP behavior is really to understand the complete set of parameters and the commands to change them. Normally, any parameter can be changed at any time by the user at his terminal. Exceptions occur when the user tries to change connection parameters on an open connection. An @OPEN 13 executed while talking to Host 15 would generate the error message "Can't" (the connection to Host 15 must be closed before a connection can be opened to Host 13).

Commands often consist of several command words; for example,

@ DEVICE CODE ASCII

Such commands may be abbreviated; for example

@ D C A

The spaces are required; @ DCA is not a legal command. Upper and lower case letters may be freely intermixed.

An unusual variation in command format is to place a number between the @ and the first word of the command. In this case, the command is not meant for the terminal typing but for the terminal attached to the port of that number on the same TIP as the user.

In the normal course of things, a user will go through five more or less distinct stages in typing into the net. First, he will be concerned with hardware--power, dialing in, etc. Then he will establish a dialogue with the TIP to get a comfortable set of parameters for this usage. Next, he will instruct the TIP to open a connection to a remote Host; and finally, he will mostly ignore the TIP as he talks to the remote Host. The following sections will describe these stages in more detail.

A. Hardware Stage

The hardware stage is out of the scope of this section, except for the final stage of connection. When the TIP detects a terminal on one of its previously idle lines it normally goes into a "hunt" mode. In this mode it expects the very first character it sees to describe the terminal, according to the following scheme

ASCII Terminals at 110, 150, or 300 baud type E. (Note that this must be upper case.)

2741 Correspondence Terminals type], 4, o, or 1 depending on the element used with the terminal -- see Table 4-A.

2741 PTTC Terminals type:

6 for models 938, 939, 961, 962, or 997

o for model 942 or 943

w for model 947 or 948

4 for model 963, 996, or 998

ASCII Terminals transmitting at 110 but receiving at 1200 baud type D. (Again, upper case)

The TIP will deduce terminal rate, character size, and code conversion based on the character typed. When the TIP makes its decision it types out TIP's name in the terminal's own language followed by the version number of the TIP software system and the octal port number. Then it is ready to go.

B. Establishing Parameters

In stage two, the user is concerned with initializing parameters. The naive user should skip stage two and accept the TIP's default parameters until an obvious problem arises. The following questions are answered in stage two:

1. When shall the TIP send off messages to the remote Host?# Here there are several options. (The TIP is initialized to send on every character, which is simple but inefficient.)

   - TRANSMIT NOW
   - TRANSMIT ON MESSAGE-END
   - TRANSMIT ON LINEFEED
   - TRANSMIT EVERY N

   TRANSMIT NOW causes the message currently being accumulated to be sent as soon as possible. TRANSMIT ON MESSAGE-END causes a message to be sent as soon as possible after an ASCII DC3 (control-S) is encountered. TRANSMIT ON LINEFEED causes a message to be sent as soon as possible after a linefeed is encountered. Additionally, both TRANSMIT ON MESSAGE-END and TRANSMIT ON LINEFEED cause characters to be accumulated in the message buffer until it is almost full. TRANSMIT EVERY#_# causes a message to be sent as near as possible to every _#th character. The command TRANSMIT EVERY#0 will reset the TIP to its initial state, transmitting every character. If the parameter to TRANSMIT EVERY is a large number (e.g., 250) the TIP will save up as many characters as it can before sending a message, but does not offer any guarantee that the total number specified can be buffered.

2. Who shall echo, and when?# Echoing is a complex problem, without any neat solution. We have chosen to give the user the means to tell us how he wants it done, since it is hard to guess correctly in advance. Basically, echoing can occur at the terminal hardware, in the TIP, or in the remote Host. The corresponding TIP commands are:

• ECHO HALFDUPLEX	(Echo at terminal)
• ECHO ALL	(Echo at TIP)
• ECHO NONE	(Echo at remote Host)
• ECHO REMOTE	(Send TELNET "remote echo" and perform internal @#E#N)
• ECHO LOCAL	(Send TELNET "local echo" and perform internal @#E#A)

In the ECHO NONE mode, although characters for the remote Host are not echoed, the TIP will echo commands. Network protocol specifies that echoing shall start out in the #ECHO ALL or #ECHO HALF modes. The TIP will try to guess from the terminal type which of the two is appropriate. The goal of echoing strategy is to avoid the unreadable alternatives of the blank page and the doubling of every character. The naive user is advised to accept the TIP's default parameters until trouble of this sort arises. A 2741 is incapable of changing echo mode; it is always echo halfduplex.

To allow more complex echoing conventions, the TELNET protocol provides a mechanism whereby the remote terminal user may instruct the serving Host whether or not to echo characters. The ECHO REMOTE and ECHO LOCAL commands at the TIP allow TIP users to use this mechanism after the connection is made.

Finally, many Hosts which provide service request the TIP to allow them to do the echoing. The TIP always grants this request (even for 2741 terminals). The user, if he does not desire this mode, must cancel it AFTER the connection to the Host is established.

C. Connection to Remote Sites

In stage three, the user is concerned with establishing a connection to a remote site.

@ OPEN 15

This amounts to "set the Host number parameter" and "add the user to the queue of users waiting for the TIP's connection mechanism". Appendix A lists the Host numbers of all the sites currently in the network.

Connecting to a Host requires establishing a bi-directional link, so that the terminal can send characters to the Host and vice versa. The request to connect to a Host is thus really a request to establish both transmit and receive sections.

When the user reaches the head of the queue waiting for the TIP's "connection" mechanism, the TIP will type "Trying...".

Following the message "Trying", the user will receive some of the following messages:

Open	success
Net Trouble	remote site cannot be reached
Refused	remote site up but refusing
Host Scheduled Down Until Sat. at 1850 GMT	Host will be back up at time and date indicated
Host Unscheduled Down Until Sat. at 1850 GMT	Host will back up at time and date indicated

Host not responding

Remote site not up, unknown when up service will resume

ICP Interfered With

The Host has not performed the ICP correctly and the TIP has refused to open a connection.

The connection mechanism will run continuously until a state described above occurs. This can be annoying when the remote site is obviously not going to respond. The command

@ CLOSE

will abort the current connection attempt. The user is then free to reattempt to open the connection or to attempt to open a connection to some other Host.

The TIP's connection mechanism has caused users some problems. Perhaps a discussion of what the connection mechanism is doing and how it works will alleviate some of the grief.

First of all, users attempting to connect to different Hosts will never interfere with each other, although users simultaneously attempting to connect to the same Host will be serviced serially.

For the user, opening proceeds in three phases. In the first, the user is queued up waiting to "get" the TIP's connection mechanism. In the second, the user has gotten the TIP's connection mechanism and is beginning the connection sequence. In the third, the user has completed the connection sequence and is waiting for the Host to open up the actual data connections. Many of the problems stem from the fact that only one user may be proceeding through phase 2 at a given time to a given Host. Hence the the TIP types out "Trying" when you get off the queue and the connection mechanism begins trying to open your connections. Thus the "Trying" message signifies the transition from phase 1 to phase 2.

D. Use of Remote Sites

In stage four, the user is normally talking to the Host without concern for the TIP. All the TIP commands are still available.

One command that will eventually be of interest here is

@ CLOSE

This command starts the shut-down procedure. The TIP will echo "Closed"* when the process is finished. The TIP does not know how to log you out of the remote Host. You must do this yourself before closing the connection.

The virtual terminal has a key labeled "BREAK". Some real terminals have a break key, and some Host systems expect to see breaks. Those terminals with a break key (but not the 2741 ATTN key) may simply use it. Others must type the command

@ SEND BREAK

The interpretation of the break is entirely up to the receiving Host -- many Hosts ignore it.

The virtual terminal also has a key labeled "SYNC". No real terminals have such a key, and the function is unique to network use. The "SYNC" key is a clue to the remote Host that there is an important message which seems to be buffered in an "inaccessible" place. The TIP and the Host go to some trouble to get the SYNC indication over a different channel which bypasses the normal buffering conventions. The command to send a SYNC is

@ SEND SYNC

Typical usage of these commands might be @ S B followed by @ S S.

As stated earlier, the TIP nominally treats a carriage-return typed by a user as a carriage-return/linefeed. The user may cause the TIP to treat carriage-return as only carriage-return by executing the command

@CLEAR INSERT LINEFEED

To return to the nominal mode of carriage-return/linefeed, the command

@INSERT LINEFEED

should be executed.

If at any given time the user types characters faster than a server Host will take them from the TIP, the TIP discards characters it can not buffer and echos them with an ASCII BEL (on a 2741, the type element is wiggled).

The user may sometimes use a server Host with which it is desirable not to have @ be a TIP reserved character. The user can change the character which introduces TIP commands using the command

@INTERCEPT N

By typing @INTERCEPT followed by a decimal number representing an ASCII character, the user changes the TIP command character for his device to the ASCII character represented by the number. The INTERCEPT ESC command resets the TIP command character to at-sign (@). Thus,

@INTERCEPT 42

*INTERCEPT ESC

changes the TIP command character to asterisk (*) and back to at-sign (@) assuming the device was in the nominal mode (@) before the first command was executed.

If the user attempts to change the intercept character but fails to type a valid decimal number (or a character string beginning with E) the TIP will type the diagnostic "Num" and will set the intercept character to at-sign.

E. Connection Loss and Restoration

Starting with Hosts running TENEX Version 1.32, if TENEX halts, the TIP will notify users connected to it of this fact by typing "Connection suspended". At this point the users are free to do one of two things. First, they can wait till TENEX restores service, in which case the TIP will type out "Connection Restored" (or if after the the service interruption the connection could not be restored, the TIP will type out "Host has reset connection"). Alternatively, the user is free to open a connection to any other Host, in which case the TIP will invisibly close the TENEX connection. It is also important to point out that if a user just leaves his terminal unattended across a TENEX service outage without releasing the connection (any network related command such as @H, @O, @N, @C will do the job) his job, directory, etc., are left at the mercy of anyone who acquires that terminal.

F. TIP News and User Feedback

There is frequently information which the group developing and debugging the TIP system wishes to convey to TIP users. For instance, when a bug is detected, we may wish to warn users not to use a certain feature until the bug is fixed. When a minor improvement is made, we may wish to notify users. Further, there is frequently news about the state of the network or the state of a particular Host which should be conveyed to TIP users. Finally, TIP users may wish to communicate with the TIP development staff or the Network Control Center staff about problems or suggested improvements for the TIP or the network. Consequently, we have constructed a mechanism which we hope will provide for communication in all the above directions. This mechanism is the Network Virtual TIP Executive.* To activate this mechanism, the TIP user may give the TIP command @N. (The same one which handles the user's login to the TIP.) This command causes the TIP to perform the necessary protocol to make a connection to the Network Virtual TIP Executive which resides on several of the network TENEX systems. Once the Network Virtual TIP Executive has been activated, we think its operation is self-explanatory. Presently available features within the Network Virtual TIP Executive are a

Network News feature, a Host Status feature, and a "Gripe" feature. The latter provides users with a mechanism for sending messages to the TIP development or NCC staffs. We recommend that TIP users get the network news at the beginning of every TIP session (during the login procedure).

When a user issues an @N command, the TIP requests support from ALL cooperating servers. Thus, the user should be able to reach a news facility, somewhere, almost all of the time. However, in the event that no cooperating server is available the TIP will time out the @N command in about thirty seconds (and respond with "Timeout OK to proceed"). An @C command will abort an @N immediately.

Of course, TIP users with an immediate need for communication with the NCC or TIP development staffs should telephone (collect) the Network Control Center (617-661-0100). Users with general questions about network usage (How do I find out if Host X is ever going to be up again? What's happening with a Host/Host protocol for graphics?) may also call the NCC.

Table 2-4. TIP MESSAGES TO THE TERMINAL USER

BAD	The TIP doesn't recognize the command
Closed	Connection closed, usually by server Host
Connection Restored	Destination Host has restored the connection as it was before the Host halted.
Connection Suspended	Destination Host has halted operation.
Host broke the connection	The Destination Host's service is restored but all network connection tables have been reset.
Host not responding	Destination Host not up from the network's point of view. It is not known when service will resume.
Host Scheduled Down Until ...	Destination Host is scheduled down until the date and time indicated.
Host Unscheduled Down Until ...	Destination Host is unscheduled down until the date and time indicated.
ICP Interferred With	The Host has not performed the ICP correctly and the TIP has refused to open a connection.
Login first	The user must login to the TIP before using this command.
Net Trouble	Destination IMP cannot be reached due to some kind of trouble in the network.
NO	Parameters may not be set for specified terminal.
Num	The TIP expected a number -- command terminated.
Open	Connection opened by server Host.

R Refers to the Receive side of a connection.

Refused The remote Host rejected the attempt to establish connections.

T Refers to the Transmit side of a connection.

Timeout OK to proceed
No RSEXEC is available to accept a login -- the user may continue this TIP session without accounting or access control.

TIP GOING DOWN The TIP is going down in the number of minutes indicated -- quickly stop what you are doing and stop using the TIP.

TIP NAME The TIP heard the user dial in and establish rate. The number following NAME is the TIP software system version number. It is followed by the octal port number.

Trying The TIP is now servicing the user's OPEN request.

Wait The TIP is attempting to contact an RSEXEC for user login.

Table 2-5. TIP COMMAND SUMMARY

BINARY INPUT END
    Leave 8-bit binary input mode

BINARY INPUT START
    Enter 8-bit binary input mode

BINARY OUTPUT END
    Leave 8-bit binary output mode

BINARY OUTPUT START
    Enter 8-bit binary output mode

CLEAR DEVICE WILD
    Set device to be unwild

CLEAR INSERT LINEFEED
    Stop inserting linefeed after carriage-return

CLOSE
    Close all outstanding connections, or abort current Host login

DEVICE CODE 37
    Establish parity computation for Model 37 Teletype

DEVICE CODE ASCII
    Establish code conversion for an ASCII terminal

DEVICE CODE EXTRA-PADDING
    Establish code conversion for a terminal with slow CR

DEVICE CODE OTHER-PADDING
    Establish code conversion for a line printer

DEVICE RATE _#
    _# is a 10-bit code specifying hardware rate and character size settings

_# DIVERT OUTPUT
    Capture device _# and divert this terminal's output to it. _# is an octal number.

ECHO ALL
    Local TIP-generated echo -- TIP echoes everything

ECHO HALFDUPLEX
Terminal-generated echo -- TIP echoes nothing

ECHO LOCAL
Send the Telnet "ECHO LOCAL" character and
perform internal E A

ECHO NONE
Remote Host-generated echo for data --
TIP echoes commands

ECHO REMOTE
Send the Telnet "ECHO REMOTE" character and
perform internal E N

FLUSH
Delete all characters in input buffer

GIVE BACK
Release control of captured device #.
is an octal number.

HOST #
Simultaneous "@S T H" and "@R F H"

INITIAL CONNECTION PROTOCOL
Start the initial connection protocol

INSERT LINEFEED
Insert linefeed after carriage-returns

INTERCEPT #
Use # as TIP command character

INTERCEPT ESC
Leave 7-bit binary mode

INTERCEPT NONE
Enter 7-bit binary mode

LOGIN##
An obsolete form of OPEN

M # #
Mag tape command # with argument #

NETWORK-VIRTUAL-TIP-EXECUTIVE
Connects the user to the Network-Virtual-TIP-Executive, also used to initiate login for some terminals.

OPEN _#
Open a bi-directional connection to the Host decimal address is specified

PROTOCOL BOTH
Simultaneous "@P T T" and "@P T R"

PROTOCOL TO RECEIVE
Manually initiate connection protocol

PROTOCOL TO TRANSMIT
Manually initiate connection protocol

RECEIVE FROM HOST _#
Establish Host _# parameter for manual initialization

RECEIVE FROM SOCKET _#
Establish socket _# parameter for manual initialization of connection -- socket _# is given in octal

RECEIVE FROM WILD
Equivalent to "@R F S <any>"

RESET
Logout current TIP connection and hang-up data set

Reset NCP
Resets NCP

SEND BREAK
Send the Telnet "BREAK" character

SEND COMMAND
Send the command escape character

SEND SYNC
Send the Telnet "SYNC" character and an "INTERRUPT SENDER" message

SEND TO HOST _#
Establish Host _# parameter for manual initialization of connection

SEND TO SOCKET #
    Establish socket # parameter for manual initialization of connection -- socket # is given in octal
SEND TO WILD
    Equivalent to "@S T S <any>"

SET DEVICE WILD
    Equivalent to the commands "@R F H <any>", "@S T H <any>", "@S T S <any>", and "@R F S <any>".

TRANSMIT EVERY #
    Send off input buffer at least every #th character where 0<#<256

TRANSMIT NOW
    Send off input buffer now

TRANSMIT ON LINEFEED
    Send input buffer every time a linefeed is encountered

TRANSMIT ON MESSAGE-END
    Send input buffer every time an end-of-message is encountered

2.3.4 The NCC Hardware and Software

2.3.4.1 Introduction

The NCC Host has two general jobs: it is a tool for diagnosing network problems and it is a network data collector.

As a tool for diagnosing network problems, the primary NCC job is to use the periodic IMP messages (plus the internal clock timeouts) to discover changes. If the changes are major, the NCC Host will sound an alarm and flash a light, to insure that the change is noticed quickly. All changes, whether major or minor, are presented as a human-engineered printout, called the LOG. The LOG reports not only complete failures of a line or an IMP, but also information about unusual events which often precede complete failure (e.g., a noisy line may report a low error rate as some hardware component wears out). The IMP programs also detect many conditions which are not normal, but recoverable, and sends special "trap" messages that appear on the LOG.

The NCC Host is aware of the network topology. Unlike each IMP, which knows only how many "hops" are necessary to reach another IMP, the NCC Host reports about "lines" to the network support staff. Each IMP sends the NCC a snapshot of its environment. The NCC integrates the snapshots into a single picture of the network as a whole, and presents the larger view with lights, alarms, and a LOG printout. This knowledge of topology is occasionally of great importance; for instance, if the network ever partitions, isolating several sites, the NCC can compute the "frontier" and flash those lights while turning off the lights for the "invisible" sites and lines.

As a network data collector, the NCC Host saves some of the raw IMP data in core memory. Host and line throughput data is collected from each IMP and summarized on a terminal hourly. The status of each IMP and line is saved every 15 minutes; a status summary printout is produced every 8 hours. The compacted encoding of the LOG printout builds up in memory. This data may be shipped to other Hosts upon request.

The data collection abilities work in conjunction with processes running on Tenex (PDP-10) computers at other network Host sites. The full data collection is a distributed computation. Other network Hosts have reliable bulk storage and general-purpose languages - the NCC Host is a mini-computer without much storage or power. The two kinds of machines work together to get interesting data to the proper place for analysis. Typically, the NCC data is used for "batch" rather than "real-time" computation on other Hosts.

2.3.4.2 The NCC Hardware

The NCC Host looks like a Terminal IMP (TIP). It is a Honeywell 316 with a Multi-Line Controller (MLC) added. It has 28K words of core memory, an IMP interface, a light-display/alarm, and several terminals. It has a paper tape reader and a modem interface for program loading. It does not need special instructions (for instance, multiply and divide), disks, or other desirable options; the hardware is almost identical to the BBN-TESTIP. In case of serious hardware failure, the BBN-TESTIP may be substituted for the NCC Host with a minimum of effort. The I/O devices and their uses are described briefly below:

1) IMP Interface - The NCC's IMP interface is identical to the IMP's Host interface; the interfaces are connected by a cable which juggles the IMP and Host definitions appropriately. All network data, whether IMP or Host traffic, uses this path.

2) Primary terminal - This Model 35 Teletype has two major functions. It prints periodic reports of Host and line throughput and status. Secondly, it is used to change NCC parameters, and as a debugging aid, much as the IMP Teletype and DDT code are used for IMP control and debugging.

3) Lightbox & Alarm - A BBN-designed I/O device with 64 LED lights, 16 switches, and an audible buzzer; this is the primary warning of important network changes. Since there is only one such device, the program can run without it, although the support staff would have to keep a careful eye on the LOG printout if the lightbox/alarm were broken.

4) LOG terminals - The LOG produces a hard copy record of network events. Normally two small lineprinters are used: a Centronics 306 is next to the NCC Host, and a quiet Inktronics is in the manned area of the Network Control Center; both run at 1200 baud.

5) MSG-DUMP terminal - One terminal is dedicated to printing an octal dump of any network message which comes from an IMP (as opposed to a Host) but is not a recognized status or throughput message. This facility is a flexible IMP debugging aid, particularly for viewing "bad" packets.

6) TT-EXER terminals - Other MLC ports may be used to exercise terminals by continuously printing a simple pattern. This is an aid to hardware maintenance of the LOG and MSG-DUMP devices.

2.3.4.3 NCC Outputs

The following few pages contain sample printouts. The first page shows the LOG printout similar to the lineprinter output in the Network Control Center. The remaining pages of printout come from the primary TTY, often referred to as the "summary TTY". The IMP and line status printouts are compressed, with the following 9 single characters used to indicate different states:

?	Unknown
+	Down in the plus direction (lines only)
-	Down in the minus direction (lines only)
*	Down
.	Up
>	Plus end looped (lines only)
<	Minus end looped (lines only)
Z	Both ends looped (lines only)
X	Status changed one or more times during the interval

The Host throughput data is divided into 8 categories as follows:

M->N	Messages to Hosts at other nodes.
M<-N	Messages from Hosts at other nodes.
P->N	Packets to Hosts at other nodes.
P<-N	Packets from Hosts at other nodes.
M->$	Messages to Hosts at this node.
M<-$	Messages from Hosts at this node.
P->$	Packets to Hosts at this node.
P<-$	Packets from Hosts at this node.

Every 8 hours, the IMP status, line status, Host throughput, and line throughput reports are printed, summarizing the period since midnight. Every hour, the Host and line throughput reports summarize the previous hour. The "quick summary" and "host summary" reports print only at the operator's command.

```
1100  NOVEMBER 13 1976   ARPA NETWORK LOG    PAGE 99

1100  IMP 60: *(102,,4,0)
     LINE 41: ERRORS PLUS 2/16
      IMP 58: HOST 1 DOWN
1101 LINE 41: ERRORS MINUS 1/17
      IMP 61: *(102,,0,0)
      IMP 58: HOST 1 UP, HOST 1 DOWN
      IMP 60: *(102,,1,177771)
1102  IMP 58: HOST 1 UP
     LINE 41: ERRORS MINUS 1/17
      IMP 61: *(102,,0,0)
1103  IMP 58: HOST 1 DOWN
      IMP 60: *(102,,177775,0)
      IMP 58: HOST 1 UP
1104  IMP 58: HOST 1 DOWN
      IMP 45: HOST 1 DOWN
      IMP 58: HOST 1 UP
      IMP 45: HOST 1 UP
      IMP 61: *(102,,0,0)

1105 LINE 41: DOWN ***

      IMP 58: HOST 1 DOWN, HOST 1 UP, HOST 1 DOWN
1106  IMP 58: HOST 1 UP
      IMP 61: *(102,,0,0)
      IMP 58: HOST 1 DOWN
1107  IMP 60: *(102,,177777,177776)
      IMP 58: HOST 1 UP
      IMP 58: HOST 1 DOWN
1108  IMP 39: TRAP: (7706,2,1)
      IMP  6: HOST 0 UP
      IMP 11: *(32,,104647,0)
1109  IMP 11: HOST 0 DOWN
1115 LINE 41: ERRORS PLUS 1/16
      IMP 41: *(63,,400,47)

1116 LINE 41: UP ***

1117  IMP 30: POWER FAILURE, RESTARTED
1118  IMP 30: OVERRIDE ON, TRAP: (23456,0,36)

     LINE 88: LOOPED PLUS ***
```

```
NOVEMBER  4 1976   ARPA NETWORK SUMMARY 0000-0800   PAGE 148

IMP STATUS

TIME   1234567890 1234567890 1234567890 1234567890 1234567890 1234567890

0000   .......... .......... .......... .......... .......... ..........
       .?.
0030   .......... .......... .......... .......... X......... ..........
       .?.
0045   .......... .......... .......... .......... *......... ..........
       .?.
0100   .......... .......... .......... .......... X......... ..........
       .?.
0115   .......... .......... .......... .......... .......... ..........
       .?.
0130   .......... .......... .......... .......... X......... ..........
       .?.
0145   .......... .......... .......... .......... X......... ..........
       .?.
0200   .......... .......... .......... .......... *......... ..........
       .?.
0300   .......... .......... .......... .......... X......... ..........
       .?.
0315   .......... .......... .......... .......... X......... ..........
       .?.
0330   .......... .......... .......... .......... *......... ..........
       .?.
0345   .......... .......... .......... .......... X......... ..........
       .?.
0400   .......... .......... .......... .......... .......... ..........
       .?.
0500   .......... .......... .......... .......... X......... ..........
       .?.
0515   .......... .......... .......... .......... .......... ..........
       .?.
0530   .......... .......... .......... .......... X......... ..........
       .?.
0545   .......... .......... .......... .......... .......... ..........
       .?.
```

```
NOVEMBER  4 1976   ARPA NETWORK SUMMARY 0000-0800   PAGE 150

LINE STATUS

TIME  1234567890 1234567890 1234567890 1234567890 1234567890 1234567890

0000  .......... .......... .....?.... .......... .......... .?....*...
      .?........ ...?...<<.
0030  .......... .......... .....?.... .......... X......... .?....*...
      .?........ ...?...<<.
0045  .......... .......... .....?.... .......... *X........ .?....*...
      .?........ ...?...<<.
0100  .......... .......... .....?.... .......... XX........ .?....*...
      .?........ ...?...<<.
0115  .......... .......... .....?.... .......... .......... .?....*...
      .?........ ...?...<<.
0130  .......... .......... .....?.... .......... XX........ .?....*...
      .?........ ...?...<<.
0200  .......... .......... .....?.... .......... **........ .?....*...
      .?........ ...?...<<.
0300  .......... .......... .....?.... .......... XX........ .?....*...
      .?........ ...?...<<.
0315  .......... .......... .....?.... .......... XX........ .?....*...
      .?........ ...?...<<.
0330  .......... .......... .....?.... .......... **........ .?....*...
      .?........ ...?...<<.
0345  .......... .......... .....?.... .......... XX.......X .?....*...
      .?........ ...?...<<.
0400  .......... .......... .....?.... .......... .......... .?....*...
      .?........ ...?...<<.
0500  .......... .......... .....?.... .......... X......... .?....*...
      .?........ ...?...<<.
0515  .......... .......... .....?.... .......... X......... .?....*...
      .?........ ...?...<<.
0530  .......... .......... .....?.... .......... X......... .?....*...
      .?........ ...?...<<.
0545  .......... .......... .....?.... .......... X......... .?....*...
      .?........ ...?...<<.
0600  .......... .......... .....?.... .......... X......... .?....*...
      .?........ ...?...<<.
```

NOVEMBER 4 1976 ARPA NETWORK SUMMARY 0000-0800 PAGE 152

	HOST 0	HOST 1	HOST 2	HOST 3	THROUGHPUT
SITE 1 UCLA					
M->N	23778	7383	2489		
M<-N	26460	10963	3284		
P->N	23778	12117	2489		
P<-N	27937	10976	3284		
M->S	4667	1152	1757		
M<-S	1877	1137	4562		
P->S	4874	1192	2742		
P<-S	2862	1344	4562		
SITE 2 SRI-5					
M->N	61632	108480			
M<-N	67072	124160			
P->N	61632	125696			
P<-N	84864	132416			
M->S	2878	5280			
M<-S	4311	3763			
P->S	2877	6810			
P<-S	5870	3814			
SITE 3 NUC					
M->N			539		
M<-N			721		
P->N			539		
P<-N			956		
SITE 4 UTAH					
M->N			52928		
M<-N			36608		
P->N			52928		
P<-N			38336		
SITE 5 BBN					
M->N	134400		385856	13979	
M<-N	172160	18	335488	13911	
P->N	148672		385856	21813	
P<-N	177856	18	350016	14062	
M->S	129600	513	217152	31506	
M<-S	177344	481	147904	53056	
P->S	143168	513	217472	34624	
P<-S	177792	2036	162816	53120	

NOVEMBER 4 1976 ARPA NETWORK SUMMARY 1600-1700 PAGE 177

LINE THROUGHPUT

LINE 1:	PENT	TO	EGLIN	22612	EGLIN	TO	PENT	23768
LINE 2:	M-MAC	TO	CCA	41216	CCA	TO	M-MAC	38144
LINE 3:	M-MAC	TO	WRPAT	23046	WRPAT	TO	M-MAC	23166
LINE 4:	USC	TO	DOCB	30635	DOCB	TO	USC	30962
LINE 5:	UCLA	TO	USC	13773	USC	TO	UCLA	14089
LINE 6:	UCLA	TO	NUC	16267	NUC	TO	UCLA	16351
LINE 7:	UCLA	TO	SCRL	14307	SCRL	TO	UCLA	13909
LINE 8:	SRI-5	TO	XEROX	46592	XEROX	TO	SRI-5	46144
LINE 9:	BELV	TO	ABER	17052	ABER	TO	BELV	17080
LINE 10:	SRI-5	TO	LLL	9630	LLL	TO	SRI-5	10199
LINE 11:	AMEST	TO	SRI-3	22871	SRI-3	TO	AMEST	21604
LINE 12:	STAN	TO	ISI22	85312	ISI22	TO	STAN	86208
LINE 13:	HARV	TO	NCC-T	55168	NCC-T	TO	HARV	49344
LINE 14:	M-MAC	TO	M-TIP	40384	M-TIP	TO	M-MAC	33088
LINE 15:	UTAH	TO	ILL	15332	ILL	TO	UTAH	14837
LINE 16:	LINC	TO	ROME	24887	ROME	TO	LINC	16972
LINE 17:	CARN	TO	ARGON	52352	ARGON	TO	CARN	52352
LINE 18:	CARN	TO	BELV	13848	BELV	TO	CARN	12871
LINE 19:	HARV	TO	NYU	28206	NYU	TO	HARV	29017
LINE 20:	STAN	TO	AMES	60352	AMES	TO	STAN	60992
LINE 21:	MITRE	TO	ARPA	19257	ARPA	TO	MITRE	19130
LINE 22:	CARN	TO	ROME	39680	ROME	TO	CARN	40976
LINE 23:	NBS	TO	NSA	12585	NSA	TO	NBS	13212
LINE 24:	AMES	TO	AMEST	39872	AMEST	TO	AMES	39040
LINE 25:	ISI22	TO	KIRT	79680	KIRT	TO	ISI22	81280
LINE 27:	GWC	TO	ARGON	29503	ARGON	TO	GWC	29941
LINE 28:	DCEC	TO	SDACP	38912	SDACP	TO	DCEC	43584
LINE 29:	MITRE	TO	SDACP	38464	SDACP	TO	MITRE	33280
LINE 30:	BBN T	TO	NCC-T	1837	NCC-T	TO	BBN T	165
LINE 31:	BBN	TO	CCA	43904	CCA	TO	BBN	43072
LINE 32:	GWC	TO	DOCB	32896	DOCB	TO	GWC	32727
LINE 33:	PENT	TO	ARPA	7842	ARPA	TO	PENT	7731
LINE 34:	NBS	TO	ABER	31350	ABER	TO	NBS	29117
LINE 35:	XEROX	TO	TYMSH	50240	TYMSH	TO	XEROX	50752
LINE 36:	RAND	TO	ISI52	38464	ISI52	TO	RAND	38656
LINE 37:	FNWC	TO	SCRL	13787	SCRL	TO	FNWC	14464
LINE 38:	RAND	TO	NELC	32768	NELC	TO	RAND	31974
LINE 39:	AMES	TO	HAW T	17047	HAW T	TO	AMES	16946
LINE 40:	UTAH	TO	LBL	27488	LBL	TO	UTAH	27827
LINE 41:	SDACP	TO	NORSR	15806	NORSR	TO	SDACP	17167
LINE 42:	NORSR	TO	ULOND	652	ULOND	TO	NORSR	907

```
1502 NOVEMBER 10 1976  QUICK SUMMARY

 IMP
 .......*.. .......... .......... .......?.. .......... ..........
 .?.
 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
 .......... .......... .....?...? .......... .......... .?........
 .?........ ...?...<<.
 LINE

IMP  1 MEM PROTECT OFF

IMP  2 MEM PROTECT OFF

IMP  3 MEM PROTECT OFF

IMP  4 TIP UP

IMP  5 MEM PROTECT OFF
       OVERRIDE ENABLED

IMP  6 MEM PROTECT OFF

IMP  7 TIP UP

IMP 10 MEM PROTECT OFF

IMP 11 MEM PROTECT OFF

IMP 12 MEM PROTECT OFF

IMP 13 TIP UP

IMP 14 MEM PROTECT OFF

IMP 15 MEM PROTECT OFF

IMP 16 TIP UP

IMP 17 TIP UP

IMP 18 TIP UP
```

```
1507 NOVEMBER 10 1976  HOST SUMMARY

 IMP
 !!!!!!!*!! !!!!!!!!!! !!!!!!!!!! !!!!!!!?!! !!!!!!!!!! !!!!!!!!!! !?!
 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 123

IMP  1 HOST 0 UP
       HOST 1 UP
       HOST 2 DOWN

IMP  2 HOST 0 UP
       HOST 1 UP
       HOST 2 DOWN

IMP  3 HOST 0 DOWN
       HOST 1 DOWN
       HOST 2 DOWN
       HOST 3 UP

IMP  4 HOST 0 DOWN
       TIP UP

IMP  5 HOST 0 UP
       HOST 1 TARDY
       HOST 2 UP
       HOST 3 UP

IMP  6 HOST 0 UP
       HOST 1 UP
       HOST 2 UP
       HOST 3 UP

IMP  7 HOST 0 UP
       TIP UP
       HOST 3 UP

IMP  9 HOST 0 UP
       HOST 1 TARDY
       HOST 3 DOWN

IMP 10 HOST 0 UP
       HOST 1 DOWN
       HOST 2 DOWN
       HOST 3 DOWN
```

2.3.4.4 The NCC Software

This section describes the NCC program organization; when reasonable, reference to specific subroutine names in the code of the NCC system has been made.

The program is structured in three logical levels - background, foreground, and I/O. Background, the idle machine state, is used as a mechanism for starting up some I/O events. Foreground processing, the bulk of the program, includes handling IMP messages. I/O routines are short, high-priority, uninterruptable strips of the program.

The Honeywell 316 has an interrupt structure which permits multiple interrupt levels. The hardware instruction set allows both global and selective inhibit/enable control. The background state means running with most or all interrupts enabled, but not currently on any interrupt level. The foreground state means operating on either of the two lowest priority interrupt levels, with these two levels disabled, but with other interrupts allowed. I/O interrupts run to completion with the global inhibit in effect. Different hardware levels must work together for many of the NCC functions. Some of the software interfaces between hardware levels are very simple, yet some are complex. Most I/O interrupts signal the main TASK processing in some way. The interactions are:

IMP interface input and TASK,
IMP interface output, Background, and TASK,
MLC outin (LOGGER), MLC output, Background, TASK, and CLOCK,
Primary TTY, Background, and CLOCK,
TASK and CLOCK,

```
                          --------------
Background:               |  BACK      |
                          --------------

                  -----------    -----------
Foreground:       |  TASK   |    |  CLOCK  |
                  -----------    -----------

                          ---------------
                 /        | IMP Input   |
                          | IMP Output  |
              /           | Primary TTY |
     I/O:                 | MLC Outin   |
              \           | MLC Output  |
                          |-------------|
                 \        | Power fail  |
                          ---------------
```

Figure 2-10: The Three Logical Levels

Foreground processing occurs on two low-priority interrupt levels called TASK and CLOCK. CLOCK, called by a 25.6 millisecond hardware interrupt, keeps track of the time of day, notices if any IMP or line has not been heard from for several minutes, and either performs or requests background to perform time-dependent functions. CLOCK is covered in more detail later. CLOCK and TASK do not interrupt each other, and can share subroutines.

The TASK interrupt is a "software" interrupt caused by execution of an I/O instruction. The IMP interface input causes the TASK interrupt whenever a full message arrives from the network. Figure 2-11 is the TASK overview, showing the four major sections of code within the wavy-edged boxes. Newly arrived network messages are routed to one of four places - IMP status message handling, IMP throughput message handling, IMP octal printout (MSG-DUMP) handling, and Host core dump request handling. The status message handling was covered in the "IMP MESSAGE PROCESSING" chapter. The other three message handling routines are short and straightforward; the throughput values are added into the proper Host and line tables; the MSG-DUMP message is copied to another area for later printout; and the core dump message is checked for legitimacy, then the parameters are saved in the NONPWC/NONPLO variables.

Figure 2-11: Major Portions of TASK Level

CLOCK is the second interrupt which is used in the foreground level of the program. It keeps track of time, notices IMPs and lines which have not reported to the NCC recently, sets flags for background level to initiate periodic printouts, and initiates periodic disk file updating. If the optional alarm and light display hardware is installed, it is updated frequently.

Since the clock can be locked out for in excess of its 25.6 millisecond period, the main entry, TOR, computes the number of intervals which have occurred since the last clock interrupt and calls TOWORK once for each elapsed interval. As explained in the section about program structure, I/O interrupts are enabled but TASK and extra CLOCK interrupts are inhibited.

Every Second

If the primary terminal timer is running (and inhibiting the output-only function), it is counted down. If zero is reached, the primary terminal state is set to "idle" (TTSSY=0).

If senseswitch 3 has been turned on, CLOCK checks for an inactive machine. The IMP-Host ready line is dropped. The LOG terminal must be idle and the primary terminal must be idle. If everything is inactive, the program halts.

Every Hour

The primary terminal hourly throughput printout flag is set for action by background level. If the hour is a multiple of eight, the total throughput and total status printout flags are set. If the hour is 24, the LOG terminal date and page number are set up for a new day.

If the "summary" time has reached the hour 1 or 25, the tables DHTP and DLTP are cleared.

The tables HTP1 and LTP1 are copied into HTP2 and LTP2. The HTP1 and LTP1 values are added into DHTP and DLTP. The HTP1 and LTP1 tables are zeroed.

Every 15 Minutes

The IMP and line status values are pushed on a stack that contains 96 such entries - one for each 15-minute segment in a day. In this way, status states are quantized to 15-minute periods, and any information on status since the previous midnight can be recovered.

Every Minute

The program checks IMP and line status each minute. If an IMP has not reported recently, the following actions are taken:

1) The IMP is declared down in the IMP status table
2) A logger message is printed to indicate the IMP is down
3) The alarm display light for the IMP is flashed
4) All previous status indicators for that IMP are cleared to nominal values

PLUS END

MINUS END	No Info	Up	Down, no errors	Down with errors	Looped
No Info	Unknown ?	Up .	Unknown ?	Unknown ?	Looped plus >
Up	Up .	Up .	limbo	limbo	limbo
Down, no errors	Unknown ?	limbo	Down *	Down plus +	Looped plus >
Down with errors	Unknown ?	limbo	Down minus -	Down *	Looped plus >
Looped	Looped minus <	limbo	Looped minus <	Looped minus <	Looped both Z

Figure 2-12 Line Status Determination

(Plus end = high-numbered IMP)

Line status is determined every minute by comparing the latest report from each of two IMPs on a line. The transition table shown in Figure 2-12 is used to choose the current state.

If a change in status is detected, the following actions are taken:

1) A logger message is printed
2) The new status is recorded in the line status table
3) The alarm display lights are set
   a) UP = turn on both plus and minus lights, turn off flashing
   b) DOWN MINUS = turn on plus, turn off minus lights, turn on flashing
   c) DOWN PLUS = turn off plus, turn on minus lights, turn on flashing
   d) ALL OTHER STATES = turn off lights, turn on flashing

Each minute, IMP and line statuses are updated. IMP visibility is a central concept to this procedure. There are two possible reasons why an IMP might not be reporting in: first, because the IMP is actually malfunctioning, and second, because it has become isolated away from the NCC (i.e., "invisible"). The NCC distinguishes these cases so that in the event of a failure which partitions the network, the NCC can determine the IMP involved, thus minimizing the number of irrelevant Logger messages and making such occurrences less of a deduction problem for the operator.

2.3.5 The Pluribus IMP Hardware and Software

In its original form, the IMP was based as much as possible on the most suitable off-the-shelf hardware then available; special hardware design was kept to a minimum and consisted of interfaces and a few special features which were added to a standard machine. During the first few years of the network's existence, new and more flexible computer structures began to appear on the market, and the special requirements of packet-switching began to be better understood. The Pluribus architecture, developed specifically to suit the needs of a packet-switch, is the outgrowth of these changes. There are two primary goals for the machine, representing areas in which the earlier IMPs were felt to be lacking. These were flexibility (i.e., the ability to expand or contract smoothly over wide ranges) and reliability. Originally there was a more primitive goal of higher throughput, but this was soon seen to be balanced by the need for a cheaper, smaller machine with less throughput. Bandwidth is thus seen as one domain in which greater flexibility is desired.

Let us consider the issue of flexibility in a little more depth. In most machines certain hardware "utilities" are shared among the various logical units. These include rack space, power, cooling, etc. Generally these utilities come in fairly large chunks, with correspondingly large steps in cost. Thus, one can typically add, say, interfaces up to a certain point; at which time a new rack, power supply, etc., must be added to permit further expansion. Even then, one may run out of logical channels or come up against other hard boundaries. The 516/316, for example, has a fixed memory channel arrangement which limits connection to a total of at most seven high-speed circuits and/or host computers. The specific component which is totally inflexible in most systems is the processor; that is, there is typically no processor modularity or possible variation of processor capacity. The flexibility goal of the Pluribus was to smooth large step functions in cost by utilizing a highly modular design and to push really hard boundaries (such as absolute limits on memory addressing capabilities or processing capacity) well beyond requirements anticipated at least for the next few years.

The machine design was thus to allow for large numbers of I/O units, for wide ranges in processor power (up to at least an order of magnitude in traffic bandwidth handling capability improvement over the 316), and for larger possible memory (to permit longer and/or faster links, say, via satellite).

Now consider reliability. If a single IMP fails on an average of once a month (quite a high MTBF for conventional computers), then in a network of thirty such IMPs, one will fail on an average of once a day. In a network consisting of large numbers of computers (e.g., hundreds), issues of reliability take on paramount importance. In a multiply connected network the failure of a single IMP should be felt only locally. Economics, however, limits the number of paths between any given pair of IMPs. In the ARPANET many IMP pairs are connected by only two disjoint paths, some by only one, and this tends to increase reliance on individual IMPs and to make their reliability a key issue.

If a single computer fails on an average of ten times a year, then a collection of ten computers, treated as a unit, will fail on an average of 100 times a year. However, suppose that rather than viewing the ten computers as a unit which is <u>down</u> if <u>any</u> one of its constituent computers is down, we view the ten computers as a unit which is <u>up</u> as long as <u>any</u> of its constituent computers is up. Further, suppose the mean time to repair a failed computer is small compared to the time between failures. In that case the probability that the constituent computers will all be down is very small, so it is unlikely that the unit as a whole will be down. The reliability of the Pluribus IMP takes advantage of such probabilities. Note that a key assumption is that individual failures are independent of one another. Although it is impossible to guarantee this independence (flood, total power failure, sabotage, etc.), nonetheless, in the Pluribus design, considerable attention has been given to maintaining as much isolation as practical so that one failure does not induce another.

With the goals of flexibility and reliability in mind and with the price and size of minicomputers dropping, it was decided that the Pluribus should be built along the lines of a minicomputer-multiprocessor, or more generally, a multi-resource (processors, memories, I/O channels, etc.) system.

In considering which minicomputer might be most easily adaptable to a multi-resource structure, the internal communication between the processor and its memory was of primary concern. Several years ago machines were introduced which combined memory and I/O busses into a single bus. As part of this step, registers within the devices (pointers, status and control registers, and the like) were made to look like memory cells so that they and the memory could be referenced in a homogeneous manner. This structure forms a very clean and attractive architecture in which any unit can bid to become master of the bus in order to communicate with any other desired unit. One of the important features of this structure is that it made memory accessing "public"; the interface to the memory had to become asynchronous, cleanly isolable electrically and mechanically, and well documented and stable. A characteristic of this architecture is that all references between users are time-multiplexed onto a single bus. Conflicts for bus usage therefore establish an ultimate upper bound on overall performance, and attempts to speed up the bus eventually run into serious problems in arbitration.

In 1972 a new computer was introduced--the Lockheed SUE--which follows the single bus philosophy but carries it an important step further by removing the bus arbitration logic to a module separate from the processor. This step permits one to consider configurations embodying multiple processors, as well as multiple memories and I/O, on a single bus. It also permits busses which do not include any processor at all. The processor used in the SUE computer is a compact, relatively inexpensive (approximately $600 in quantity), quite slow processor with a microcoded inner structure. Table 2-5 shows some of its characteristics. Its slowness and cheapness, of course, go together and since in a modular multiprocessor, increased bandwidth is achieved merely by adding more processors, the weak/cheap processor has the advantage of allowing smaller steps to be taken along the cost/performance curve.

16-bit word

8 General Registers

3.7 microseconds add or load time

Microcoded

Two words/instruction typical

8-1/2" x 19" x 18" chassis

64K bytes addressable by a
single instruction

200 ns minimum bus cycle time

850 ns memory cycle time

425 ns memory access time

Table 2-5 Sue Computer Characteristics

Several components of the SUE computer were adopted for the Pluribus system, in particular the bus, the processor, and the bus arbitration logic.

2.3.5.1 Hardware Structure of the Pluribus IMP

Reliability was a main concern in planning the hardware architecture. The goal was to provide hardware which could be exploited by the program to survive the failure of any individual component.

The hardware consists of asynchronously and independently functioning communication busses, coupled together. From a physical point of view, the SUE chassis represents the basic construction unit; it incorporates a printed circuit back plane which forms the bus into which 24 cards may be plugged. From a logical point of view this chassis includes a bus which provides a common connection among all units plugged into the chassis. All specially designed cards as well as all Lockheed-provided modules plug into these bus chassis. With this hardware, the terms "bus" and "chassis" are used somewhat interchangeably, but we will commonly call this standard building unit a "bus." Each bus requires one card which performs arbitration. A bus can be logically extended (via a bus extender unit) to a second bus if additional card space is required; in such a case, a single bus arbiter controls access to the entire extended bus.

One can build a small multiprocessor just by plugging several processors and memories (and I/O) into a single bus. For larger systems one quickly exceeds the bandwidth capability of a single bus and is forced to multi-bus architecture. Please refer to figure 2-13 for the following discussions.

The functional units of the system (processors, memories, I/O controllers, and special devices) are distributed on these busses in such a way that units which must communicate frequently are placed on the same bus, whereas units which communicate less frequently will in general be on different busses. Units on the same bus can thus communicate at high speed without disturbing the remainder of the system. When a unit on one bus must communicate with a unit on another bus, some interference occurs while both busses are momentarily involved in the interaction. Each bus, together with its own power supply and cooling, is mounted in its own modular unit, permitting flexible variation in the size and structure of systems. There are processor busses each of which contains two processors, each in turn with its own local 4K memory which stores frequently run and recovery-related code. There are memory busses to house the segments of a large memory common to all the processors. Finally, there are I/O busses which house device controllers as well as certain central resources such as system clocks and special (priority-ordered) task disbursers which replace the traditional priority interrupt system.

The design is highly modular and permits systems of widely varying size and performance. In the interests of clarity,

Figure 2-13: Pluribus Architecture

however, we will describe that design in terms of a particular system containing 14 processors--a system designed to have ten times the traffic-handling capability of the 316 IMP.

__Resources.__ A central notion in a parallel system is the idea of a "resource", which we define to mean a part of the system needed by more than one of the parallel users and therefore a possible source of contention. The three basic hardware resources are the memories, the I/O, and the processors. It is useful to consider the memories, furthermore, as a collection of resources of quite different character: a program, queues and variables of a global nature, local variables, and large areas of buffer storage. The multiprocessor system is therefore in reality a multi-resource system, as mentioned above. The basic idea of the system is to provide multiple copies of the vital resources in the hope that the algorithm can run faster by using them in parallel and can survive failures of copies of any of the resources. The number of copies of the resource which are required to allow concurrent operation is determined by the speed of the resource and the frequency with which it is used. An additional advantage of multiple copies is reliability: if a system contains a few spare copies of all resources, it can continue to operate when one copy breaks.

It may seem peculiar to think of a processor as a resource, but in fact in a Pluribus IMP system the parallel parts of the algorithm compete with each other for a processor on which to run. Indeed, a novel feature of the Pluribus IMP design is the consistent treatment of all processors as equal units, both in the hardware and in the software. There is no specialization of processors for particular system functions, and no assignment of priority among the processors, such as designating one as master. Not only the IMP application job but also the multiprocessor control and reliability jobs are distributed among the processors so that all jobs are uniformly treated. The processors are viewed as a resource used to advance the algorithm; the identity of the processor performing a particular task is of no importance. Programs are written as for a single processor except that the algorithm includes interlocks necessary to insure multiprocessor sequentiality when required. The software thus consists of a single conventional program run by all processors. Each processor has its own local copy of about one quarter of this program and the remaining three quarters is in commonly accessible memory.

Processor busses. The bus used in the Pluribus can support up to four processors. As more are added, contention for the bus increases, and the performance increment per processor drops. The Pluribus IMP uses two processors per bus, which loses almost nothing in processor performance. When a processor makes access to shared memory via the switching arrangement, that access incurs delays due to contention and delays introduced by the intervening switch. In the IMP application, some parts of the program are run very frequently and other parts are run far less frequently. This allows a significant advantage to be gained by the use of private memory. A 4K local memory containing an individual copy of the frequently run code is associated with each processor on its bus. This allows faster access to this "hot" code; the local memories all contain the same code. In the IMP application, the ratio of accesses to local versus shared memory is better than three to one.

Shared memory busses. The shared memory contains program, message buffers, global variables, etc. Buffer requirements, of course, vary depending on the site. In a regular IMP, 40K words of common memory are necessary. Four memory units operating concurrently are required in order to hold processor contention to an acceptable level in the 14-processor system. Since the bus is considerably faster than the memories, two logical memory units may be placed on each shared memory bus with almost no interference. Two such memory busses are thus required.

I/O busses. The I/O system consists of more standard busses. In the case of the 14-processor system, two such busses allow for the necessary bandwidth and provide redundancy for reliability purposes. Into these busses are plugged cards for each of the various types of I/O interfaces that are required, including interfaces for modems, terminals, host computers, etc., as well as interfaces for standard peripherals. The I/O bus also houses a number of special units including (1) a clock which serves roughly the same functions as the clock in the 516/316 IMP; (2) a checksum/block transfer card which flows a block of memory through itself computing a checksum as it goes (used to checksum critical code, to compute end-to-end-checksums, etc.); (3) a special hardware task disbursing unit known as a Pseudo-Interrupt Device (PID) discussed further below; and (4) a "reload" card which monitors up to eight communication lines, watching for specially formatted reload messages from the network and processing them should any arrive.

__Intercommunication system.__ To adhere to the requirement that all processors must be equal and able to perform any system task, buses must be connected so that all processors can access all shared memory, so that I/O can be fed to and from shared memory, and so that any of the processors may control the operation and sense the status of any I/O unit.

A distributed inter-communication scheme was chosen in the interest of expandability, reliability, and design simplicity. The kernel of this scheme is called a Bus Coupler, and consists of two cards and an interconnecting cable. In making connections between processors and shared memory, one card plugs into a shared memory bus, the other into a processor bus. Similar connections are made for every processor bus to every shared memory bus. When the processor requests a cycle within the address range which the Bus Coupler recognizes, a request is sent down the cable to the memory end, which then starts contending for the shared memory bus. When selected, it requests the desired cycle of the shared memory. The memory returns the desired information to the Bus Coupler, which then provides it to the requesting processor, which, except for an additional delay, does not know that the memory was not on its own bus. The Bus Coupler also does address mapping. Since a processor can address only 64K bytes (16-bit address), and since we wished to permit multiprocessor configurations with up to 1024K bytes (20-bit address) of shared memory, a mechanism for address expansion is required. The Bus Coupler provides four independent 8K-byte windows into shared memory. The processor can load registers in the Bus Coupler which provide the high-order bits of the shared memory address for each of the four windows.

Given a Bus Coupler connecting each processor bus to each shared-memory bus, all processors can access all shared memory. I/O devices which do direct memory transfers must also access these shared memories. These I/O devices are plugged into as many

I/O busses as are required to handle the bandwidth involved, and bus couplers then connect each I/O bus to each memory bus. Similarly, I/O devices also need to respond to processor requests for action or information; in this regard, the I/O devices act like memories and Bus Couplers are again used to connect each processor bus to each I/O bus. The path between processor busses and I/O busses is also used to allow processors to examine and control other processors for startup and trouble situations.

2.3.5.2 Software Structure of the Pluribus IMP

The problem of building a packet-switching store-and-forward communications processor lends itself especially well to parallel solution since packets of data can be treated independently of one another. Other functions of the IMP program such as general housekeeping, routine computations, reliability tasks, etc., can also be easily performed in parallel. The structure chosen works as follows: first, the program is divided into small pieces, called strips, each of which handles a particular aspect of the job. For example, one strip handles special routing messages from neighboring IMPs, another handles input from a local host, and others handle further I/O and housekeeping functions. When a particular task needs to be performed, for instance upon receipt of a message over a communications circuit, the name (number) of the appropriate strip is put on a queue of tasks to be run. Each processor, when it is not running a strip, repeatedly checks this queue. When a strip number appears on the queue, the next available processor will take it off the queue and execute the corresponding strip. The program is broken into strips in such a way that a minimum of context saving is necessary.

Strips have different levels of importance. Data coming in over a high-speed communication circuit must be serviced more rapidly than data coming in over a Teletype-speed line. The number assigned to each strip reflects the priority of the task it performs. When a processor checks the task assignment queue, it takes the highest priority job then available. Since all processors access this queue frequently, the contention for it is very high. For efficiency, therefore, a special hardware device, the Pseudo Interrupt Device, was designed to serve as a task queue. A single instruction allows the highest priority task to be fetched and removed from the queue. Another instruction allows a new task to be put onto the queue. All contention is arbitrated by standard bus logic hardware.

The length of strips is governed by how long priority tasks can wait if all the processors are busy. The worst case arises when all processors have just begun the longest strip. In the IMP application, the most urgent tasks can afford to wait a maximum of 400 microseconds. Therefore, strips must not be longer than that.

(Of course, a strip might be longer if it is run infrequently and if the urgent tasks do not have absolute time requirements. That is, one might build a statistically acceptable set of strip lengths.)

An inherent part of multiprocessor operation is the locking of critical resources**. This is the mechanism by which the algorithm enforces sequentiality when it is needed. Our system uses an uninterruptable load-and-clear operation (load an accumulator with the contents of a memory location and then clear the location) as its primitive locking facility (i.e., as the necessary multiprocessor lock equivalent to Dijkstra semaphores). To avoid deadlocks, we assign a priority ordering to our resources and arrange that the software not lock one resource when it has already locked another of lower or equal priority.

Using these facilities and techniques, the logical structure of the IMP program is similar to that described for the 516/316 machines. In addition there is a substantial new program segment devoted to maintaining the hardware and keeping the program running in the face of hardware failures.

2.3.5.3 PLURIBUS IMP Reliability

Computer reliability is a common, serious, and difficult problem which has been approached in many ways. For critical applications (e.g., space exploration), large amounts of money are spent to overcome such apparently trivial weaknesses as problematical power supplies and connectors. Although a great deal of attention is given to tailoring computers to particular job environments, the commercial world of computer manufacturers has provided no adequate answer to the reliability problem.

The notions of faulttolerant and failsafe systems have been around for a number of years and because the reliability is such a crucial issue in a communications network, it was decided that some of these ideas should be exploited in the design of the PLURIBUS IMP.

The reliability goal of the PLURIBUS IMP is not that the system should never break, but rather that it should recuperate automatically within seconds or minutes from most troubles and that the probability of total failure should be dramatically reduced over traditional machines, say to once a year or less. The system should survive not only transient failures but also solid failures of any single component. It is not necessary to operate correctly most of the time so long as outages are infrequent, kept brief, and fixed without human intervention. Outages of a few seconds are tolerated easily, and outages of many seconds, while causing the particular node to become temporarily unusable, will not in general jeopardize operation of the network as a whole.

Achieving this sort of reliability requires hardware that will survive any single failure, even a solid one, in such a way as to leave a potentially runnable machine intact (potentially in that it may need resetting, reloading, etc.). Second, it requires all of the facilities necessary to survive any and all transients stemming from the failure and to adapt to running in the new hardware configuration. To provide adequate hardware, extra copies of every vital hardware resource are included. Sufficient isolation is provided between the copies so that any single component failure will impair only one copy.

Appropriate hardware. It is not sufficient merely to provide duplicate copies of a particular resource; it is necessary to assure that the copies are not dependent on any common resource. Thus, for example, in addition to providing multiple memories, there are multiple busses on which the memories are distributed. Furthermore, each bus is not only logically independent, but also physically modular. The chassis, with its own power supply and cooling, is built into an integral unit which may be powered down, disconnected, and removed from the rack for servicing or replacement while the rest of the machine continues to run.

All central system resources, such as the real time clock and the PID, are duplicated on at least two separate I/O busses. All connections between bus pairs are provided by separate bus couplers so that a coupler failure can disable at most the two busses it is connecting; all other interconnections between busses are unaffected.

When a particular communications circuit is deemed critical, it is connected to two identical interface units (on separate I/O busses), either of which may be selected for use by the program. When the extra reliability is not worth the extra cost, the line is only singly connected.

In order for the system to adapt to different hardware configurations, facilities have been provided which make it convenient for the software to search for and locate those resources which are present and to determine the type and parameters of those which are found.

To allow for active failures, all bus couplers have a program-controllable switch that inhibits transactions via that coupler. Thus, a "malicious" bus may be effectively "amputated" by turning off all couplers from that bus. These switches are protected from capricious use by requiring a password. Naturally an amputated processor has no access to these switches.

__Software Survival.__ With the above features, the PLURIBUS hardware can experience any single component failure and still present a runnable system. One must assume that as a consequence of a failure, the program may have been destroyed, the processors halted, and the hardware put in some hung state needing to be reset. Three broad strategies have guided the means used to restore the algorithm to operation after a failure: keep it simple, worry about redundancy, and use watchdog timers throughout.

SIMPLICITY: First, all processors are identical and equal; they are viewed only as resources used to advance the algorithm. Each is able to do any system task; none is singled out (except momentarily) for a particular function. A consequence of this is that the full power of the machine can be brought to bear on the part of the algorithm which is busiest at a given time. A further consequence is that should any processor fail, the rest will continue to perform the necessary tasks, albeit at reduced capacity.

A second system characteristic which arose from a desire to keep things simple is passivity. The terms active and passive describe communication between subsystems in which the receiver is expected to put aside what it is doing and respond. The quicker the required response, the more active the interaction. In general, the more passive the communication, the simpler the receiver can be, because it can wait until a convenient time to process the communication. Neither the hardware interfaces nor other processors tell a processor what to do; rather, tasks to be done are posted in the PID and processors ask the PID what should be done next.

There are some costs to such a passive system: first, the resulting slower responsiveness has necessitated additional buffering in some of the interfaces; second, the program must regularly break from tasks being executed to check the PID for more important tasks. The alternatives, however, are far worse. In a more active system, for example one which uses classical priority interrupts, it is difficult to decide which processor to switch to the new task. The possibilities for deadlocks are frightening, and the general mechanism to resolve them cumbersome.

As a third example of simplicity, the entire system is broken into reliability subsystems which are parts of the overall system that verify one another in an orderly fashion. The subsystems are cleanly bounded with well-defined interfaces. They are self-contained in that each includes a self-test mechanism and reset capability. They are isolated in that all communication between subsystems takes place passively via data structures. Complete interlocking is provided at the boundary of every subsystem so that the subsystems can operate asynchronously with respect to one another.

The monitoring of one subsystem by another is performed using timer modules, as discussed below. These timer modules guarantee that the selftest mechanism of each subsystem operates, and this in turn guarantees that the entire subsystem is operating properly.

REDUNDANCY: Redundancy is simultaneously a blessing and a curse. It occurs in the hardware and the software, and in both control and data paths. We deliberately introduce redundancy to provide reliability and promote efficiency, and it frequently occurs because it is a natural way to build things. On the other hand the mere existence of redundancy implies a possible disagreement between the versions of the information. Such inconsistencies usually lead to erroneous behavior, and often persist for long periods.

There are several methods of dealing with redundancy. The first and best is to eliminate it, and always refer to a single copy of the information. When we choose not to eliminate it, we check the redundancy and explicitly detect and correct any inconsistencies. It does not really matter how this is done as the system is recovering from a failure anyway. What is important is to resolve the inconsistency and keep the algorithm moving. Sometimes it is too difficult to test for inconsistency; then timers are used as discussed below.

TIMERS: There is a uniform structure for implementing a monitoring function between reliability subsystems based on watchdog timers. Consider a subsystem which is being monitored. Such a subsystem is designed to cycle with a characteristic time constant, and a complete self-consistency check is included within every cycle. Regular passage through this cycle is therefore sufficient indication of correct operation of the subsystem. If excessive time goes by without passage through the cycle, it implies that the subsystem has had a failure from which it has not been able to recover by itself. The mechanism for monitoring the cycle is a timer which is reset by every passage through the cycle. There are both hardware and software timers ranging from five microseconds to two minutes in duration in the IMP system. Another subsystem monitors this timer and takes corrective action if the timer ever runs out. To avoid the necessity for subsystems to be aware of one another's internal structure, each subsystem includes a reset mechanism which may be externally activated. Thus, corrective action consists merely of invoking this reset. The reset algorithm is assumed to work although a particular incarnation in code may fail because it gets damaged. In such a case another subsystem (the code checksummer) will shortly repair the damage.

The entire system consists of a chain of subsystems in which each subsystem monitors the next member of the chain. Lower subsystems provide and certify some important environmental feature used by higher level systems. For example, a low level code tester checksums all code (including itself), insures that all subsystems are receiving a share of the processor's attention, and guarantees that locks do not hang up. It thus guarantees the most basic features for all higher levels. These will, in turn, provide further environmental features, such as a list of working memory areas, I/O devices, etc., to still higher levels.

Before they can work together to run the main system, a common environment must be established for all processors. The process of reaching an agreement about this environment is called "forming a consensus", and the group of agreeing processors is known as the Consensus. An example of a task requiring consensus is the identification of usable common memory and the assignment of functions (code, variables, buffers, etc.) to particular pages.

The Consensus maintains and counts down a timer for every processor in the system in order to detect uncooperative or dead processors. This monitoring mechanism includes reloading the failing processor's local memory and restarting it. Reliance on the Consensus is vulnerable to simultaneous transient failure of all processors. For many cases (as for example when all of the processors halt), a simple reset consisting of a one-second timer on the bus and a 60 Hz interrupt routine suffices.

For more catastrophic failures the machine can be reset, reloaded, and restarted directly from the Network Control Center, which depends on the continual presence of human operators for successful operation. It is correspondingly powerful, resourceful, and erratic in its behavior.

2.4 Subnet Protocols

We begin this part of our report with a brief summary of four important protocols: IMP-IMP, Routing, Source IMP-Destination IMP, and Host-Host.

<u>IMP-IMP Protocol.</u> A technique has been adopted for IMP-to-IMP transmission control in which each physical inter-IMP circuit is broken into a number of logical channels. Acknowledgements are returned piggybacked on normal network traffic in a set of acknowledgement bits, one bit per channel, contained in every packet, thus requiring less bandwidth than sending each acknowledgement in its own packet. Normally the number of logical channels is eight per physical line. A pair of IMPs connected by a high delay line such as a transoceanic satellite line may, however, be configured to use sixteen logical channels on that line in order to utilize the line more fully.

Each packet is assigned to an outgoing logical channel and carries the odd/even bit for its channel (which is used to detect duplicate packet transmissions), its channel number, and the acknowledge bits - one for each channel in the reverse direction. The transmitting IMP continually cycles through its used channels (those with packets associated with them), transmitting the packets along with the channel number and the associated odd/even bit. At the receiving IMP, if the odd/even bit of the received packet does not match the odd/even bit associated with the appropriate receive channel, the packet is accepted and the receive odd/even bit is complemented; otherwise the packet is a duplicate and is discarded.

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

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

Each packet is individually routed from IMP to IMP through the network toward the destination. At each IMP along the way, the transmitting hardware generates initial and terminal framing characters and checksum digits that are shipped with the packet and are used for error detection by the receiving hardware of the next IMP.

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

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

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

The routing table is periodically and dynamically updated to adjust for changing conditions in the network. The system is adaptive to the ups and downs of lines, IMPs, and congestion; it does not require the IMP to know the topology of the network. In particular, an IMP need not even know the identity of its immediate neighbors. Thus, the leased circuits could be reconfigured to a new topology without requiring any changes to the IMPs.

Source IMP-Destination IMP Protocol. In order for a source Host to communicate with a destination Host, both source and destination IMPs must establish a record of the connection for that Host pair. This simplex connection, consisting of a Transmit Message Block at the source, and a corresponding Receive Message Block at the destination, is created, and later removed, using a special protocol which detects duplicate or missing messages. The connection is disallowed if the Host/Host access control mechanism does not permit that Host pair to communicate. A pair of Hosts may communicate with each other only if they are members of the same logical subnetwork or if one is allowed to communicate with Hosts in a subnetwork of which the other is a member.

To insure that messages arrive at a destination Host in proper order, an independent message number sequence is maintained for each connection. A message number is assigned to each message at the source IMP and this message number is checked at the destination IMP. Out of an eight-bit message number space, both the source and destination keep a small window (currently eight) of valid message numbers, which allows several messages to be in flight simultaneously. Messages arriving at a destination IMP with message numbers outside of the current window or with message numbers already marked as received are duplicates to be discarded. The message number concept serves two purposes: it orders the messages for delivery to the destination Host, and it provides for the detection of duplicate and missing messages. The message number is internal to the IMP subnetwork and invisible to the Hosts.

A sequence control system based on a single source/destination connection, however, does not permit priority traffic to go ahead of other traffic. More generally, a Host may wish to specify a particular treatment for each message; thus, a separate connection is created for each "handling type". Currently, there are two possible handling types, regular (for high bandwidth) and priority (for low delay).

Since message numbers and reserved storage are so critical in the system, very stringent and careful procedures were developed to account for a lost message. The source IMP keeps track of all messages for which a RFNM has not yet been received, and the destination IMP keeps track of the replies it either has yet to send or has already sent. When the RFNM is not received for too long (presently about 30 seconds), the source IMP sends a control message (using the same message number) to the destination inquiring about the possibility of an incomplete transmission. Depending on the state of the reply table and message window at the destination, it will respond with either an indication that the message was not received or that it is out of range, or with a correct duplicate reply (RFNM). The source IMP continues inquiring until it receives a response. This technique generally insures that the source and destination IMPs keep their message number sequences synchronized and that any allocated space will be released should a message become lost in the subnetwork because of a machine failure.

A connection is terminated either after a prolonged period of inactivity (presently 3 minutes), or a somewhat shorter period of inactivity coupled with the need for the Message Block by some other connection, or by the need to resynchronize a message number sequence that has been broken. The special termination protocol can be initiated by either the source or the destination in the first two cases above, or by the source in the third case, upon the receipt of an "out of range" response to an incomplete query. Upon closing a connection, both source and destination release all resources held or allocated for that connection.

There is a facility outside of the normal Host/Host connection mechanism for sending and receiving a stream of "raw packets". These messages are identified by a special Host-IMP and IMP-Host code and bypass the connection mechanism. They are routed normally through the subnetwork, but no sequencing, error control, reassembly, or storage allocation is performed. Thus, they may arrive out of order at the destination Host, some packets may be missing or duplicated, or packets may be thrown away by the subnetwork if insufficient resources are available to handle them. No RFNMs or other messages are sent back to the source Host about such raw packets.

Host-Host Protocols. A network working group, comprised of system oriented program designers from the research installations, was formed to define a protocol and develop techniques for communication between computers. Some constraints were imposed by the Host-IMP protocol and by the wide variance in specifics of the operating systems involved. A Host-Host protocol was specified, detailing procedures for establishing "logical" connections between a pair of Host computers wishing to communicate. The Host-IMP, Host-Host protocols are implemented as a network control program (NCP).

The NCP is typically implemented as part of the executive system of the Host computer. The NCP is responsible only for communicating with the IMP and with the NCP in other Hosts. Its primary functions are establishing, breaking, and switching connections between user-level processes in a pair of Hosts, and controlling the flow of byte-streams transmitted over the established connection. System primitives are provided at the user-level interface whereby user-level processes direct the NCP to establish or terminate connections with user-level processes in other Hosts. Thus user-level processes maintain control over the timing of transmission sequences and over the interpretation of the contents of the byte-streams transmitted.

The communications subnetwork and the Host-Host protocol provide a framework or set of building blocks for the development of user-level capabilities. Several function-oriented protocols for the more common uses of the network have been specified by the network working group. Each protocol is implemented as a pair of cooperating user-level processes; a "user" process and a "server" process. The "user" process appears as a subsystem to the network user. When activated by a user at a local Host, the "user" process interacts with the NCP to establish a connection with the corresponding "server" process in a remote Host. When using these subsystems, the functions performed by the communications subnet and the NCP are transparent to the user.

A user can log into a local host and activate a subsystem called TELNET which performs the functions necessary to connect him to any other host on the network. Alternatively, a user can connect to a TIP to access any network facility. For all practical purposes, a user will appear both to himself and to the remote host as a user directly connected to the remote host. Once a connection is established, he follows the conventions of the remote host for logging in, executive functions, and for use of subsystems.

The TELNET protocol specifies a "virtual network terminal" which resolves differences in echoing conventions, interrupt signals, device rates, character sets, etc. Thus almost any type of local terminal can be used with any remote host. Many TELNET subsystems perform other peripheral functions such as saving a transcript of a session in a local file, sending a command file in place of user-typed input, and reporting whether various hosts are or have been up.

Many ad hoc protocols have been implemented to provide capabilities for transferring files between hosts, for remote job entry to batch systems, and for interactively using graphics systems. These implementations generally require explicit user intervention and have not solved the difficult problem of dealing with incompatible data streams. There is an ongoing effort within the network working group to define network standard protocols for these functions which will handle the incompatibilities in a uniform manner. The protocols are in varying stages of development, including some experimental implementations.

It should be noted that the primitives provided by the NCP are available to any user-level process. Thus any user can build his own network functions by explicitly interacting with the NCP primitives making them an integral part of his application.

2.4.1 IMP-IMP Protocols

In this section we discuss some of the issues in designing node-to-node transmission procedures, that is, the packet processing algorithms. We touch on these points only briefly since many of them are simple or have been discussed previously. Note that many of these issues occur again in the discussion of source-to-destination transmission procedures.

2.4.1.1 Basic Concepts

Buffering and Pipelining. As we noted in discussing memory requirements, the amount of node-to-node packet buffering needs to equal the product of the circuit rate times the expected acknowledgment delay in order to get full line utilization. It may also be efficient to provide a small amount of additional buffering to deal with statistical fluctuations in the arrival rates, i.e., to provide queueing. These requirements imply that the nodes must do bookkeeping about multiple packets, which raises the several issues discussed next.

Error Control. We have discussed many of the aspects of node-to-node error control above; the need for a packet checksum, its size, the basis of the acknowledgment/retransmission system, the decision on whether the line is usable, and so on. These procedures are critical for network reliability, and they should therefore run smoothly in the face of any kind of node or circuit failure. Where possible, the procedures should be self-synchronizing; at least they should be free from deadlock and easy to resynchronize [24].

Storage Allocation and Flow Control. Storage allocation can be fairly simple for the packet processing algorithms. The sender must hold a copy of the packet until it receives an acknowledgment; the receiver can accept the packet if it is without error and there is an available buffer. The receiver should not use the last free buffer in memory, since that would cut off the flow of control information such as routing and acknowledgments. In accepting too many packets, there is also the chance of a storage-based deadlock in which two nodes are trying to send to each other and have no more room to accept packets. This is explained fully in [19].

The above implies that the flow control procedures can also be fairly simple. The need to buffer a circuit can be expressed as a quantitative limit of a certain number of packets. Therefore, the node can apply a cut-off test per line as its flow control throttle. More stringent rules can be used, but may be unnecessary.

Priority. The issue of priority in packet processing is quite important for network performance. First of all, the concept of two or more priority levels for packets is useful in decreasing queueing delay for important traffic. Beyond this, however, careful attention must be paid to other kinds of transmissions. Routing messages should go with the highest priority, followed by acknowledgments (which can also be piggy-backed in packets). Packet retransmissions must be sent with the next highest priority, higher than that for first transmission of packets. If this priority is not observed, retransmissions can be locked out indefinitely. The question of preemptive priority (i.e., stopping a packet in mid-transmission to start a higher priority one) is one of a direct tradeoff of bandwidth against delay since circuit bandwidth is wasted by each preemption.

Packet Size. There has been much thought given in the packet-switching community to the proper size for packets. Large packets have a lower probability of successful transmission over an error-prone telephone line (and this drives the packet size down), while overhead considerations (longer packets have a lower percentage overhead) drive packet size up. The delay-lowering effects of pipelining become more pronounced as packet size decreases, generally improving store-and-forward delay characteristics; further, decreasing packet size reduces the delay that priority packets see because they are waiting behind full length packets. However, as the packet size goes down, potential effective throughput also goes down due to overhead. Metcalfe has previously commented on some of these points [28].

It has been suggested that the ARPA Network packet size is suboptimal and should perhaps be reduced from about 1000 bits to 250 bits. This is based on optimization of node buffer utilization for the observed traffic mix in the network. However, the relative cost of node buffer storage vs. circuits is possibly such that one should not try to optimize node buffer storage. The true tradeoff which governs packet size might well be efficient use of phone line bandwidth (driving packet size larger) vs. delay characteristics (driving packet size smaller). If buffer storage is limiting, one should just buy more (up to the limits of the address space, of course). Further, it is probably true that if one is trying for high bandwidth utilization, buffer size must be large. That is, high bandwidth utilization probably implies the use of large packets, which implies full buffers; when idle, the buffer size does not matter.

As noted above, the choice of packet size is influenced by many factors. Since some of the factors are inherently in conflict, an optimum is difficult to define, much less find. The current ARPA Network packet size of about 1000 bits is a good compromise. Other packet sizes (e.g., the 2000 bits used in several other networks) may also be acceptable compromises.

However, note that a 2000-bit packet size generally means a factor of two increase in delay over a 1000-bit packet size, because even high priority short packets will be delayed behind normal long packets which are in transmission at each node. The use of preemptive priority might make longer packet sizes efficient.

Other authors have been cited as recommending a minimum length "packet" of about 2000 bits because they have concluded that most of the messages currently exchanged within banks and airlines fit nicely in one packet of this size. To clarify this point, we note that they use the term "packet" for the unit of information we call a "message" and thus are not actually addressing the issue of packet size. We discuss message size below.

2.4.1.3 Subsequent Modifications

We now take a close look at the algorithm used in the ARPA Network for IMP-to-IMP transmission control. As has been noted elsewhere, the inter-IMP modem interface hardware has the capability of generating checksums for outgoing packets and checking the checksums on incoming packets. This allows packets which are damaged in transmission to be detected and discarded without acknowledgment. Packets correctly received are acknowledged. A good IMP-to-IMP transmission control algorithm must detect errors, acknowledge good transmissions, and provide retransmission in the event of errors. In addition, the IMP-to-IMP transmission control algorithm is improved if it detects duplicates that are sometimes generated by retransmission. An algorithm which performs all four of these tasks is described below (it is like the HDLC data communications standard with a 1-bit sequence number per logical channel and up to 8 logical channels).

A number of logical "channels" are maintained between each pair of IMPs. Consider only one channel to begin with, and further consider packet transmissions in only one direction on this channel. Of course, acknowledgments go the other direction on the channel. At both the transmit and receive end of this channel a one bit sequence number is kept. We call this bit an odd/even bit. Both transmit and receive odd/even bits are initialized to be zero. Also, at the transmit end, a used/unused bit is kept for the channel. It is of course initialized to zero, meaning unused. When it is time to transmit a packet, a check is first made for the channel's being unused. If it was previously unused, it is marked as used and the packet is transmitted. The state of the transmit odd/even bit is included with the packet. When the packet arrives at the receiver, assuming the packet is received correctly, the packet's odd/even bit is checked against the receive odd/even bit. If they match, the packet is accepted and the receive odd/even bit is complemented. Otherwise, the packet would be ignored. In any case the receive odd/even bit is returned as an acknowledgment. At the transmitter, if the acknowledgment bit does not match the transmit odd/even bit, the packet has been successfully sent and acknowledged and the packet can be discarded, the channel marked unused, and the transmit odd/even bit complemented. Otherwise the acknowledgment is a duplicate and is ignored. Suppose now a second copy of the packet arrives at the receiver, a packet which was sent before the first acknowledgment had a chance to get back to the transmitter. When this packet arrives at the receiver, its odd/even bit does not match the receive odd/even bit and so that packet is discarded as a duplicate. Nonetheless, an acknowledgment is sent for the packet using the present state of the receive odd/even bit. When the acknowledgment gets to the transmitter, it does match the transmit odd/even bit, so the acknowledgment is a duplicate and is ignored.

The acknowledgment bit is the state of the receive odd/even bit after it is complemented rather than before it is complemented. Hence the need for the "not match" rule when the acknowledgment arrives at the transmitter.

Because of the potentially long distances between IMPs, one channel is not enough to keep the inter-IMP lines fully loaded. Therefore, eight logical channels are supplied between each pair of IMPs (32 are supplied between Satellite IMPs). It is not necessary to maintain ordering of IMP-to-IMP transmissions since packet ordering is performed at the destination IMP. This means that the transmit channels can be filled in any convenient order, and at the receive side, packets can be forwarded onwards as soon as they are correctly received regardless of the channel over which they arrived.

To avoid requiring separate packets for acknowledgments, acknowledgment bits are "piggy-backed" in packets going the other way on the line. In fact, all eight receive odd/even bits are transmitted with every packet going the other way. In the absence of any traffic going the other way on the line, a packet carrying only the eight acknowledgments is sent. In either case, the acknowledgments get back to the transmitter as fast as possible. Therefore, the transmitter knows very soon whether a packet requires retransmission or not, allowing the use of a minimal timeout before retransmission. "Piggy-backing" all acknowledgments into every packet going the other way saves program bandwidth, line bandwidth, and buffer space over a system which sends individual acknowledgments in individual packets.

In view of the use of a number of channels and the delay encountered on long lines, some packets might have to wait an inordinately long time for transmission. Traffic that is essentially interactive should not be subjected to waiting for several thousand-bit packets to be transmitted, multiplying by ten or more the effective delay seen by the source. Therefore, the IMPs maintain two sets of queues for each output line, and service all priority transmissions before any regular transmissions. Preemptive priority is not employed. The following transmission ordering scheme has been instituted: priority packets which have never been transmitted are sent first; next sent are any regular packets which have never been transmitted; finally, if there are no new packets to send, previously transmitted packets are periodically retransmitted even when there is a continuous stream of new traffic. Routing has priority over all packets.

Figure 2-14 shows the format of packets as they appear on the inter-IMP circuits. Packets are permitted to be variable length up to a maximum of about 1000 bits. The IMP's modem interface transmission hardware adds framing characters to each packet as it is transmitted onto an inter-IMP circuit. At the front of the packet two characters (DLE and STX) are added indicating the beginning of the packet. At the end of the packet two characters (DLE and ETX) are added indicating the end of the packet. The checksum is appended after the end of packet characters. The IMP's modem interface reception hardware has the capability of detecting the start and the end of a packet from these framing characters thus freeing the program from the burden of detecting packet boundaries. Between packets the transmission hardware automatically generates synchronizing characters (SYN) which the reception hardware automatically discards.

There is no restriction on the content of a packet. Arbitrary sequences of bits may be transmitted without restriction. This transparency is achieved by a method known as DLE-doubling. If the data in the packet itself contains a DLE-character (the character which introduces the packet start and packet end sequences), that DLE-character in the data is doubled by the transmission hardware. At the receiver, the hardware collapses double DLEs in the data back into one; so there is complete transparency on the inter-IMP channels. This is a very important point. All too many networks require transmission to be limited to characters from a particular character set. Besides preventing the network's users from sending arbitrary messages, the designers of such networks are themselves prevented from such things as loading programs over the network.

In addition to showing the part of the packet format done with hardware, Figure 2-14 also shows the channel field, "piggy-back" acknowledgment field, etc., mentioned in the preceding section. The portions of the packet which carry the end-to-end control information (e.g., destination address, message sequence numbers, etc.) are also shown.

Figure 2-14: Packet Format

Notice that much of the inter-IMP communication algorithm is performed with hardware. Software is particularly bad at generating powerful checksums, scanning for packet boundaries, and so forth, especially when such chores must be done on a character-by-character or bit-by-bit basis. Therefore, they are done with hardware in the ARPA Network.

Consider Figure 2-15. Two-way single-packet traffic flows between A and A' and also between B and B', and is constrained by network topology to use the circuit between IMPs C and D. Suppose all the buffer storage in IMP C can become filled with packets on the output queue to IMP D, and all the buffer storage in IMP D can become filled with packets on the output queue to IMP C. In this case, both IMP C and IMP D would be engaged in a direct confrontation in which both IMPs must lose all incoming packets (and acknowledgments) as neither machine has any buffer space with which to receive inputs. We call this a store-and-forward lockup.

It is straightforward to prevent such store-and-forward lockups, and this is done in the ARPANET implementation. The key technique used is to guarantee that sufficient buffers are reserved so that it is always possible to input and to output one packet over each circuit. Thus, packets (and acknowledgments) can always be passed between neighboring machines (albeit at a trickle, perhaps). In particular, in the IMP system, one buffer

Figure 2-15: Store-and-Forward Lockup

is always allocated for output on each line, guaranteeing that output is always possible; and double buffering is provided for input on each line, which permits all input traffic to be examined by the program, so that acknowledgments can always be processed, which frees buffers. Additionally, an attempt is made to provide enough store-and-forward buffers so that all lines may operate at full capacity.

2.4.1.4 Conclusions

We conclude this section with some comments: 1) negative acknowledgments could be useful in activating dormant buffers more quickly, but they add complexity and are not used in the ARPANET; 2) more complex forms of store-and-forward lockup than that given in the example above are possible, and while most are protected against in the ARPANET implementation, at least one rare case is not protected against.

2.4.2 The Routing Algorithm

2.4.2.1 Basic Concepts

The fundamental step in designing a routing algorithm is the choice of the control regime to be used in the operation of the algorithm. Non-adaptive algorithms make no real attempt to adjust to changing network conditions; no routing information is exchanged by the nodes, and no observations or measurements are made at individual nodes. Centralized adaptive algorithms utilize a central authority which dictates the routing decisions to the individual nodes in response to network changes. Isolated adaptive algorithms operate independently with each node making exclusive use of local data to adapt to changing conditions. Distributed adaptive algorithms utilize internode cooperation and the exchange of information to arrive at routing decisions.

2.4.2.1.1 Non-adaptive Algorithms

Under this heading come such techniques as fixed routing, fixed alternate routing, and random routing (also known as flooding or selective flooding).

Simple fixed routing is too unreliable to be considered in practice for networks of more than trivial size and complexity. Any time a single line or node fails, some nodes become unable to communicate with other nodes. In fact, networks utilizing fixed routing always assume manual updates (as necessary) to another fixed routing pattern. However, in practice this would mean that every routine network component failure becomes a catastrophe for operational personnel, every site spending frantic hours manually reconstructing routing tables.

At their best, in the absence of network component failure, fixed routing algorithms are inefficient. While the routing tables can be fixed to be optimal for some traffic flow, fixed routing is inevitably inefficient to the extent that network traffic flows vary from the optimal traffic flow. Unreliability and inefficiency are also characteristic of two alternative techniques to fixed routing which fall under the heading of non-adaptive algorithms: fixed routing with fixed alternate routes and random routing.

Non-adaptive algorithms are all extremely simple and can therefore be implemented at low cost. They are thus possibly suitable for hardware implementation, for theoretical analysis, and for studying the effects of varying other network parameters and algorithms.

In conclusion, we do not recommend non-adaptive routing for most networks because it is unreliable and inefficient. Despite these drawbacks, many networks have been proposed or begun with non-adaptive routing, generally because it is simpler to implement and to understand. Perhaps this tendency will be reversed as more information about other routing techniques is published and as network technology generally grows more sophisticated.

2.4.2.1.2 Centralized Adaptive Algorithms

In a centralized adaptive algorithm, the nodes send the information needed to make a routing decision to a Routing Control Center (RCC) which dictates its decision back to the nodes for actual use. The advantages claimed for a centralized algorithm are: a) the routing computation is simpler to understand than a non-centralized algorithm, and the computation itself can follow one of several well known algorithms; b) the nodes are relieved of the burden and overhead of the routing computation; c) more nearly optimal routing is possible because of the sophistication that is possible in a centralized algorithm; and d) routing "loops" (a possible temporary property of distributed algorithms) can be avoided.

Unfortunately, the processor bandwidth utilization at the center is likely to be very heavy. The classical algorithms that a centralized approach might use generally run in time proportional to N cubed (where N is the number of nodes in the network), while their distributed counterparts can run (through parallel execution) in time proportional to N squared. While it may be a saving to remove computation from the nodes, it may not be possible to perform a cubic computation on a large network in real time on a single computer, no matter how powerful.

The claim that more optimal routing is possible with a centralized approach is not true in practice. To have optimal routing, the input information must be completely accurate and up-to-date. Of course, with any realistic centralized algorithm, the input data will no longer be completely accurate when it arrives at the center. Similarly, the output data -- the routing decisions -- will not go into effect at the nodes until some time after they have been determined at the center.

Distributed routing algorithms, whether fixed random, fixed alternate, or adaptive, may contain temporary loops; that is, a packet may traverse a complete circle while the algorithm adapts to network change. Proponents of centralized routing often argue that such loops can best be avoided by centralization of the computation. However, because of the time lags cited above, there may indeed be loops during the time of propagation of a routing update when some nodes have adopted the new routes and other nodes have not.

A centralized routing algorithm has several inherent weaknesses in the updating procedure, the first being unreliability. If the RCC should fail, or the node to which it is connected goes down, or the lines around that node fail, or a set of lines and nodes in the network fail so as to partition the network into isolated components, then some or all of the nodes in the network are without any routing information. Of course, several steps can be taken to improve on the simple centralized policy. First, the RCC can have a backup computer, either doing another task until a RCC failure, or else on hot standby. This is not sufficient to meet the problem of network failures, only local outages, but it is necessary if the RCC computer has any appreciable failure rate. Second, there can be multiple RCCs in different locations throughout the network, and again the extra computers can be in passive or active standby. Here there is the problem of identifying which center is in control of which nodes, since the nodes must know to which center to send their routing input data.

A related difficulty with centralized algorithms lies in the fact that when a node or line fails in the network, the failed component may have been on the previously best path between the RCC and the nodes trying to report the failure. In this case, just at the time the RCC needs routes over which to receive and transmit routing information, no routes are available; the availability of new routes requires the very change the RCC is unsuccessfully attempting to distribute. Solutions which have been proposed to solve this "deadlock" are slow, complicated, awkward, and frequently rely on the temporary use of distributed algorithms.

Finally, centralized algorithms can place heavy and uneven demands on network line bandwidth as shown in Figure 2-16, near the RCC there is a concentration of routing information going to and from the RCC. This heavy line utilization near the center means that centralized algorithms do not grow gracefully with the size of the network and, indeed, this may place an upper limit on the size of the network.

2.4.2.1.3 Isolated Adaptive Algorithms

One of the primary characteristics of an isolated algorithm which attempts to adapt to changing conditions is that it takes on the character of a heuristic process; it must "learn" and "forget" various facts about the network environment. While such an approach may have an intuitive appeal, it can be shown rather simply that heuristic routing procedures are unstable and are therefore not of interest for most practical network applications. The fundamental problem with isolated adaptive algorithms is that they must rely on indirect information about network conditions, since each node operates independently and without direct knowledge of or communication with the other nodes.

There are two basic approaches to be employed, seperately or in tandem, to the process of learning and forgetting. These approaches can be termed positive feedback and negative feedback. One way to implement positive feedback was suggested by Baran as

Figure 2-16 Centralized Routing Algorithms part of his hot-potato routing doctrine. Each node increments the handover number in a packet as it forwards the packet. Then the handover number is used in a "backwards learning" technique to estimate the transit time from the current node to the source of the packet. Clearly, this scheme has drawbacks because it lacks any direct way of adapting to changes. If no packets from a given source are routed through a node by the rest of the network, the node has no information about which route to choose in sending a message to that source. In general, as part of a positive feedback loop, the routing algorithm must periodically try routes other than the current best ones, since it has no direct way of knowing if better routes exist. Thus, there must always be some level of traffic traveling on any route that the nodes are to learn about, since it is only by feedback from traffic that they can learn.

The other half of an adaptive isolated algorithm is the negative feedback cycle. One technique to use here is to penalize the choice of a given path when a packet is detected to have returned over the same path without being delivered to its destination. The relation of this technique to the exploratory nature of positive feedback is evident.

An adaptive isolated algorithm, therefore, has this fundamental weakness: in the attempt to adapt heuristically, it must oscillate, trying first one path and then another, even under stable network conditions. This oscillation violates one of the important goals of any routing algorithm, stability, and it leads to poor utilization of network resources and slow response to changing conditions. Incorrect routing of the packets during oscillation increases delay and reduces effective throughput correspondingly. There is no solution to the problem of oscillation in such algorithms. If the oscillation is damped to be slow, then the routing will not adapt quickly to improvements and will therefore declare nodes unreachable when they are not, with the result that suboptimal paths will be used for extended periods. If the oscillation is fast, then suboptimal paths will also be used much of the time, since the network will be chronically full of traffic going the wrong way.

2.4.2.1.4 Distributed Adaptive Algorithms

In our experience, distributed adaptive algorithms have none of the inherent limitations of the above algorithms; e.g., not the inherent unreliability and inefficiency of non-adaptive algorithms, nor the unreliability and size limitations of centralized algorithms, nor the inherent inefficiency and instability of isolated algorithms. For example, the distributed adaptive routing algorithm in the ARPANET has operated for five years with little difficulty and good performance. However, distributed algorithms do have some practical difficulties which must be overcome in order to obtain good performance.

Consider the following example of a distributed adaptive algorithm. Each node estimates the "distance" it expects a packet to have to traverse to reach each possible destination over each of its output lines. Periodically, it selects the minimum distance estimate for each destination and passes these estimates to its immediate neighbors. Each node then constructs its own routing table by combining its neighbors' estimates with its own estimates of distance to each neighbor. For each destination, the table is then made to specify that selected output line for which the sum of the estimated distance to the neighbor plus the neighbor's distance estimate to the destination is smallest.

Such an algorithm can be made to measure distance in hops (i.e., lines which must be traversed), delay, or any of a number of other metrics including excess bandwidth and reliability (of course, for the latter two, one must maximize rather than minimize). The above algorithm is representative of a class of distributed adaptive algorithms which we consider briefly in the remainder of this section. For simplicity of discussion we will assume that distance is measured in hops.

2.4.2.1.5 Routing Processing Goals

In this section we outline some of the desirable characteristics for the processing of routing information. There are six divergent goals for the routing algorithm in its task of accepting the input data about the network and generating the required output.

Simplicity. Simplicity is the goal we list first, because it assumes increasing importance as further requirements are placed on the routing algorithm and the trend to complexity grows. There are two distinct kinds of simplicity that are of great value here, and in any computer algorithm. First, it is almost essential that the routing program running in a network be simple enough for a man to understand what is happening in a given situation. This is desirable not from the point of view of keeping the man in control, but merely to permit him to follow the operation of the algorithm and find problems and suggest improvements in its performance. This may sound trivial, but in a very large network even the most basic measurements and the most elementary observations are difficult to undertake. Thus it is useful if the routing algorithm itself does not present further complexities to the man trying to interpret its behavior. In practice, it should be possible to understand or predict the behavior of the routing algorithm without difficulty. For example, order-dependent or non-deterministic rules may be too obscure to follow, particularly in a network environment.

The second kind of simplicity that is desirable is simplicity in the design and structure of the algorithm, so that it can be coded in a small, simple program. It is more likely that the program will be efficient and reliable if it can be expressed simply to the computer. It is always good software design to write simple programs, and this is especially true in a network environment, where the programs must run continuously in real time, and run on many computers.

Reliability. It is critical that the routing algorithm be reliable in the face of node and line failures. Such failures must be expected, and, indeed, when they occur the successful operation of the routing calculation is most essential. Therefore, the momentary or prolonged malfunction of any component in the network should not interfere with the steady, accurate process of calculating the best routes for traffic in the network. When parts of the network are isolated from each other, and when they are reunited, the changeover should be managed smoothly. If a node fails to receive one or two routing messages due to line errors or node problems, or receives extra messages, or erroneous data, the global reliability of the distributed computation should not be affected.

Steady State Solution. A basic requirement of any routing algorithm is that, given a static set of inputs, it should arrive at a steady state solution which is accurate and stable. This is a very elementary goal, but one that should not be overlooked. The operation of the algorithm should not be so approximate that its outputs oscillate under static input conditions. For instance, random or heuristic algorithms are undesirable for this reason. Also, order-dependent algorithms and techniques with many special cases may not always arrive at the same solution to a given set of routing input data with slight changes in the specifications. Attention to this goal is particularly important in the early stages of designing an algorithm. Then, when test cases are being thought up, it is essential that the algorithm arrive at an accurate and stable solution for all the simple static tests that can be devised.

This means, for instance, that in thinking about algorithms that should work for a network with a connectivity that is assumed to change, one should test the algorithm on the following kinds of networks:

single node

loop networks

star networks

tree networks

union of a loop and a tree

union of two loops

series-parallel combinations

and so on. The algorithm should also be tested with various traffic requirements, such as:

no traffic

one-way traffic along a single path

two-way traffic along a single path

two traffic streams on separate paths

two traffic streams with a line in common

and so on. Solving the routing problem for a static situation is much easier than for a dynamically changing one. It is also much easier to verify that the routing actually performs as desired.

Adaptation to Change. In a real network, of course, the determination of the output data outlined above is not a one-time calculation. As we have pointed out, the routing algorithm must run continuously, and its input data may change at any time. This leads us to the next requirement, that the algorithm be quickly adaptive to changes in network topology and traffic patterns. This point is of central importance, since routing for fixed inputs is a trivial problem by comparison. The computation to determine whether any paths exist between a pair of nodes must be sensitive to changes in the configuration of the network. The circuits and nodes in the network may fail, isolating some nodes from others, or there may be an alternate path remaining to connect them. That is, traffic should be routed around broken lines and nodes as long as any path exists to the desired destination. This may even require that a packet be returned over a path it has already traveled. For instance, consider a packet going the short way around a circle from its source to its destination when a line in the path ahead breaks. The packet should turn around and go back the other way around the circle. Similarly, circuits and nodes may be added to the network, and an adaptive routing algorithm has the advantage that these changes may happen smoothly and without any human intervention in the operation of the network. The algorithm must also be sensitive to changing traffic levels and patterns, since these will affect the computation of the paths of low delay and high capacity.

<u>Adaptation to Change & Priority of Routing</u>. One important aspect of the requirement that a routing algorithm be adaptive to changing network conditions is that it implies a priority structure to the task processing in the node computer. Routing must always have higher priority than packet processing because it may be essential to change the routes being used to some new routes, especially at moments of high traffic load. If routing cannot be recomputed because of higher priority tasks, then the

network performance will degrade significantly as congestion sets in due to out-of-date routing information. In practice, this means that:

1. The input of routing messages must always be possible.
   a. Storage must be reserved for the messages (a reserve of the common free list is necessary since the node cannot know ahead of input time whether the next input will be a packet or a routing message).
   b. The input process must be able to run often so that messages are not lost.

2. The output of routing messages must always be possible.
   a. Storage must be reserved for the messages (this can be dedicated fixed table storage).
   b. The output process must be able to run often enough to send routing messages as required, and they should have higher priority than any other transmission.

3. The routing update computation must always be possible.
   a. Storage must be reserved for the routing update (this also can be dedicated tables).
   b. The updating process must be able to run periodically, either on a clock interrupt or called by the input and output processes which have been guaranteed to run frequently enough.

Adaptation to Change; Speed. Given that the routing algorithm is adaptive to changes in the network, there are further desirable characteristics we can list. It should adapt as quickly as possible. Once the pattern of traffic has changed in the network, the old paths may become suboptimal, and perhaps even counter-productive, interfering with other useful traffic. The nodes should decide quickly and in a harmonious, cooperative manner how to use the resources of the network. When the routing algorithm must adapt, it should do so smoothly, without oscillation, and without creating undue conflicts in other parts of the network.

Global Optimality. There are other goals in addition to the local choice of paths of low delay and high throughput. It is important that the routing algorithm meet certain global requirements as well. For instance, the algorithm should lead the nodes in the network to a global optimum in utilizing shared resources. It is not enough for an individual node to find a good path to another node, it must do so in the light of the needs of other nodes in the network. It is possible to think up routing algorithms which stabilize at several operating points in a given situation, and only one of them is the global best use of resources. A good routing algorithm routes high bandwidth traffic to achieve the maximum global flow. Suppose, as in Figure 2-17, it is desired to send 50 kilobits/second of traffic from node 1 to

Figure 2-17 Global Optimality

node 4 and at the same time it is desired to send 50 kilobits/second from node 6 to node 5. If the traffic from 6 to 5 as well as the traffic from 1 to 4 must pass over the link from 6 to 5, the global flow is 50 kilobits/second. If, however, either the traffic from 1 to 4 or the traffic from 6 to 5 were to go around the other way via the link from 2 to 3, the global flow would be 100 kilobits/second which is the maximum global flow under the desired traffic inputs. Of course, it is clearly best for the 1 to 4 traffic to go via the link from 2 to 3 since that also minimizes the global delay.

__Fairness.__ This suggests that the algorithm should be fair to competition for shared resources. While fairness can sometimes be quantified, it is often a subjective judgment, but it is vital that the routing algorithm operating at one node does not prevent another node from gaining a share of network services. Consider Figure 2-18, in which it is desired to send 1 unit of traffic from node 0 to node 0', from 1 to 1', from 2 to 2', and so on. Clearly the maximal global flow is attained under this desired loading by stopping all traffic from 0 to 0' since any traffic from 0 to 0' reduces the global flow. But this maximum flow assignment is unfair. Let f be the flow from 0 to 0' and g be the flow from i to i' for i from 1 to N. Then $f + g \leq 1$.

Figure 2-18 Fairness

TF = Total flow = f + N*g

Max TF [f=0, g=1] = N

Min TF [f=1, g=0] = 1

Fairness demands that 0 be able to get some traffic to 0', even if it decreases the global flow. It might be considered fair to give all nodes an equal share of the available bandwidth, or to allocate bandwidth to each node in proportion to demand or to available capacity. For instance, one assignment which is certainly fairer than f=0, g=1 is:

f=1/2, g=1/2; TF=(N+1)/2.

A different approach is:

f=1/N, g=(N-1)/N; TF=N-1+(1/N)

which is quite close to the maximum flow for large N. Each definition of "fair" has some merits and drawbacks, but the important point is that the routing algorithm should be designed to adhere to some fairness doctrine, or else some traffic will be locked out.

2.4.2.1.6 Routing Performance Measures

In this section, we will take up the question of how to evaluate the performance of a routing algorithm. We will use the four basic factors underlying the performance of the network as a whole in considering the routing algorithm.

Delay. As we indicated in the section above on network delays, the most important point about delay usually is that interactive traffic experience the minimum delay possible, although sometimes uniformity of delay is important. Apart from general considerations such as giving such traffic higher priority than bulk traffic, what effect can routing have on delay performance? One way to answer this question is to review the components of delay introduced above, and to note what action the routing algorithm can take with regard to each. Before giving this list, it should be noted that the actual values of the variables in question here may vary enormously from one network to another, and within a given network. Therefore, some of the considerations will certainly outweigh others.

1. Propagation delays. The routing algorithm may keep track of the speed-of-light delays in the network, which may vary significantly if there are some long lines, and certainly if there are satellite links. Although these delays are fixed and beyond the control of the algorithm, it can choose paths with the least propagation delay.

2. Transmission delays. The routing algorithm may also keep track of the bandwidth of the circuits in the network, to know the transmission delays that packets will experience. Again, these delays are not likely to be dynamic, but the routing computation can use fast lines and avoid slow lines where possible.

3. Nodal processing delays. Here delay is more likely to have a large dynamic range. Some nodes may have different traffic loads than others by several orders of magnitude, and the nodes themselves may have different capacities, so the input queueing delays for processing service may vary. If this component can be significant, the routing algorithm should monitor the values for processing delays at the nodes.

4. Queueing delay. The case of queueing delays on circuits is similar. There will most certainly be some lines which are loaded more heavily than others, and the queueing delays are inversely proportional to line bandwidth, so long queues on slow lines lead to very long delays. In both of these cases, not only can the routing algorithm avoid long delays by choosing alternate routes, but it is also the major determinant of processing and queueing delays. That is, the routing algorithm may be structured to sense the buildup of these delays and change the routes being used accordingly, explicitly to reduce the load on individual nodes or lines.

5. Retransmission delays. This case is somewhat similar as well. The routing algorithm should adapt to any sizable number of retransmissions on any circuit, since they affect many of the other variables. By reducing the effective bandwidth of the circuit, retransmissions increase the processing and queueing

delays experienced by all packets. In this sense, the transmission delay for a given circuit is not strictly constant. This is especially true for satellite links used in broadcast mode.

Throughput. The next topic we will examine is that of network throughput as related to routing. We have stated that the important goal here is that large data transfers, as opposed to interactive traffic, gain high throughput. A list similar to that for delay can be proposed; these are issues which may be important for the achievement of high throughput, depending on the kind of network considered:

1. Circuit bandwidth. The routing algorithm must ascertain the effective bandwidth of the circuits in the paths that it chooses for high throughput traffic. This includes such effects as the use of the line by other nodes, the deterioration due to retransmissions, and so on. It is clear that in this regard the routing program is capable of much more than a passive measurement role. Since it is routing which determines at each node how much traffic to send over a particular line, it is possible to regulate the flow over high throughput paths if desired. It is even possible, in order to provide as high a utilization rate as possible, to ensure that all traffic is long packets and messages, although sending all interactive traffic by other routes may penalize that traffic in terms of delays.

2. Node bandwidth. The same comments apply to node bandwidth.

3. Buffering. We will assume that routing and flow control are different operations and are unrelated. Thus, the routing program need not be concerned with the existence of buffering at any level in the network.

4. Multiple paths. We have already mentioned the necessity for load-splitting in certain network situations. It is likely to be much more relevant for high-throughput applications than for low-delay traffic, but the technique of using many paths to a destination is of fundamental importance.

<u>Cost.</u> We now turn to a consideration of the relationship between routing and network cost. There are 3 basic network resources which the routing algorithm utilizes, and it can take several actions to utilize them effectively:

1. Line bandwidth. A primary consideration here is that the routing algorithm elect the paths with the fewest intermediate nodes, thus minimizing the total line bandwidth utilization. This is an important argument, and the basis for several algorithms based on shortest path routing. We should note that this goal may be in direct conflict with the goals of low delay and high throughput, though there are often cases, particularly in networks with uniform node and line characteristics, when shortest path routing is an excellent policy.

2. Node bandwidth. A second cost factor, similar to the first, is node bandwidth. Again, the routing algorithm which chooses short paths over long ones is at an advantage here. In both cases, the routing program would do well to seek underutilized resources rather than concentrating traffic on a few paths.

3. Node storage. The final cost factor is node storage, a specific instance of the higher cost of concentrated traffic. The routing algorithm has the capability of keeping the network queues as short as possible if that is a specific objective. Avoiding the inefficient use of storage in overlong queues may also tend to defer problems of congestion, which also make the network efficiency suboptimal.

A different cost consideration is that of network connectivity. Here too, the routing algorithm is an important factor in determining cost, though in a more indirect fashion. The cost of connecting the network is related to how adaptive the routing procedures are in practice. For instance, some networks have been proposed with fixed routing matrices giving two alternate routes to each node. In this kind of a network, it is essential to provide enough connectivity so that nodes are seldom declared unreachable by the routing algorithm (this may happen even though a network path exists between them). Of course, this raises the cost of the network. A similar problem holds for routing algorithms which adapt slowly, or only with human intervention, or only with some given probability of accuracy, and so on. When we discuss area routing later in Section 4, it will be clear that the problem of rapid and accurate determination of reachability is an important and difficult problem in networks with hundreds of nodes or more.

__Reliability.__ The last performance measure we will discuss is network reliability. Routing can have several different kinds of effects on reliability. In terms of network use, the important measure is the fraction of messages which are undelivered due to failures in the communications subnetwork. The routing program has as its main function in the network the efficient and reliable delivery of messages to their destinations. Despite all kinds of component failures, from lines to nodes to Hosts, the routing algorithm should continue to deliver messages properly or report that they are undeliverable because the destination is unreachable or not functioning.

The parallel requirement concerning the reliability of network connectivity has been examined in the section above on network cost. The appropriate measure here is the fraction of the time that the routing algorithm is in error concerning the reachability of some node.

A different set of issues arises in the relationship between the routing program itself and network reliability. Given the central role of the routing process in any network, it is particularly important that routing never break down altogether, the way most systems, software and hardware, eventually do. The reason is clear: if the routing in the network is incorrect or nonfunctioning, the network is completely unusable. This means that a different measure of routing performance is the percentage of network unavailability that is due to routing algorithm failures. In summary, the routing program must also be considered as another module of network software, with some given level of reliability, which is more sensitive in terms of network reliability than most other modules, because of its global impact.

2.4.2.1.7 Routing Cost Measures

There are five basic costs involved in the continuous operation of a routing algorithm. All these factors act to reduce the effective capabilities of the nodes and lines in the network with regard to the processing of data packets. In other words, these costs are various kinds of network system overhead. First, there is the CPU utilization at each node in the network needed to perform the calculation of the new best routes to all destinations. Second, there is the delay at each node associated with the processing of the routing information, introducing delays in the processing of packets. Third, there is the storage needed at each node as a data base for the routing calculation, and for the exchange of routing information with other nodes. Fourth, there is the line bandwidth used for the exchange of routing information between nodes. Finally, there is the line delay caused by the fact that sometimes a routing message is being transmitted at a time when a data packet is queued. We will now consider each of these costs in more detail.

Line Bandwidth. The first cost factor to consider is the fraction of the available line bandwidth needed to exchange routing messages between nodes. Clearly, this bandwidth is a function of the size of the routing message and its frequency as previously stated:

$$BWC_r = B_r * F_r$$

Depending on the algorithm, one may choose also to make the bandwidth used for routing a fixed number of bits per second, regardless of available line bandwidth, or one may wish to keep the bandwidth used for routing below some acceptable overhead fraction of the line bandwidth. Thus, on slow lines, routing would be sent less often or in an abbreviated form. One may also choose different priority strategies for the transmission of routing. As we have pointed out, routing messages are very important, and should probably take precedence over most other traffic. However, it may be desirable to specify that some classes of transmission take priority over routing messages. In this way, the routing messages need not introduce added delays to special high-priority messages, though they still represent a reduction in the effective bandwidth of the communications circuits.

__Line Delay.__ Next we examine the added delays on circuits caused by the transmission of routing messages. Consider a single line with one routing message of length Sr sent with frequency Fr per second, and a very light data traffic load. Then the probability of a data packet having to wait is the ratio of the time duration in which routing is being sent to the routing period. For a given circuit bandwidth BWC, this probability of line delay is:

BWCr/BWC = Sr*Fr/BWC.

Given that a packet must wait, the average wait is half the maximum, or

ALD = Sr/(2*BWC).

The expected line delay is the product of the probability of line delay and the average line delay:

ELD = (BWCr/BWC)*ALD,

That is,

ELD = (Br*Br*Fr)/(2*BWC*BWC),

This means that the delay to packets due to routing messages increases as follows:

linearly with increasing routing frequency,

quadratically with increasing routing message length,

quadratically with decreasing line bandwidth.

The reason that the analysis presented above is valid only for light traffic loads is that a queueing phenomenon causes total network delay to increase nonlinearly with load. Therefore, the added traffic due to the presence of routing messages has a correspondingly greater effect in terms of delay at high loads.

<u>Nodal Bandwidth</u>. The calculation of the best routes to all destinations represents a steady, periodic demand for CPU processing time. One can view the computation as a certain percentage of overhead in the CPU bandwidth available for message processing. We have called this factor BWPr, the fraction of the bandwidth of the processor used for routing. In general, it has two components, one based on the time taken to process routing messages on each line, and the other a simple periodic component:

BWPr = NLN*Pr*Fr + Pp

where NLN is the number of lines per node, Pr is the processor time for routing messages, Fr is the frequency of routing messages, and Pp is the fractional overhead of processor time for periodic processing of routing. For instance, if NLN=2, Fr=5 messages/sec, and Pp=1%, then BWPr=3%. If BWPr becomes too large, the processor becomes much less cost-effective in its primary role as message processor. Therefore, a major cost consideration in evaluating a routing algorithm is the number of CPU cycles per second it requires in each node of the network.

Nodal Delay. Along with the reduction in effective nodal processing power comes the effect of delays in the processing of packets while the routing computation is proceeding, given that it takes priority over data processing. Suppose one routing message is received from each of NLN adjacent nodes with frequency Fr messages per second. Then, given the time Pr and the fractional time Pp above, there are three cases to consider for nodal delay:

1. NLN*Pr*Fr >> Pp message processing dominates
2. NLN*Pr*Fr << Pp periodic processing dominates
3. NLN*Pr*Fr = Pp factors have equal magnitude

If we assume that NLN*Pr*Fr < 1 (=total processor bandwidth), and that packet processing time is infinitesimal, we can analyze each of these cases:

1. The probability that a packet has to wait is

NLN*Fr*Pr

and if it must wait for only one message, it experiences an average wait of

Pr/2.

This means that the expected delay is

NLN*Fr*Pr*Pr/2

which is a lower bound on the actual value. It can be shown with a more detailed analysis that this is the expected value for delay under the condition that 1/(Fr*Pr) >> NLN, which means that there are many more time periods in which the processor is able to process routing inputs than there are adjacent nodes to send the routing. In practice, this is the only assumption that makes sense, since otherwise BWPr becomes close to unity, and there is no bandwidth available for data traffic.

2. This case can be analyzed as line delay was analyzed above, based on the length and frequency of the periodic processing, Pp.

3. This case can be analyzed on the basis of the first two cases, summing the effects of each of the terms.

Nodal Storage. The final cost factor we will discuss is the storage required at each node to maintain the routing information. This cost may vary greatly among various algorithms, depending on how much information is needed in the routing computation, and also on the method of exchanging routing messages. The node must save the incoming routing information in some fashion, then the routing computation may generate other, new information, and the transmission of routing messages to the adjacent nodes may call for still more storage for data. In a small communications processor, memory is dedicated to a fairly small program, and message buffers. Any storage used for the routing calculation must be taken from the message buffer pool, either once and for all at design time, or dynamically. Therefore, the storage requirements of the routing algorithm represent still another overhead factor.

2.4.2.2 Original Implementation

In this section we describe the routing algorithm originally installed in the ARPANET, and examine it from the point of view of the performance and cost criteria outlined above.

The original ARPANET algorithm can be summarized as follows. This algorithm directs each packet to its destination along a path for which the total estimated transit time is smallest. This path is not determined in advance. Instead, each IMP individually decides which line to use in transmitting a packet addressed to another destination. This selection is made by a simple table lookup procedure. For each possible destination, an entry in the routing table designates the appropriate next line in the path.

Each IMP maintains a network delay table which gives an estimate of the delay it expects a packet to encounter in reaching every possible destination over each of its output lines. This table and other tables mentioned below are shown in Figure 2-19 as kept by IMP 2, for example. Thus, the delay from IMP 2 to IMP 5 using line 3 is found to be 4 in the Network Delay Table. Periodically, every 2/3 of a second, the IMP selects the minimum delay to each destination and puts it in the minimum delay table. It also notes the line giving the minimum delay and keeps the number of the line in a table for use in routing packets. Also every 2/3 of a second, the IMP passes its minimum delay table to each of its immediate neighbors, that is, it sends the minimum delay table out each of its phone lines. Of course, before the minimum delay table is transmitted to the neighboring IMPs, the IMP sets the minimum delay to itself to zero.

Since all of the neighbors of an IMP are also sending out their minimum delay table every 2/3 second, with their own entry set to zero, an IMP receives a minimum delay table from each of its neighbors every 2/3 second. These tables are read in over the

Figure 2-19 ARPANET Routing Algorithm

rows of the delay table as they arrive. The row to be written over is the row corresponding to the phone line that the arriving minimum delay table came in over. After all the neighbors' estimates have arrived, the IMP adds the delay saved by the IMP itself to the neighbors' estimates. This is done by adding the IMP delay table, i.e., the contribution of this IMP to the total delay to each destination, to each column of the delay table. Thus the IMP has an estimate of the total delay to each destination over the best path to that destination.

In parallel with this computation, the IMPs also compute and propagate shortest path information in a similar fashion. This information is used only in the determination of connectivity. An upper limit of the number of lines in the longest path in the network is used as the cut-off for disconnected or nonexistent nodes.

Now let us consider the performance of this algorithm. First of all, it explicitly determines the connectivity of the network, since all IMPs are continuously exchanging the length of the shortest path from each IMP to each other IMP. Information travels at roughly 2/3 of a second per line, so that changes in topology are recognized by the whole network in a matter of a few seconds. This figure is probably acceptable if one assumes that the network connectivity does not change too often. Second, the algorithm also explicitly calculates the path of least delay. However, here the approximations due to the low frequency of routing update mean that the delay for traffic at one instant is a function of the traffic of several seconds before. This could potentially lead to oscillations and poor line utilization. The ARPANET algorithm attempts to head off this class of problems by biasing delay heavily toward the shortest path. That is, delay is measured by the number of packets on an output queue, plus a fixed increment, so that even an empty queue represents additional delay.

The algorithm has several faults, some of which are relatively simple to cure, and others which are more fundamental in nature. The strong bias in the algorithm towards the shortest path is basically a good idea, and leads to stable flows near optimum values. However, the bias makes the algorithm somewhat insensitive to changes in traffic patterns, so that global optimization of delay and throughput is not likely as network loading increases. A second fault is that the algorithm only maintains one route per destination, updated every 2/3 second. This means that no load-splitting is possible, at least not on a short term basis. The algorithm could be modified to use one of several routes to each destination, with weights assigned to each. Further, the algorithm might maintain additional data on the loading of the various paths, to facilitate more rapid adaptation to changes in traffic. This change, combined with the expansion to several routes, might also lead to smoother and more uniform adaptation.

The original ARPANET routing algorithm was quite a good design in many respects. Perhaps its strongest point is that it is simple. The IMP does not have to know the topology of the network, or even the identity of its neighbors. When IMPs and lines go down, the algorithm functions as usual, and the new routing information propagates through the network by a process of exchanges between neighbors. Therefore, the algorithm scores quite well in reliability. Although there are no explicit controls to ensure fairness to competition, the algorithm does relatively well in this category as well.

Finally, the algorithm is not a costly one in terms of the measures discussed above. The program in the IMP picks the minimum delay and hop counts from the routing messages received from each neighbor, for all destinations. Thus, the calculation is proportional to the number of IMPs in the network, and the number of lines connected to each IMP. The routing computation takes up about 5% of the CPU bandwidth of the IMP. The delay and hop information is packed into a single 16-bit word, so that the routing message sent out each line consists of 64 words, one for each IMP in the network, plus some header information. This amounts to less than 2% of the bandwidth of a 50 Kbs line. At these low bandwidth rates, added node delays and line delays are not appreciable. In addition to the storage required for sending the routing message out each line (one copy of the message is shared by all lines), the IMP reserves storage for receiving routing messages from each of its lines. These tables, together with its own directory of the best line to each destination, amount to about 3% of the core storage on an IMP. In summary, the IMP routing algorithm is a simple, inexpensive algorithm which performs well in steady state, and in reacting to small changes in traffic.

2.4.2.3 Subsequent Modifications

One problem with distributed algorithms is that they are slow in adapting to some kinds of change; in particular, the original ARPANET algorithm reacts quickly to good news, and slowly to bad news. If the number of hops to a given node decreases, the nodes soon all agree on the new, lower, number. If the hop count increases, the nodes will not believe the reports of higher counts while they still have neighbors with the old, lower values. Another point is that there is no way for a node to know ahead of time what the next-best or fallback path will be in the event of a failure, or indeed if one exists. In fact, there must be some finite time, the network response time, between the occurrence of a change in the network and the adaptation of the routing algorithm to that change. This time depends on the size and shape of the network.

In 1973, the decision was made that the routing algorithm should continue to use the best route to a given destination, both for updating and forwarding, for some time period after it gets worse. That is, the algorithm should report to the adjacent nodes the current value of the previous best route and use it for routing packets for a given time interval. This technique was termed "hold down". One purpose of hold down is to distinguish between changes in the network topology and traffic that necessitate changing the choice of the best route, and those changes which merely affect the characteristics of the route, like hop count, delay, and throughput. In the case when the identity of the path remains the same, the mechanism of hold down provides an instantaneous adaptation to the changes in the characteristics of the path; certainly, this is optimal. When the identity of the path must change, the time to adapt is equal to the absolute minimum of one network response time, while the other nodes have a chance to react to the worsening of the best path and to decide on the next best path.

The routing algorithm is extremely important to network reliability, since if it malfunctions the network is useless.

Further, a distributed routing algorithm has the property that all the nodes must be performing the routing computation correctly for the algorithm to be reliable. A local failure can have global consequences; e.g., one node announcing that it is the best path to all nodes. Routing messages between nodes must have checksums and must be discarded if a checksum error is detected. All routing programs must be checksummed before every execution to verify that the code about to be run is correct. The checksum of the program should include the preliminary checksum computation itself, the routing program, any constants referenced, and anything else which could affect its successful execution. Any time a checksum error is detected in a node, the node should immediately be stopped from participating in the routing computation until it is restored to correct operation again.

The new concept introduced in the ARPANET, known as "hold down", facilitates the process of adapting quickly, uniformly, and without oscillation. This technique is of central significance to routing design, and many aspects of its use are examined and explained. These changes make it work considerably faster and also more efficiently. The previous approach, based on path length, required routing messages to be exchanged which contained hop counts to all nodes. The need for this mechanism is eliminated through the use of hold down. Finally, later sections consider the application of this technique to the problems of traffic assignment.

The original ARPANET routing technique has a flaw that it shares with all algorithms built on a repeated distributed minimization or maximization. The algorithm operates in such a way that the hop count to a given node increases smoothly at greater distances from that node. No node ever has a hop count to a given destination which differs by more than one from the minimum count held by any of its neighbors. What this means in terms of adaptation is that

<u>the reachability algorithm reacts very quickly to good news, and very slowly to bad news.</u>

If the number of hops to a given node decreases, the nodes soon all agree on the new, lower, number. If the hop count increases, the nodes will not believe the reports of higher counts while they still have neighbors with the old, lower values. The simple case of a line of nodes shows how this happens. The four cases in Figure 2-20 below show how the hop counts change when the actual number of hops increases and decreases, for the best and worse cases of sequencing among the nodes. Clearly, the order in which the nodes compute and exchange routing is critical here, and we will assume that there is some definite order in which the nodes periodically perform the calculation and updating. We assume that each node in the sequence finishes the calculation, sends out the new tables, and the information arrives at the nodes adjacent to that node, all before the next node in the sequence begins its computation.

```
0----1----2----3----4 0----1----2----3----4

ON MAX MAX MAX MAX ON MAX MAX MAX MAX

UP 1 2 3 4 UP 1 MAX MAX MAX

1 2 MAX MAX

1 2 3 MAX

1 2 3 4
```

a. Left-to-right sequence, b. Right-to-left sequence,

best case, node 0 comes up worst case, node 0 comes up

```
0----1----2----3----4 0----1----2----3----4

UP 1 2 3 4 UP 1 2 3 4

DN 3 4 5 6 DN 3 2 3 4

5 6 7 8 5 4 3 4

 ... 7 6 5 4

9 8 7 6

MAX MAX MAX MAX 11 10 9 8

 ...

MAX MAX MAX MAX
```

c. Left-to-right sequence, d. Right-to-left sequence,

best case, node 0 goes down worst case, node 0 goes down

Figure 2-20 How the Reachability Algorithm Adapts

In the light of this analysis, we can remark on the choice of the hop count mechanism for the ARPANET reachability algorithm. It could have been built around minimum delay, which was being computed anyway for route selection, instead of around minimum hop. However, the value of delay corresponding to unreachable nodes is much higher than MAX, since hop counts increase by a single unit for each line, and delay has a large dynamic range (4 to 24 in the original ARPANET implementation, with all line capacities equal). Thus, using a high value of delay to detect unreachable nodes would take proportionately more time, given the linear rise time characteristic of the process. This time would have been so long, on the order of minutes, that a delay-based reachability algorithm was not practical.

One can investigate the function served by rise and fall times (times required to adapt to or change) in a routing algorithm, in order to determine whether these delays are necessary, and if bounds can be placed on them. To begin with, it is easy to understand why a distributed algorithm has a zero fall time. If a node receives information about a better path, it will use it right away. As we shall see, however, this information may not be accurate or long-lived, and therefore acting on the information may be a poor decision. On the other hand, it is also easy to see why the distributed approach leads to a long rise time. When it discovers that the current best path has grown more costly to use, it has no second-best path readily available. This is an instance of solving a global problem on the basis of local estimates. The node must choose the best path to a destination, and the only direct information it has is about the lines to which it is connected. The routing information, which is about paths, is always slightly out of date and thus inaccurate. Further, the simple distributed routing algorithm only sends routing data about the best path, not all possible paths. Even if such information were available to the routing decision process, it still requires the cooperation of the whole network, and thus some time and many routing messages, to determine the identity of the next-best path after the best path degrades. We next give some examples to illustrate this process.

Consider the networks shown in Figure 2-21. In all these networks, we are concerned with a path from node 1 to node 3. Such a path exists, two hops long, in all three networks shown, when all components are up. In Figure 2-21a, a line has gone down, and no path exists from 1 to 3. The same is true in Figure 2-21b, where node 3 has gone down. In both these cases, the routing information held in the cycle of nodes 1,2, and 4 will circulate around, with the hop counts to node 3 slowly counting up. Initially node 2 is 1 hop away from node 3, and nodes 1 and 4 are 2 hops away. After the component fails, node 2 thinks there is another path through node 1 or node 4, and sets its hop count to 3. Likewise, node 1 counts to 4. Depending on the sequence in which the nodes send routing, the nodes will begin to count until the hop count reaches MAX.

The other cases shown in Figure 2-21 provide a possible interpretation of this counting sequence. In the network of Figure 2-21c, the line from 2 to 3 is down, but an alternative path of 3 hops exists. This is not true for the same network in Figure 2-21d, where node 3 is down. Similarly, in Figure 2-21e, a path of 4 hops exists and is the shortest path, while in Figure 2-21f, there is no path to node 3. Obviously, there are an infinite number of such examples, each constructed upon the last. When a shortest path of H hops is disrupted, the routing algorithm which is operating in the network at large must investigate for the existence of a new path. Conceptually, this can be done by looking for a path of length H, and then (if unsuccessful) for one of length H+1, H+2, H+3, ..., MAX-2, MAX-1. At that point the algorithm can conclude that no path exists to the given node. As indicated by the examples in Figure 2-21, this search must involve all the nodes and lines in the network. There is no way for a node to know ahead of time what the next-best or fall-back path will be in the event of a failure, or indeed, if one exists. This line of reasoning indicates that

There must be some finite time, the network response time, between when a change in the network occurs and when the

Figure 2-21 The Choice of a Second-Best Path

<u>routing algorithm adapts to the change." Further, this time depends on the size and shape of the network.</u>

One can interpret the operation of the reachability algorithm above as searching the network for successively longer paths after the best path fails, until it gives up at paths of length MAX. It is important to note that while conducting this search for a path of some length, the routing algorithm must continue to believe that the node is reachable, and thus must route traffic to it.

Now that we have discussed what makes the simple distributed reachability algorithm slow, and why these effects are common to a wide class of distributed algorithms, we present a partial solution. It is clearly desirable to reduce the rise time of the routing algorithm if that can be done without impairing its effectiveness. One way to think about this goal is to treat it as collapsing the one-by-one search for paths of increasing length into a single search for paths of any length, and choosing the best one. This high-level idea still leaves open the problem of what to use as the best route while the search is in progress, while the old best route is no longer good but no replacement has been found. One possibility, instead of oscillating between various routes as the search proceeds, is to continue to use the earlier best path until a new, better path is found. Now consider this technique for handling the transition period. Is it not just the technique we need to speed the search itself? Taking the value of the old best route as still the best has exactly the desired effect in the search. If there are any routes with better values they will be found in the time it takes the network to perform a search, or what we termed above the lag time. That is, it takes only one step to find a better route if one exists. In the case that the old best route now is unusable (has a length of MAX), the node waits only the specified time period, the lag time, for a better route to be found. If a path does not turn up in that time, it can be assumed that one does not exist. This is the partial solution we want, which we will call hold down:

> <u>The routing algorithm should continue to use the best route to a given destination, both for updating and forwarding, for some time period after it gets worse.</u>

That is, the algorithm would report the current value of the previous best route to its neighbors and use it for routing packets, for a given time interval. This idea was first suggested by W. R. Crowther of BBN (internal memo, 1971).

Another way to look at this is to distinguish between changes in the network topology and traffic that necessitate changing the choice of the best route from those changes which merely affect the characteristics of the route, like hop count, delay, and throughput. The latter may represent a much larger class than the former, particularly in large networks. While the actual length of the path may change, the general direction in which packets should be routed is not likely to change, given that nodes are widely distributed and have only a few lines. This is what motivates believing the best line for a few ticks longer. The identity of the best path probably does not change, though its characteristics may.

1. In the case when the identity remains the same, the mechanism of hold down provides a stepwise adaptation to the changes in the characteristics of the path, that is, a rise and fall time of zero. Certainly, this is optimal.

2. When the identity of the path must change, the rise and fall times are both equal to one lag time. This is optimal for any algorithm within the practical limits of propagation times discussed above.

Yet another interpretation of this technique is suggested by the network configurations of a chain of nodes, each having 2 lines. In this network it is clear that when the first node, call it 0, goes down an intermediate node, say node 2, thinks it has an alternate path, longer than its original path, via its other line. Of course this is not true, because the hop count at node 3 is based on the hop count at node 2, and has no meaning if the best path at node 2 is not longer usable. Clearly, this situation holds in general because of the way that the distributed computation smooths the routing information over the network.

Since no node holds a hop count which differs by more than one from that held by any of the adjacent nodes, there will certainly be apparent alternate paths out all its other lines. The technique of holding the original best path as still the best, with its new value, has the effect of _purging_ all those sites depending on that path as a subpath for their routing. Within one lag time, no nodes in the network will be using the old information. If the other lines at the node are reporting better paths than the original one, the node can then believe that they are new paths, independent of the original path. Thus a node enters a transitional state, which we call hold down, when it needs to acquire accurate information about a new best route.

There are two points which should be clarified about when hold down should be invoked.

> _The node should enter hold down when the value of the best line has gotten worse, but not when some other line becomes better than the original best line._

The reason for this distinction becomes clear when one considers the motivation for hold down laid out above. The purpose of the hold down mechanism is restricted to forcing stepwise adaptation to changes for the worse in routing information. The algorithm should be allowed to adapt freely to improvements in the routing information being received on any line, the previous best line or any other. A second clarification to the mechanism is as follows:

> *Whenever a line goes down (or the node to which it is connected goes down) which is the best route to a destination, the node should enter hold down.*

Clearly, there must be a procedure to acquire a new route, and the whole case for hold down is based on the observation that the acquisition of the new route cannot proceed reliably until all the old information has been purged. There are two practical considerations which should be mentioned. First, the hold down time should be run in parallel with the line-going-down time rather than in serial, after the line has gone down. That is, if hold down runs for the last n seconds of the line-going-down time, then a new route should be established (or the absence of any route established) by the time that the line is declared unusable. A second point to make is that this hold down mechanism can be invoked quite conveniently by introducing a dummy input routing message from the dying line, containing entries for all destinations set to MAX. This has the effect of causing the algorithm to enter hold down for just those nodes for which the dying line was the best route.

Now we turn to the question of the duration of the hold down mechanism. For this purpose there must be a timer corresponding to the best path to each destination, along with the identity of the best path, RH in the algorithms above, and the value of the variable being optimized, D above when we considered distance. The first point to make about this timer is:

If the best line gets worse and then gets better again, the hold down "timer" must be allowed to run to completion.

A simple example suffices to illustrate that resetting the timer if the information from the best line improves may cause the original degradation in the line not to be communicated. Consider the following network, with delay of unity in both directions on both lines:

1==========2==========3

Figure 2-22 The Hold Down Timer Must Complete

Suppose the delay from node 2 to node 1 increases to 5, then decreases to 4, while other delays remain constant at 1. The delay to node 1 seen at nodes 2 and 3 is as follows:

node 2 node 3

1 2

5 6 begin hold down

4 5

...

4 5 end hold down

Now suppose we allow the improvement in delay from node 2 to node 1 to reset the hold down timer. It may happen that node 3 does not hear about the increase in delay to 5 until after it sends routing to node 2. This routing then may arrive at node 2 just after the improvement in delay from node 2 to node 1. When the improvement is detected, hold down is terminated, and the information from node 3, out of date and incorrect, is believed. The delays can be shown as:

node 2 node 3

1 2

5 2 begin hold down at node 2

4 2 end hold down at node 2

3 2 incorrect information from node 3

adopted by node 2

This example suggests two more observations. The first concerns avoiding the problem, described above, of introducing spurious routing information into the network.

> <u>The hold down time must be long enough for the routing information to propagate from the node in question to all adjacent nodes and back again.</u>

The key function of hold down is to purge the old routing information from any neighboring nodes so that they do not echo it back and thus impede the rapid adaptation to new information. For this end it is not necessary to wait for the entire network to respond to the change in routing, but only for the immediately adjacent nodes. Actually, this statement may be too strong in some cases. For example, in the network in Figure 2-22 the best routes (based on delay, say) to node 1 are indicated by arrows. In the event that the line from node 3 to node 2 gets worse, node

3 will enter hold down. Then node 6 will soon initiate a hold down because its best line has become worse. Node 4, on the other hand, is not affected since its best route is not by way of node 3. It will not begin to hold down until nodes 3, 6, and 5 have all begun their hold down timers. Thus, node 3 must keep hold down on for long enough to allow the new information to propagate around the cycle. Otherwise, its hold down time will expire and node 4 will report old information that is no longer valid. The conclusion is

<u>The hold down time must be long enough for the routing information to propagate through several hops.</u>

The exact number of hops depends on the likelihood of a path of several hops being superior to one of fewer hops. With minimum hop routing, hold down need last only long enough for routing information to propagate one hop. If the objective function is delay or throughput, it must in general be longer. The penalty for a hold down timer which is too short is not severe; routing will oscillate when the recently-acquired alternate route is found to be no better after all. No permanent incorrect information is introduced.

A second observation related to the network shown in Figure 2-23 is that if the line from node 2 to node 3 carries routing updates less often than the other line, its hold down time must be longer:

Figure 2-23

The hold down time must be inversely proportional to the frequency of routing propagation on the lines. Thus, if routing is sent less often on slow lines, or if there is a long delay (e.g., satellite links), the hold down time must be longer.

Ideally, the node has separate hold down timers for each line, running at independent rates, holding down the transmissions. In practice, it may be easier to adopt a set of hold down counters for each line speed present at that node.

The first point to make about reachability is one that we overlooked previously, but which assumes more importance now. Consider the problem of having the reachability algorithm adapt after the network has been isolated into two or more components, as shown in Figure 2-24. Suppose the line between nodes 1 and 2, which is the only connection between Networks 1 and 2, has been down and now comes back up. Nodes 1 and 2 exchange routing information. Node 2 reports that all of Network 2 is up. Node 1 finds out about this, but it cannot initiate a message to a node in Network 2 because none of those nodes have yet been informed that node 1 is up. Data packets can travel faster than routing information. If node 1 were to send a message to one of the nodes in Network 2, the returning RFNM and any replies from the Host would be lost because the destination node would think node 1 was still unreachable. Thus when node 1 sees all the nodes in Network 2 come up for the first time, it must mark them as in a special coming-up state. While a node has a given destination node marked as coming up, it can accept store-and-forward traffic for that node, and can propagate alive routing about that node, but it cannot initiate any Host traffic for it. The timer must be long enough for routing to propagate through the whole network (since Network 1 may be a single node), though it can be cleared as soon as a message is received from a node in the coming-up state.

While the need for a coming up delay is common to any reachability scheme, there is a similar delay which is necessary only because of the nature of hold down. A going down delay is also necessary, as can be demonstrated by the large ring shown in Figure 2-25. Suppose the shortest paths to node 1 from all other nodes are indicated by the arrows and the line from node 1 to node 2 goes down. Then node 2 begins a hold down, reporting a value of MAX hops to node 1 in its routing message to node 3. Then node 3 in turn begins a hold down, and so on. Eventually, node X holds down but node Y keeps its old route. By this time, the hold down at nodes 2 and 3 may be over if the loop is large. They must now wait for the information about the other route to propagate from node Y to node X, and eventually to nodes 3 and 2. Thus the going down timer for a node must also be long enough for the network to respond to a change in the routing. Periodically, in concert with the running of the shortest path algorithm, the node makes the

Figure 2-24 The Need for a Coming Up Delay per Node

Figure 2-25 The Need for a Going Down Delay per Node determination as to whether all destinations are reachable on the basis of whether HOP(I) is less than MAX or not. And as we have seen, the transitions in state must be damped according to specified time constants. As shown in Figure 2-26, each destination node is in one of 4 states as seen by each of the other nodes in the network. The constants NCU, coming up time, and NGD, going down time, are topology-dependent. Here "reachable" refers to whether HOP(I)=MAX or not, and "get message" means that a message is received from the destination node.

The state diagram for the reachability of a given node resembles closely the state diagram indicating whether a given line between nodes is usable or not. The logic shown in Figure 2-27 is from the line state program used in the ARPANET, but the principles are quite general. The IMPs decide whether a line is up or down on the basis of "Hello" and "I Heard You" (IHY) messages. There is a quality control placed on the line by the two constants LGD and LCU. If the IMP misses LGD I Heard You's in a row, it declares the line unusable and waits for LHD (Line Held Dead) ticks so that the other IMP learns that the line has gone down. Then the IMP must receive LCU IHY's before the line can come up. The line is reset if LR ticks go by without success.

Terminology. We begin by explaining the terminology used below to describe the ARPANET routing algorithm. The actual IMP program is written in assembly language and does not use the terms

Figure 2-26 Node State Diagram

Figure 2-27 Line State Diagram

employed here; we have re-written it below in a PL/I-like language. The algorithm performs two basic functions: determining whether each other IMP is reachable, and calculating the path of least delay to those IMPs which are reachable. The program maintains a number of tables indexed by IMP; these all have one entry for each of the NN IMPs in the network. All these tables carry checksums which are a function of the NN entries, and which the program periodically verifies in order to minimize and localize the effects of any hardware or software failures. ROUTE is the directory giving the output line corresponding to the shortest hop path to all IMPS; DROUTE is the analogous minimum delay directory. HOP and DEL are the tables containing the number of hops and estimated delay on these paths. The maximum values in these tables are denoted by HMAX and DMAX respectively. HOPIN and DELIN are the two parallel tables which together constitute the routing message received at an IMP. HOPOUT and DELOUT are the tables constructed by an IMP to form its next routing message to be transmitted, and to be used as the new HOP and DEL tables. The algorithm uses a technique called hold down to ensure that an IMP will not base its routing decisions on old information sent to it by its neighbors. The technique consists of using a timer, HOLDT for the best hop path and DHOLDT for the minimum delay path, to wait a short interval before deciding to change to a new path. This interval is long enough for all the neighbors of the IMP to learn about the new information, so that the IMP can make its decision accurately, rather than on the basis of "echoes" of its old information. This interval is set to HTMAX. In order to determine if hold down is necessary on the minimum delay path, the program keeps two counters, DELOLD and DELTHS, which accumulate the increase in delay in the previous update period, and in the current update period respectively. Finally, the program inserts a transition delay so that all IMPs in the network learn about the transition before it becomes final. GODOWN is a table of timers which are set to GDMAX when an IMP begins to go down; COMEUP is a similar set of timers set to CUMAX when an IMP is first detected as reachable.

Routing Message Input Processing. When a routing update message is received, the input module first verifies the checksum on the message. If it is incorrect, the program reports an inter-IMP failure and ignores the message. Otherwise, it puts the message on a queue for a lower-priority routine. This routine then takes the following actions:

1. Take the routing message off the input queue.

2. Verify the checksum on the following code before executing it.

3. Abort if the checksum is incorrect.

4. Ignore the message if the input line is down, or if it is looped, or if the serial number on the routing message is the same as that on the last message received, or if the format compatibility number does not match the number used by the receiver.

5. Initialize the checksum on the output message to be generated, put in the sender's node number, compatibility number, and the incremented serial number.

6. Perform the basic routing computation as follows:

```
For I<-1 to NN do;
 if I=SELF then HOPOUT(I),DELOUT(I),ROUTE(I),DROUTE(I)<-0;
   else do;
     if ROUTE(I)=RMAX
       then /* I has been declared unreachable */
         if HOPIN(I)≥HMAX
           then /* I is still not reachable via J */
             do; DELOUT(J)<-DMAX; HOPOUT(I)<-HMAX;
             end;
           else /* a new path to I has been discovered */
             do; COMEUP(I)<-CUMAX; /* Set Come-up timer */
                 ROUTE(I)<-J;
                 HOPOUT(I)<-HOPIN(I)+1;
             end;
       else /* I has not been declared unreachable */

         if ROUTE(I)=J
           then /* Current min-hop path is via J */
             do; if HOPIN(I)+1>HOP(I)
                   then /* Best path got worse, hold it down */
                     HOLDT(I)<-HTMAX;
                 if HOPIN(I)+1<HMAX
                   then /* Best path still exists */
                     do; GODOWN(I)<-0;
                         /* Make sure godown timer is off */
                         HOPOUT(I)<-HOPIN(I)+1;
                     end;
```

```
                else /* I is no longer reachable through J */
                  do; if GODOWN(I)=0 then GODOWN(I)<=GDMAX;
                      HOPOUT(I)<=HMAX;
                  end;
          end;
        else /* current min-hop path is not through J */
          if HOLDT(I) not = 0
            then /* I is being held down, make no changes */
              HOPOUT (I)<= HOP(I);
            else if HOPIN(I)+1>HOP(I)
              then /* No reason to switch paths */
                HOPOUT(I)<=HOP(I);
              else /* New min-hop path is via J */
                do; ROUTE(I)<=J; GODOWN(I)<=0;
                    HOPOUT(I)<=HOPIN(I)+1;
                end;

      if DROUTE(I)=J
        then /* current least-delay path is through J */
          do; D <=DELIN(I)+DELLOC-DEL(I);
              /* D=change in delay since last update */
              DELOLD(I)<=MAX (DELOLD(I)+D, 0);
              DELTHS(I)<=MAX (DELTHS(I)+D, 0);
              if DELOLD(I)>8 or DELTHS(I)>8
                then /* delay has gotten worse - hold down */
                  DHOLDT(I)<=HTMAX;
              DELOUT(I)<=DELIN(I)+DELLOC;
          end;
        else /* current least delay path is not through J */
          if DHOLDT(I) not = 0
            then /* I is being held down, make no changes */
              DELOUT(I)<=DEL(I);
            else if DELIN(I)+DELLOC>DEL(I)
              then /* No reason to switch paths */
                DELOUT(I)<=DEL(I);
              else /* New least delay path is via J */
                do; DROUTE(I)<=J;
                    DELOLD(I), DELTHS(I)<=0;
                    DELOUT(I)<=DELIN(I)+DELLOC;
                end;
    end;
  /* update checksums on any tables with changed entries */
end;
```

7. Finalize the routing message checksum and put it in the message buffer.

8. If the current routing message buffer is not being transmitted on any line, circulate the triple buffer system as follows:

   1. assign the new message to the output buffer;
   2. assign the output buffer as the free buffer area;
   3. assign the free buffer area as the buffer in which the next message will be built.

Routing Message Output Processing. When the program receives the completion interrupt on one of the IMP's lines, it first checks to see if the send-routing flag is set. Routing transmissions take priority over all others. If the flag is set, the program takes the following steps:

1. Verify the checksum on the following code before executing it.

2. Abort if the checksum is incorrect.

3. Locate the next routing message buffer from which to transmit, and verify that the message has a correct checksum.

4. Report an intra-IMP failure if the checksum is in error, and resynchronize the line protocol before continuing.

5. Otherwise, initiate the output and dismiss the interrupt.

Periodic Routing Processing. Every 640 milliseconds the program takes the follow steps:

1. Verify the checksum on the following code before executing it.

2. Abort if the checksum is incorrect.

```
For I<-1 to NN do;
  if HOLDT(I)>0
    then /* decrement hold down timer for hops: 1.28-1.92 secs */
    HOLDT(I)<-HOLDT(I)-1;
  if DHOLDT(I)>0
    then /* decrement hold down timer for delay: 1.28-1.92 secs */
    DHOLDT(I)<-DHOLDT(I)-1;
  DELOLD(I)<-DELTHS(I); DELTHS(I)<-0;
  if GODOWN(I)>0
    then /* decrement timer for IMPs going down: 7.7-11.5 secs */
    do; GODOWN(I)<-GODOWN(I)-1;
        if GODOWN(I)=0
          then /* IMP I just died */
            do; ROUTE(I)<-RMAX; DROUTE(I)<-RMAX;
            end;
    end;
  if COMEUP(I)>0
    then /* decrement timer for IMPs coming up: 34.6-40.3 secs */
    COMEUP(I)<-COMEUP(I)-1;
  /* update checksums on any tables with changed entries */
end;
```

3. Verify the checksums on the various tables: ROUTE, DROUTE, HOLDT, DHOLDT, GODOWN, COMEUP.

Every 25 milliseconds, the program takes the following steps:

1. Set the send-routing flag for the appropriate lines according to their line speed and line utilization. The rule used is that the overhead due to routing messages should range from 3% of the line bandwidth on busy lines to 15% of the line bandwidth on idle lines.

2. Attempt to circulate the routing message buffers as in step 8 in the section above on Routing Message Input Processing.

2.4.3 Source IMP -- Destination IMP

2.4.3.1 Basic Concepts

There has been considerable controversy during the last several years over whether or not a store-and-forward subnetwork of nodes should concern itself with end-to-end transmission procedures. Many workers feel that the subnetwork should be close to a pure packet carrier with little concern for maintaining message order, for high levels of correct message delivery, for message buffering in the subnetwork, etc. Other workers, including the ARPANET designers have felt that the subnetwork should take responsibility for many of the end-to-end message processing procedures. Of course, there are some workers who hold to positions in between. However, many design issues remain constant whether these functions are performed at host level or subnetwork level, and we discuss these constants in this section.

Buffering and Pipelining. As noted earlier, any practical network must allow multiple messages simultaneously in transit between the source and the destination, to achieve high throughput. If, for example, one message of 2000 bits is allowed to be outstanding between the source and destination at a time, and the normal network transit for the message including destination-to-source acknowledgment is 100 milliseconds, then the throughput rate that can be sustained is 20,000 bits per second.

If slow lines, slow responsiveness of the destination host, great distance, etc., cause the normal network transit time to be half a second, then the throughput rate is reduced to only 4,000 bits per second. Similarly, we think that pipelining is essential for most networks to improve delay characteristics; data should travel in reasonably short packets.

To summarize, low delay requirements drive packet size smaller, network and host lines faster, and network paths shorter (i.e., fewer node-to-node hops). High throughput requirements drive the number of packets in flight up, packet overhead down, and the number of alternative paths up.

Error Control. We consider source-to-destination error control to comprise three tasks: detecting bit errors in the delivered messages, detecting missing messages or pieces of messages, and detecting duplicate messages or pieces of messages.

The former task is done in a straightforward manner through the use of checksums. A checksum is appended to the message at the source and the checksum is checked at the destination; when the checksum does not check at the destination, the incorrect message is discarded, requiring it to be retransmitted from the source. Several points about the manner in which checksumming should be done are worthy of note. (a) If possible, the checksum should check the correctness of the resequencing of the messages which possibly got out of order in their traversal of the network. (b) A powerful checksum is more efficient than alternative methods such as replication of a critical control field; it is better to extend the checksum by the number of bits that would have been used in the redundant field. (c) Unless encryption is desirable for some other reason it is simpler (and just as safe) to prevent delivery of a message to an incorrect host through the use of a powerful checksum than it is to use an encryption mechanism. (d) Node-to-node checksums do not fulfill the same function as end-to-end checksums because they check only the lines, not the nodes.

An inherent characteristic of packet-switching networks is that some messages or portions of messages (i.e., packets) will fail to be delivered, and there will be some duplicate delivery of messages or portions of messages, as described in the section on network properties. (Please note that throughout the remainder of this subsection we use the word "message" to mean either messages or portions of messages (i.e., packets).)

Missing messages can be detected at the destination through the use of one state bit for each unit of information which can be simultaneously traversing the network. An interesting detail is that for the purposes of missing message detection, the state bits used must precisely cycle through all possible states. For example, stamping messages with a time stamp does nothing for the process of missing message detection because, unless a message is sent for every "tick" of the time stamp, there is no way to distinguish the case of a missing message from the case where no messages were sent for a time.

Duplicate messages can be detected with an identifying sequence number such that messages which arrive from a prior point in the sequence are recognized as duplicates. What should be noted carefully here is that duplicate messages can arrive at the destination up to some time, possibly quite long, after the original copy, and the sequence number must not complete a full cycle during this period. For example, if a network goal is to be able to transmit 200 minimum length messages per second from the source to the destination and each needs a unique sequence number, and if it is possible for messages to arrive at the destination up to 15 seconds after initial transmission from the source, then the sequence number must be able to uniquely identify at least 3000 packets. It is usually no trouble to calculate the maximum number of messages that can be sent during some time interval. What is more difficult is to limit the maximum time after which duplicate messages will no longer arrive at the destination. One method is to put a timer in each message which is counted down as the message traverses the network; if the timer ever counts out, the message is discarded as too old, thus guaranteeing that no messages older than the initial setting of the timer will be delivered to the destination. Alternatively, in practice one can make a reasonably good approximate calculation of the maximum arrival time through study of all the worst case paths through the network and all the worst case combinations of events which might cause messages to loop around in the network.

In either case, there certainly must be mechanisms to resynchronize the sequence numbers between the source and the destination at node start-up time, to recover from a node failure, etc. A good practice is to resynchronize the sequence numbers occasionally even though they are not known to be out of step. A good frequency with which to do redundant resynchronization would be every time a message has not been sent for longer than the maximum delivery time. In fact, this is the maximum frequency with which the resynchronization can be done (without additional mechanisms); if duplicates are to be detected reliably, the sequence number at the destination must function without disruption for the maximum delivery time after the "last message" has been sent. If it is desirable or necessary to resynchronize the sequence numbers more often than the maximum time, an additional "use" number must be attached to the sequence number to uniquely identify which "instance" of this set of sequence numbers is in effect; and, of course, the packets must also carry the use number.

The next point to make about end-to-end error control is that any message going from source to destination can potentially be missing or duplicated; i.e., not only data messages but control messages. In fact, the very messages used in error control (e.g., sequence number resynchronization messages) can themselves be missing or duplicated, and a proper end-to-end protocol must handle these cases.

Finally, there must be some inquiry-response system from the source to the destination to complete the process of detecting lost messages. When the proper reply or acknowledgment has not been received for too long, the source may inquire whether the destination has received the message in question. Alternatively, the source may simply retransmit the message in question. In any case, this source inquiry and retransmission system must also function in the face of duplicated or lost inquiries and inquiry response control messages. As with the inter-node acknowledgment and retransmission system, the end-to-end acknowledgment and retransmission system must depend on positive acknowledgments from the destination to the source and on explicit inquiries or retransmissions from the source. Negative acknowledgments from the destination to the source are never sufficient (because they might get lost) and are only useful (albeit sometimes very useful) for increased efficiency.

Storage Allocation and Flow Control. One of the fundamental rules of communications systems is that the source cannot simply send data to the destination without some mechanism for guaranteeing storage for that data. In very primitive systems one can guarantee a rate of disposal of data, as to a line printer, and not exceed that rate at the data source. In more sophisticated systems there seem to be only two alternatives. Either one can explicitly reserve space at the destination for a known amount of data in advance of its transmission, or one can declare the transmitted copy of the data expendable, sending additional copies from the source until there is an acknowledgment from the destination. The first alternative is the high bandwidth solution: when there is no space, only tiny messages travel back and forth between the source and destination for the purpose of reserving destination storage. The second alternative is the low delay solution: the text of the message propagates as fast as possible.

In either case storage is tied up for an amount of time equal to at least the round trip time. This is a fundamental result -- the minimum amount of buffering required by a communications system, either at the source or at the destination, equals the product of round trip time and the channel bandwidth. The only way to circumvent this result is to count on the destination behaving in some predictable fashion (an unrealistic assumption in the general case of autonomous communicating entities).

As we stated earlier, our experience and analysis convinces us that if both low delay and high throughput are desired, then there must be mechanisms to handle each, since high throughput and low delay are conflicting goals. This is true, in particular, for the storage allocation mechanism. It has occasionally been suggested, mainly for the sake of simplicity, that only the low delay solution be used; that is, messages are transmitted from the source without reservation of space at the destination. Those people making the choice never to reserve space at the destination frequently assert that high bandwidth will still be possible through use of a mechanism whereby the source sends messages toward the destination, notes the arrival of acknowledgments from the destination, uses these acknowledgments to estimate the destination reception rate, and adjusts its transmissions to match that rate. We feel that such schemes may be quite difficult to parameterize for efficient control and therefore may result in reduced effective bandwidth and increased effective delay. If the source never sends to the destination so fast that the destination must discard anything, then the delay is very low, but the throughput is not as high as it might be. Further, unless the source pushes now and then, it will never discover that the destination is able to increase its throughput. On the other hand, when the source is pushing hard enough, the destination may suddenly cut back on its throughput, causing all the messages which will be discarded at the destination due to the sudden cutback to have to be retransmitted, increasing effective delay. If the destination could be predicted to accept traffic at a steady rate and vary this rate only very slowly, the type of feedback system cited above might work. In this case, unacknowledged messages should be retransmitted from the source to the destination shortly after the expected time for the acknowledgment to return has elapsed, if minimum delay and maximum throughput are to be obtained (this is in contrast to the often suggested practice of keying retransmissions to the discard rate). However, in practice, the time for the acknowledgment to return is likely to be very difficult to predict due to variations (possibly rapid) in the transit time of the communications channel and particularly in the response time of the destination. Furthermore, the greater the sum of transit time and response time, the looser and less efficient the feedback loop will be. In fact, there appear to be oscillatory conditions which can occur where performance degrades completely. (Note that if there is much possibility of message loss, then the acknowledgment and retransmission system should allow quite selective retransmission of messages rather than, for instance, requiring a complete window of messages to be retransmitted to affect retransmission of the specific messages requiring it; otherwise, message retransmission will use excessive bandwidth.)

The above discussion assumes that all mechanisms are attempting to minimize the probability of message discard. If, in addition to possible discards at the destination, the communications channel solves its internal problems (e.g., potential deadlocks) with cavalier discarding of messages, or if the destination solves its internal problems with cavalier discarding of messages, the detrimental effects of discarding (reduced effective bandwidth and increased effective delay) are probably drastically increased. Further, the above discussion assumed the destination was able to minimize the probability of discard. While this may be possible for a single source, we think it is unlikely that the destination will be able to resolve, in a way that does not entail excessive discards, the contention for destination storage from multiple uncoordinated sources. Detrimental contention for destination storage, in the absence of a storage reservation mechanism, happens practically continuously under even modest traffic loads, and in a way uncoordinated with the rates and strategies of the various sources. As a result, well-behaved hosts may unavoidably be penalized for the actions of poorly-behaved hosts.

In addition to space to hold all data, there must also be space to record what needs to be sent and what has been sent. If a message will result in a response, there must be space to hold the response; and once a response has been sent, the information about what kind of answer was sent must be kept for as long as retransmission of that response may be necessary.

Precedence and Preemption. The first point to note about precedence and preemption is that the total transit time being specified for most packet-switching networks of which we are aware is on the order of less than a few seconds (often only a fraction of a second). Thus, the traditional specifications (for example, low priority traffic must be able to preempt all other traffic so that it can traverse the network in under two minutes) no longer make much sense. When all messages traverse the network in less than a few seconds, there is generally no need to specify that top priority traffic must preempt other traffic, nor to specify the relative precedences between the other types of traffic. Priority can be used, however, to admit traffic into the network selectively.

Though priority is not strictly necessary for speed, it may be useful for contention resolution. It appears to us that there are three precedence and preemption strategies that are reasonable to consider for a packet-switching network. Strategy 1 is to permanently assign the resources necessary to handle high priority traffic; this guarantees the delivery time for the high priority traffic but is expensive and should only be done for limited high priority traffic. Strategy 2 is to preempt resources as necessary for high priority traffic. This can have two effects. Preempting packet buffers results in data loss; preempting internal node tables (e.g., the tables associated with packet sequence numbering) results in state information loss. State information loss means that data errors are possible which may go unreported. Strategy 3 is not to preempt resources, and to rely on the standard mechanisms with a priority ordering. This is simple for the nodes, but it does not of itself guarantee delivery within a certain time.

We think the correct strategy is probably a mixture of the strategies above. Possibly some resources, on a very limited basis, should be reserved for the tiny amount of flash traffic. This guarantees minimum delay without any queueing latency. For the rest of the traffic, the normal delivery times are probably acceptable. The presence of higher priority traffic can cause gradual throttling of lower priority traffic, without loss of state information. As the time to do this graceful throttling is normally only a fraction of a second, the higher priority traffic has no real reason to demand instantaneous, information-losing preemption of the lower priority traffic.

Message Size. The question is often asked, "If one increases packet size and decreases message size until the two become the same, will not the difficult message reassembly problem be removed?" The answer is that, perhaps unfortunately, message size and packet size are almost unrelated to reassembly.

We have already noted the relationship between delay and packet size. Delay for a small priority message is, to first order, proportional to the packet size of the other traffic in the network. Thus, small packets are desirable. Larger packets become desirable only when lines become so long or fast that propagation delay is larger than transmission time.

Message size needs to be large because the overhead on messages is significant. It is inefficient for the nodes to have to address too many messages and it may be inefficient for hosts to have too many message interrupts. The upper limit on message size is what can conveniently be reassembled, given node storage and network delays.

When a channel has an appreciable delay, it is necessary to buffer several pieces of data in the channel at one time in order to obtain full utilization of the channel. It makes little difference whether these pieces are called packets which must be reassembled or messages which must be delivered in order.

We do not feel that the choice between single- and multi-packet messages is as important as all the controversy on the subject would lead one to believe. There is agreement that buffering many data units in transit through the network simultaneously is a necessity. Having multi-packet messages is probably more efficient (as the extra level of hierarchy allows overhead functions to be applied at the correct, i.e., most efficient, level); having single-packet messages probably offers the opportunity for finer grained storage allocation and flow control mechanisms.

Multiplexing and Addressing. Up to this point in our paper, we have not been very specific about whether the abovementioned flow control, sequencing, error control, etc. mechanisms were performed for each pair of communicating processes, or whether several processes communicating between a given pair of source and destination nodes share a set of these control mechanisms. The tradeoff is between overhead and precision of control. If many conversations are multiplexed on each instance of a source-to-destination control mechanism, the control overhead is lower than if each conversation has its own control mechanism. On the other hand, if several conversations are multiplexed on the same control mechanism, all the conversations tend to have to be treated equally (e.g., if one is stopped, all are stopped); while if each conversation has its own control mechanism, exact decisions about the allocation of various resources to the various conversations can be made. To give some examples of the latter, conversations over separate control mechanisms can be given differing allocations, priorities, treatments of error conditions, etc.

Another issue is the management of the space available for control mechanisms when it is insufficient to handle the number of conversations competing for the communications channel. Should late-comers be left out until resources are available, or should some way be found to multiplex the available control mechanisms in time among the demanding conversations? We believe the latter should be done. The key here is not to allow users to explicitly acquire and hold resources (e.g., control mechanism space) needed for interprocess communication. Instead, the system should notice which users are actively communicating and dynamically gather the needed resources by garbage-collecting the resources previously being used by users which appear inactive. This dynamic assignment of resources is obviously not fundamentally different from the scheduling of any limited resource in an operating system (e.g., memory, CPU cycles, the I/O channel to the disk) and therefore has all the normal possibilities for thrashing, unfairness, and so on, if care is not taken.

Once decisions in all of the above areas of multiplexing are made, one must choose the addressing mechanism and formats to be used. This is usually quite straightforward. The main point here is that addressing comes last; but very often we see designs begun by choosing the addressing system and format. A similar statement can be made about the choice of all other message formats.

2.4.3.2 Original Implementation

In this section we describe the mechanisms by which the IMPs in the ARPANET manage the flow of data from a source host to a destination host.

2.4.3.2.1 Message Format

In the ARPANET, a host presents messages to the IMP to which the host is directly connected. These messages must be less than about 8100 bits long, and the messages are transmitted between the host and IMP over the host/IMP interface. This interface has two parts, the hardware part and the software part.

The hardware part of the host/IMP interface itself has two parts, a standard portion supplied with the IMP which is (almost) identical for all hosts and a special portion supplied by the host. For simplicity and power, the standard interface has been defined to be full duplex. Electrically, the standard interface follows a bit-by-bit handshaking procedure. The procedure is something like "I'm ready to send a bit." "I'm ready to receive a bit." "Here's the bit." "I got the bit." "Good!" The procedure also provides for saying "That's the last bit in the message." In our opinion this procedure is important. The ARPANET has to connect together all kinds of different computers with different word lengths, different speeds, different loading, and so forth; and it is desirable to place only minimal constraints on the hosts' behavior. The above procedure permits this. The asynchronous, bit-by-bit serial interface permits both the host and the IMP to be able to start and stop transmission whenever necessary. Incidentally, the IMP's side of the above procedure is implemented entirely with hardware.

An optional synchronous host/IMP interface is also available. It is considerably more complicated, less flexible, and slower than the normal asynchronous, serial interface; and its use is recommended only when an communications interface suitable for operation over long distances is required. HDLC would have been a better choice had it been available at the time.

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

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

An IMP breaks messages arriving from its hosts into packets 1000 or fewer bits long. As the IMP segments a message into packets, it appends to the front of each packet some control information called the header. The header contains the destination, the packet number within the message, a message sequence number which is used for reconstructing the message stream as the messages arrive at the destination, and other control information (e.g., priority information) copied from the message into each constituent packet. This message segmentation and message reconstruction is completely invisible to the hosts in the ARPANET implementation of messages and packets.

2.4.3.2.2 The Original Source-to-Destination Transmission Procedure

Having given this generalized introduction to the subject of message processing, we turn to an examination of the original ARPA network implementation, in effect from 1970 to 1972. This system was based on the concept of a link, a logical connection between two hosts, on which only one message at a time could be transmitted. The source and destination IMPs kept tables of active links over which messages had recently been transmitted. This technique performed several of the functions described above, though some in a rather rudimentary fashion. As we shall see, some of the functions noted above for the 'source' and 'destination' were performed in the source IMP and destination IMP, while others were left partly in the domain of the source host and destination host.

First, the buffering in the network was up to the host program; it could send only one message per link but the IMP placed no limits (in practice, a very high limit) on the number of links that could be used. Pipelining was provided for, and still is, by breaking up messages (up to 8000 bits) into packets (up to 1000 bits) to decrease delays. A reassembly process at the destination is necessary to perform ordering, and is also implied by pipelining. Error control was performed by both the source and destination IMP by maintaining a sequence number fo the messages on each active link. Each message and RFNM (Ready For Next Message, the delivery confirmation) carried this identifying number, and thus missing and duplicated transmissions could easily be detected. Message sequencing was no problem with the one-message-one-link rule, since the destination host was then guaranteed to receive an ordered data stream on each link being used by the source host. Storage allocation was quite primitive;

the source IMP did not hold a copy of any message, nor was storage at the destination IMP reserved. Instead, the packets of the message were simply transmitted to the destination, where they were accepted if there was sufficient storage, or otherwise rejected; that is, the destination IMP would not send an ack back to the previous IMP, which would subsequently time out and retransmit the packet. This approach therefore used the storage of the store-and-forward subnetwork when the destination IMP was full. Finally, flow control was likewise rudimentary and also somewhat in the province of the hosts, since it rested primarily on the technique of permitting only one message per link. This concept provided flow control to the extent that each link was well-controlled; the rate at which the source host could send was tied to the rate of the destination host, on a given link. But if too many links were used, no adequate flow control measures were present.

A brief critique of this first approach can be offered: its major advantage, to our perspective now, was its simplicity and therefore its basic reliability. The link table in the source and destination IMPs was dynamic, accessed by a hash key, and garbage-collected on a time-out. The format of the table is illustrated in Figure 2-28. The one-message rule meant that the link tables were easy to deal with and were self-synchronizing. It was fair since all hosts had equal access to the common line

Figure 2-28 Link Table Format

tables and storage pool in the IMPs. This technique was also efficient in terms of allowing hosts great freedom to achieve minimum delays and maximum throughput levels, since it did not impose any overhead transactions. It was not efficient in IMP storage, since a fairly large hash table entry was needed to describe a single message, or in IMP bandwidth, since the hashed access was time-consuming.

The major criticism of this scheme is simple: it can lead to a storage-based deadlock that we call reassembly lockup, as illustrated in Figure 2-29. This deadlock occurs under high load, which first causes reassembly congestion, in which efficiency is reduced due to retransmissions.

In Figure 2-29, IMP 1 is sending multi-packet messages to IMP 3; a lockup can occur when all the reassemble buffers in IMP 3 are devoted to partially reassembled messages A and B. Since IMP 3 has reserved all its remaining space for awaited packets of these partially reassembled messages, it can only take in those particular packets from IMP 2. Those outstanding packets, however, are two hops away in IMP 1. They cannot get through because IMP 2 is filled with store-and-forward packets of messages C, D, and E (destined for IMP 3) which IMP 3 cannot yet accept. Thus, IMP 3 will never be able to complete the reassembly of messages A and B.

Figure 2-29 Reassembly lockup

We were always aware that hosts could defeat this flow control mechanism by "spraying" messages over an inordinately large number of links, but we counted on the nonmalicious behavior of the hosts to keep the number of links in use below the level at which problems occur. However, simulations and experiments artificially loading the network demonstrated that communication between a pair of hosts on even a modest number of links could defeat our flow control mechanism; further, it could be defeated by a number of hosts communicating with a common site even though each host used only one link. Simulations showed that reassembly lockup may eventually occur when over five links to a particular host are simultaneously in use. We should stress that ordering is the basic cause of the problem, not reassembly of multi-packet messages.

Note that much of the trouble with this approach to message processing is that the responsibility for tasks such as buffering, sequencing, and flow control was shared between the IMPs and the hosts. That is, the IMP was responsible for sequencing, flow control, etc., with regard to a single link, but the host was responsible for the number of links that were active at once. As we shall see, we changed the ARPA network so that the IMPs had complete and effective control over all the message processing functions in the communications subnetwork, while still leaving the hosts freedom to exercise their own controls as well.

2.4.3.2.3 The Next Implementation

In mid-1972, the message processing algorithms just described were changed in several substantial ways. First, reassembly lock-ups were prevented by providing explicit storage allocation. Secondly, the sequencing and error control schemes were modified to be more general and more efficient. These techniques have remained in the ARPA network until late 1974. Further changes are described below in Section 2.4.3.3.

As we noted before, if storage is allocated one can optimize for low-delay transmissions. If the storage is allocated at the destination, one can optimize for high-bandwidth transmissions. To be consistent with our view of a balanced communications system, we have developed an approach to reassembly congestion and lock-up that utilizes some buffer storage at both the source and destination; our solution also utilizes a request mechanism from source IMP to destination IMP.

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

Effective bandwidth is maximized for sequences of long messages by permitting all but the first message to by-pass the request mechanism. When the message itself arrives at the destination IMP is about to return the Ready-For-Next-Message (RFNM), and end-to-end delivery confirmation message, the destination IMP waits until it has room for an additional multi-packet message. It then piggy-backs a storage allocation on the RFNM. It the source host is prompt in answering the RFNM with its next message, al allocation is ready and the message can be transmitted at once. If the source host delays too long, or if the data transfer is complete, the source IMP returns the unused allocation to the destination. With this mechanism, we have minimized the inter-message delay and the hosts can obtain the full bandwidth of the network.

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

The above mechanisms make the IMP network much less sensitive to unresponsive hosts, since the source host is effectively held to a transmission rated equal to the reception rate of the destination host. Further, reassembly lockup is prevented because the destination IMP will never have to turn away a multi-packet message destined for one of its hosts, since reassembly storage has been allocated for each such message in the network. The cost o this system is small; the onetime cost of implementing the more complex algorithm, plus a negligible overhead in line bandwidth.

Since the link mechanism was no longer needed for flow control (and had not been successful anyway), we felt that a less costly mechanism should be employed for sequence control. The link mechanism was therefore eliminated from the IMP subnetwork. RFNMs are still returned to the source hosts on a message basis, but they are used only as acknowledgments to messages. To replace the per-link sequence control mechanism, we decided upon a sequence control mechanism based on a single logical "pipe" between each source and destination IMP. Each IMP maintains an independent message number sequence for each pipe. A sequence number is assigned to each message at the source IMP and this message number is checked at the destination IMP. All hosts at the source and destination IMPs share this message space. Out of an 8-bit message number space (large enough to accommodate the settling time of the network), both the source and destination keep a small window of currently valid message numbers, which allows several messages to be in the pipe simultaneously, permitting high throughput rates. Messages arriving at a destination IMP with out-of-range message numbers are duplicates to be discarded. The window is presently four numbers wide; this parameter must be chosen in relation to the delay and bandwidth of the average end-to-end network path. The message number serves the two purposes of sequencing and error control.

A sequence control system based on a single source/destination pipe, however, does not permit priority traffic to go ahead of other traffic. We solved this problem by permitting two pipes between each source and destination, a priority (or low-delay) and a non-priority (high-bandwidth) pipe, with duplicate detection performed in common.

hosts may, if they choosed, have several messages outstanding simultaneously to a given destination but, since priority messages can 'leapfrog' ahead, and the last message in a sequence of long messages may be short, priority can no longer be assigned strictly on the basis of message length. Therefore, hosts must explicitly indicate whether a message has priority or not.

With message numbers and reserved storage to be accurately accounted for, cleaning up in the event of a lost message must be done carefully. The source IMP keeps track of all messages for which a RFNM has not yet been received. If the RFNM is not received within a certain time (presently about 30 seconds), the source IMP sends a control message to the destination enquiring about the possibility of the message been lost. The destination responds to this message by indicating whether the message in question was previously received o not. The source IMP continues enquiring until it receives a response. This technique guarantees that the source and destination IMPs keep their message number sequences synchronized and that any allocated space will be released in the rare case that a message is lost in the subnetwork because of a machine failure.

The format of the current message tables is shown in Figure 2-30. TMESS is the next message number to send; there are four bits to indicate whether the messages to which they refer (TMESS, TMESS-1, TMESS-2, TMESS-3) are still outstanding (unanswered). The ORD-T field gives the next 2-bit ordering number to use on priority messages. RMESS and the associated fields work in an

Figure 2-30 Message Table Format

analogous fashion for received messages, which are marked complete when they are given to the IMP-to-host routine. AMESS and RALLY are used to send back replies to messages in the window; AMESS is the next message to reply to; RALLY holds what kind of a reply to send. INC times out overdue transmissions; RST-T and RST-R time out after a message number reset (as explained later) is performed on the transmit or receive side, to prevent more than one reset per 30 seconds, and thus detect duplicates; and RST indicates that a transmit reset is still in progress.

Several aspects of this data structure are worth noting. First, the tables are quite compact compared to the earlier link tables, since a 4-word entry describes the complete state of four messages in flight between two IMPs, as opposed to three words in each of the source and destination IMPs to describe one message. Also, it is easy to access since it is indexed by IMP. Secondly, the structure of the RALLY table is important, both from the point of view of deadlock prevention and as a compact representation.

The destination IMP queues up replies (such as RFNMs, ALLOCATES, etc) before they can be sent. An entry in this table is either blank or contains an RFNM, an ALLOCATE, etc. The IMP number to send it to and the message number it refers to are implicit in the position of the entry on the table. That is, the IMP has table storage to save all four possible replies that could be pending, given the window size of 4, to all possible source IMPs (currently 64). This is important; the destination IMP always has a place to put a reply. When the reply is sent, the entry is marked as blank, and the number indicating which message number to answer next, AMESS, is incremented.

Without table storage for each possible reply, in highly bit-coded form, the IMP would have to deal with queues of pending replies that could grown very long. Of course, no limit can be placed on such a queue without running the risk of causing reassembly lockup. All packets must be accepted by the destination IMP from the store-and-forward part of the network to avoid such deadlock states.

In fact, the RALLY table is even more compact than it appears, since it can also keep a record of what kind of reply was sent back for each message number. That is, an entry for message N in the RALLY table is either a pending reply, or, if it is marked blank, a record of the reply sent for message N-4. Then, should the source IMP fail to receive the answer and send an incomplete query 30 seconds later, the destination IMP can send back the correct original answer. Note that no more entries need to be kept to do this. If RALLY holds answers for messages 7, 8, 9, and 10, and then 7 is sent off, the destination IMP needs to remember the answer is sent to 7 only until it has an answer queued up for message 11. When it overwrites the answer for message 7, and the entry is marked full again, the source IMP will not now enquire about message 7; its message number window has advanced past that point.

A final aspect of this structure worth noting is the resynchronization mechanism we have installed (we found that, in practice, all subnetwork systems need to be checked for consistency and fixed if wrong). It is suggested by the idea just noted; the source IMP will legitimately enquire about messages only up to one window's worth behind the current position. This is best shown by explaining the actions the IMP takes under various circumstances, as illustrated in Figure 2-31.

Case 4 is the one just discussed and case 5 triggers message number resynchronization. It says that if an IMP receives an incomplete inquiry for a message outside of the range that the destination expects the source IMP to be using, it sends back a special out-of-range message. Note that the offending incomplete may simply be an old duplicate. Therefore, the source IMP, when it receives the out-of-range message, will check its message number to see if it is still waiting for the answer to that inquiry. If it is, a "message number lockup" exists (presumably, this results from some software or hardware failure affecting the message tables). If not, an old duplicate incomplete has landed

Case	Action	Message Type	Position Relative to destination window
1	Accept	Not incomplete	In window
2	Discard as duplicate	Not incomplete	Out of window
3	Discard any packets of message in progress	incomplete	In window
4	Send back duplicate correct reply	Incomplete	In 4 numbers just previous to current window
5	Send back an out-of-range message	Incomplete	Outside of window of 4, and also window of 4 previous

Figure 2-31 Action by Destination IMP on Various Messages

at the destination, and the out-of-range message can be ignored, Now, in the case that a lock-up has been detected, the IMPs take the steps shown in Figure 2-32 to free it.

This set of mechanisms allows the IMPs to:

keep more accurate message accounting;

detect lock-ups with a single incomplete transmission time-out delay;

report the frequency of certain events to the Network Control Center, including:

incomplete queries sent

out of range incompletes received

receiver resets

sender resets

(by comparing the last two ARPANET operations personnel can try to understand the causes for any message number lock-ups that do occur, and perhaps fix the failing hardware or software.

In short, the system is event-driven, giving a maximum amount of diagnostic information while still performing the necessary function at relatively low cost in delay, complexity, or bandwidth.

Source IMP	Destination IMP
Mark 'reset-in-progress' bit, prohibiting traffic to this IMP.	
Send a reset message.	get reset message. Reset all receive tables.
	Send back 'reset reply' message
Receive reset reply message. If 'reset-in-progress' bit is on, reset all transmit tables, return an Incomplete Transmission for all messages in progress to that IMP. Clear the 'reset-in-progress' bit. Resume normal operations.	Do not believe any other reset message foo another 30 seconds, to ignore duplicates (we assume no duplicate is more than 30 seconds old).

Figure 2-32 Action by Source and Destination IMPs on Detection of Look-up

2.4.3.3 Subsequent Modifications

The final topic to consider under the general heading of problems with the next ARPA network approach is that of adapting the various techniques described for use in large networks with hundreds of IMPs and thousands of hosts. The key point is that the cost of keeping linear tables indexed by IMP, host, or anything else, becomes prohibitive at some network size. The simplicity and reliability of such tables, and their resulting freedom from deadlock, are naturally desirable but cannot be maintained beyond a certain level because of the high cost. The 1972-1974 tables were structured for a network of up to 64 IMPS (each IMP has 16K of memory), four hosts per IMP (plus 4 fake hosts internal to each IMP), with line bandwidth of 50Kbs typically, and round-trip delays of roughly 1/2-second. With the growth of the ARPA network, and the introduction of new technology in IMPs and circuits, including satellites, all the parameters above must be reexamined.

2.4.3.3.1 Dynamic Blocks

The approach we decided to take is to make all the data structures in the IMP associated with message processing take the form of dynamic blocks, each containing a few words of storage. One example is the transaction block, which the source IMP creates when a message is initiated. This block keeps track of whether the message has a copy kept at the source, whether it needs an allocate, and so on. A key function of the block is to hold a copy of the message header to identify the message. When the RFNM returns from the destination, a RFNM for the source host can be formatted in place in the transaction block, solving a difficult storage allocation problem. Thus transaction blocks can be on a host queue (RFNM ready to be sent). In addition, they can hold the information about whether an ALLOCATE was sent back with the RFNM.

Another example of the dynamic block is the reassembly block, which is kept at the destination IMP for the purpose of ordering the packets of a multi-packet message. It also contains the message header, and a list of which packets have arrived. If none has arrived yet, it constitutes a record of an allocation that has been sent to a particular host and IMP. It can also be on several queues: a free list, an active list, or a host queue (this last is an implementation option we did not choose).

In keeping with this general philosophy of table structure, we redesigned the current message tables to use message blocks instead. These blocks are made dynamic by keeping them only as long as a conversation between two hosts is active. This concept allows the set of hosts at an IMP to send messages to an arbitrary number of other hosts over a wide range of addresses, with a limited number of message blocks. The message number tables on each block can work exactly as they do now, and the other message processing functions such as flow control can also proceed accordingly.

In order to successfully and efficiently handle the large number of conversations with various hosts that can be simultaneously in progress, all of the data structures in the IMP associated with message processing take the form of blocks dynamically gathered from a pool of blocks, each containing a few words of storage. The alternative of keeping linear tables indexed by IMP, host, or anything else is prohibitively expensive as the network becomes large.

One example of such a dynamic data structure is the transaction block which the source IMP creates when a message is initiated. This block keeps track of whether the message has a copy kept at the source, whether it needs an allocate, and so on. A key function of the transaction block is to hold a copy of the message header to identify the message. When a Ready For Next Message returns from the destination, a Ready For Next Message for the source host can be formatted in place in the transaction block, solving a difficult storage allocation problem. The transaction blocks can be on a free list, on an active list (i.e., message outstanding), or on a host queue (i.e., Ready For Next

Message to be sent). In addition, they can hold the information as to whether an "allocate" was sent back with the Ready For Next Message.

Another example of a dynamic block is the reassembly block which is kept at the destination IMP for the purpose of ordering the packets of a multi-packet message. It also contains the message header and a list of which packets have arrived. If none have arrived yet, it constitutes a record of an allocation that has been sent to a particular host or IMP. It can also be on several queues: a free list, an active list, or (theoretically) a host queue (this last is an implementation option which was not chosen in the case of the IMP system). Notice that in each of these cases, that of the transaction block and the reassembly block, the data structure represents an important resource. There can only be a finite number of blocks; their use must be allocated among several competing source and destination hosts; critical questions of efficiency and fairness arise in the process of block allocation.

In keeping with this general philosophy of table structure, message blocks are used to keep track of the host/host message numbers. This concept allows the set of hosts on an IMP to send messages to an arbitrary number of other hosts over a wide range of addresses, with a limited number of message blocks. Several issues arise with consideration to dynamic message blocks:

There must be control messages between the source and destination IMPs of the form "get a block" and "got a block", in order to establish a "conversation" between a given pair of hosts on the two IMPs.

There must be an error control mechanism to detect duplicate or missing "get a block" and "got a block" messages.

Once a conversation is established, messages can flow. Then there must be a technique to distinguish messages in this conversation from old duplicates from a previous conversation between this pair of hosts. The messages and packets must carry some identifying number for this purpose.

Conversations should be able to be broken by either end if an IMP finds its storage for message blocks filling up. The messages to do this, "do a reset" and "did a reset" as we call them, must also be error-controlled.

Conversations should begin without undue startup delay.

The dynamic tables should be simplex; that is, one-directional message numbers should be used to avoid deadlocks of the form A tries to talk to B, B tries to talk to C, C tries to talk to A, and none of A, B, or C have any more free blocks to use to begin a new conversation.

The tables should be fast to access at both the source and destination IMPs.

The method used the the ARPANET for implementing this system is based on a small pool of blocks, each of which carries a "use number", four bits wide, permanently associated with the block. All packets exchanged between IMPs carry the block number to be used in processing the packet and the use number. As explained in more detail below, these numbers provide the key information necessary for error control in a dynamic block environment.

The system works as follows: When a host at an IMP gives its IMP a message, the IMP first looks to see if a block exists for that source host to that destination IMP and host. Instead of searching through all blocks, a "bucket sort" (a simple hash) is performed, to cut the average search length by some number (by 16 in actual fact) using a few bits of the key (source host, destination IMP, destination host) to begin the sort. It then checks successive blocks in the bucket led to by the sort (all the blocks in a single bucket are actually chained together for speed of access). If no block is found already existing for this key, a new block must be acquired at both the source and destination IMPs for this host/host conversation. The source IMP gets a block from the list of free blocks (the case of no free blocks is discussed later). It puts the new block at the top of the bucket (head of the chain) that it just searched. The program then copies in all the key information, adds one to the use number of the block, initializes the transmit message number entry, turns on a bit to indicate that the block is not yet in use, and calculates the index of the block it found.

The program then constructs a "get a block" message which includes the index number calculated just above and the use number from the block and sends the "get a block" message to the destination IMP. The source IMP then waits for an answering "got a block". The "get a block" must be retransmitted every few seconds if no answer returns. When a "got a block" is returned, the initialization bit is cleared, and the foreign block and use number which arrived in the "got a block" are copied into the source block. Now the message can be sent using the message number window, allocation, and ordering techniques described above.

At the destination IMP, when a "get a block" is received, the program tries to get a free block. If the free list is empty, nothing more is done (in effect, the "get a block" request is ignored). If a free block is available, all the key data carried in the "get a block" is copied into the block and a "got a block" message is constructed including the key data and sent to the source.

When the source IMP sends a packet to the destination, it carries the foreign block number and use number which are kept in the source block. The destination IMP uses this number to calculate the address of its message block and verifies the key information including the use number. In all other ways, the logic discussed above for accepting packets within the message number window is followed. When a Ready For Next Message is generated at the destination IMP, it carries back the block number and use number kept in the destination block. This allows the source IMP to find its block and detect duplicate Ready For Next Messages in a simple manner.

We have explained how blocks are acquired. It is also necessary to discard blocks. There are two timers in the transmit and receive block. One counts two seconds of inactivity, the other two minutes of inactivity. A block _may_ be discarded after two seconds of idle time, and a block _must_ be discarded after two minutes of idle time. The two-second timer serves the function of quickly time-multiplexing the use of the dynamic blocks by many different conversations. The two-minute timer is used merely in a background manner to refresh the pool of free blocks by garbage collecting blocks associated with conversations long inactive, thus avoiding more often expensive searches for a free block at the instant a new free block is required. The two-minute timer could be made longer if desired, to allow hosts to pause longer in a conversation without incurring some setup delay; the two-second timer value is more critical. If, as we assume, duplicate packets may arrive at an IMP up to thirty seconds after initial arrival, then a mechanism is needed to allow blocks to be created and deleted more often than every thirty seconds that protects against the same pair of hosts using the same block twice and not catching a duplicate. The 4-bit use number allows sixteen cycles of acquisition and discard of the same block in any thirty second interval. Therefore, at least two seconds must elapse after acquisition of a block before it can be discarded. The rule that the message number must be idle for two seconds before a block can be discarded is actually stronger, but it seems a good rule to prevent thrashing and inefficiencies. The two minute timer serves to keep everything in synchronization in steady state (actually the timer runs for two minutes on the transmit side, and for slightly longer on the receive side, to prevent races).

The best policy to follow for choosing when to delete blocks seems to be for an IMP to attempt to find deletable blocks (either transmit or receive) when its free block list goes below some clip, say 10% of the total pool. When this happens, if the program can locate a transmit block that can be deleted, it sends out a "reset" message to the destination, which causes the destination to discard its block and to send a "reset reply" back to the source which causes the source to discard its block. If the program finds a receive block to delete, it sends a "reset request" message to the source which then follows the above protocol for performing a reset. On all these message, the block number and use number provide a duplicate detection facility, since a given block with a particular use number can only be reset once.

At this point, it is worth noting the duplicate detection mechanism applied to the "get a block" and "got a block" messages. The "get a block" carries no identifying information other than the addresses of the source and destination and the source block number and use number. If a duplicate arrives during the conversation it initiated, it can be detected; likewise, if it arrives during any other later conversation between those two hosts it can be detected. The only problem arises if the "get a block" duplicate arrives at the destination when no block exists between the two hosts. Then the destination IMP must get a block and return a "got a block" to the source. If the source receives a "got a block" when it is not expecting one, it must send out a "reset" to clear the destination. This fixes the problem, and since the program ignores "reset" and "reset reply" messages which do not match the block and use number then active, it also takes care of the case of duplicate "got a block" messages and unwanted "reset" and "reset reply" messages generated to deal with these circumstances.

This concludes our discussion of dynamic message blocks. They do not present a large penalty in storage, delay or packet size; they are reliable because they are maintained dynamically and all communications are error controlled; they allow the available blocks to be quickly multiplexed in time among a number of conversations (albeit at reduced performance for each conversation) rather than shutting some conversations out as the available message transmission resources become fully used; and they allow separate message numbers between each host/host pair even when the total number of hosts in the network is beyond a number where fixed linear tables indexed by host would be possible. In fact, expanding on the final point, the dynamic message block mechanism permits multiple message number streams between a given host/host pair. The ARPA implementation currently permits two, one priority stream and one normal stream; but additional streams are easy to imagine (e.g., special "permanent" streams which cannot be preempted after two seconds, streams which follow a different ordering or retransmission criterion or even the lack of one, streams which have varying message number windows depending perhaps on stated host needs). Altogether, the dynamic message block mechanism appears to permit very flexible and reliable operation.

2.4.3.3.2 Restructured Message Numbers

The next major change involves not the packaging of message number tables into blocks but the restructuring of the actual message number tables themselves. The following changes are planned under this general heading:

a restructuring of RMESS, AMESS, and RALLY to simplify them and eliminate a redundancy;

expansion of the message window from 4 to 8;

extension of the priority bit concept to a general multi-level handling type, with independent message sequences for each type;

the capability of always sending back the correct duplicate reply (RFNM, Dead, etc) in the event that the first reply is lost; this has not been implemented in the current system;

implementation of the message number scheme in such a way as to facilitate traffic through the network that does not use all (or any) of the message-processing mechanisms.

The current program maintains RMESS, the next message number to accept, and AMESS, the next message number for which to send and answer (RFNM, ALLOCATE, etc), which are obviously not independent. With a window of 4, the rule is RMESS =< AMESS =< RMESS + 7. Further, a single bit is kept with RMESS to indicate if a message has been received, which cannot provide any information on exactly what state the message is in. Finally, the 4 bits in RALLY are used to specify the type of answer to return. The basic idea, then, is to consolidate these data structures into some improved tables that serve the requirements better. Specifically, we can meet these goals with one message number and one set of status bits per message number.

The plan we have in mind is to keep RMESS as the next message number to accept (plus an offset, possibly), and use the bits in a new word, RSTATE, to indicate the state of each message in the window, as follows:

idle: nothing doing; reply sent for this number - 8

request: got a request, allocate should be sent

message: accepting this message

reply: got a message, reply should be sent

We will then use another word, RTYPE, to replace RALLY in detailing what kind of a reply to send (RFNM, RFN,/ALL, DEAD, INC) in the idle and reply states and also to provide additional data of other kinds, as desired, in the request and message states. Figure 2-33 shows how RMESS and RSTATE interact. Note particularly the ambiguity of the RSTATE + I entry: it refers to RMESS + I unless it is a reply, then it refers to RMESS + I - 8.

Figure 2-33 Restructured Message Numbers

Finally, the format of the message block containing the new restructured message numbers is shown in Figure 2-34.

Figure 2-34 Message Block Formats

2.4.4 host-IMP

In the previous section we discussed a number of issues of end-to-end procedure design which must be considered wherever the procedures are implemented, whether in the subnetwork or in the hosts. In this section we discuss the proper division of responsibility between the subnetwork and the hosts.

2.4.4.1 Basic Concepts

Extent of Message Processing in the Subnetwork. There has been considerable discussion in the packet-switching community about the amount and kind of message processing that should be done in communications subnetworks. An important part of the ARPA Network design which has become controversial is the ARPANET system of messages and packets within the subnetwork, ordering of messages, guaranteed message delivery, and so on. In particular, the idea has been put forth that such functions should reside at host level rather than subnetwork level.

We summarize the principle usually given for eliminating message processing from the communications subnetwork: a) for complete reliability, hosts must do the same jobs, and therefore the nodes should not; b) host/host performance may be degraded by the nodes doing these jobs; c) network interconnections may be impeded by the nodes doing message processing; d) lockups can happen in subnetwork message processing; e) the node would become simpler and have more buffering capacity if it did not have to do message processing.

The last point is true, although the extent of simplification and the additional buffering is probably not significant, but we believe the other statements are subject to some question. We have previously given our detailed reasons for this belief. Here we simply summarize our main contentions about the place of message processing facilities in networks:

a. A layering of functions, a hierarchy of control, is essential in a complex network environment. For efficiency, nodes must control subnetwork resources, and hosts must control host resources. For reliability, the basic subnetwork environment must be under the effective control of the node program -- hosts should not be able to affect the usefulness of the network to other hosts. For maintainability, the fundamental message processing program should be node software, which can be changed under central control and much more simply than all host programs. For debugging, a hierarchy of procedures is essential, since otherwise the solution of any network difficulty will require investigating all programs (including host programs) for possible involvement in the trouble.

b. The nature of the problem of message processing does not change if it is moved out of the network and into the hosts; the hosts would then have this very difficult job even if they do not want it.

c. Moving this task into the hosts does not alleviate any network problems such as congestion, host interference, or suboptimal performance but, in fact, makes them worse since the hosts cannot control the use of node resources such as buffering, CPU bandwidth, and line bandwidth.

d. It is basically cheaper to do message processing in the nodes (small inexpensive computers) than in the hosts, and it has very few detrimental effects.

Peripheral Processor Connections. In a number of cases, an organization has desired to connect a large host to a network by inserting an additional minicomputer between the main host and the node. The general notion has been to locate the host-host transmission procedures in this additional machine, thus relieving the main host from coping with these tasks. Stated reasons for this notion include:

- It is difficult to change the monitor in the main host, and new monitor releases by the host manufacturer pose continuing compatibility problems.

- Core or timing limitations exist in the main host.

- It is desirable to use I/O arrangements that may already exist or be available between the main host and the additional mini (and between the mini and the node) to avoid design or procurement of new I/O gear for the main host.

While this approach may sound good in principle, and, in fact, may be the only possible approach in some instances, it often leads to problems.

First, the I/O arrangements between the main host and any preexisting peripheral processor were not designed for network connection and usually present timing and bandwidth constraints that greatly degrade performance. More seriously, the logical protocols that may have preexisted will almost certainly preclude the main host from acting as a general purpose host on the network. For instance, while initial requirements may only indicate a need for simple file transfers to a single distant point, requirements tend to change in the face of new facilities, and the network cannot then be used to full advantage.

Second, the peripheral processor and its software are often provided by an outside group, and the host organization may know even less about their inner workings than they know about the main host. The node is centrally maintained, improved, modified, and controlled by the Network Manager, but the peripheral processor, while an equally foreign body, is not so fortunate. This issue alone is crucial; functions that do not belong in the main hosts belong in centrally monitored network equipment. Note that it is exactly those host groups who are unwilling to touch the main host's monitor who will be unlikely to be able to make subtle improvements in the protocols, error message handling and timing of the peripheral processor. From a broader economic view, common functions belong in the network and should be designed once; the peripheral processor approach is a succession of costly special cases and the total cost is greatly escalated.

The long term solution to the dilemma is to have the various manufacturers support hardware and software interfaces that connect to widely used networks. This is not likely to occur until commercial networks exist and are widely available. In the meantime, potential host organizations that wish to use early networks (like the ARPANET) should try to find ways to put the network connection directly into the main host. An anthropomorphic illustration may be helpful; the network is, among other things, a set of standardized protocols or languages. A potential network host is in the position of a person who needs to have dealings with people who speak a language he does not know. If he does not want to learn the language, he can indeed choose to use an interpreter, but performance is poor, the process is very inconvenient, expensive, and unpleasant, and subtle meaning is always lost. The situation is quite similar when a host tries to work through a peripheral processor. If a host wishes to interact with a network, it is usually unrealistic to try to make the host think that the network is a card reader or some other familiar peripheral. As usual, you get what you pay for.

Other Message Services. Other commonly suggested design requirement is for storage in the communications subnetwork, usually for messages which are currently undeliverable because a host or a line is down. This requirement should have no effect whatsoever on the design of the communications part of the network; it is an orthogonal requirement which should be implemented by providing special storage hosts at strategic locations in the network. These can be at every node, at a few nodes, or at a single node, depending on the relative importance of reliability, efficient line utilization, and cost.

Another commonly suggested design requirement is for the communications subnetwork to provide a message broadcast capability; i.e., a host gives a message to its node along with a list of host addresses and the nodes somehow send copies to all the hosts in the list. Again we believe that such a requirement should have no effect on the design of the communications part of the network and that messages to be broadcast should be sent to a special host (perhaps one of the ones in the previous paragraph) for such broadcast.

2.4.4.4 Conclusions

We conclude this section on planned changes to the ARPA network with a brief look at some modifications we would like to make to the interface between the IMP and the host. These plans are affected by the enormous difficulty of getting the host computers (of which there are more than 50) all to change their programs. Therefore, any changes we do make are likely to be installed so that they are compatible with existing conventions and formats. By this we mean that the IMP and host will have to identify which conventions are in use, and the IMP will have to keep the software to support both formats for an arbitrary length of time.

One simple kind of change, yet a difficult one to implement everywhere, is an extension of many of the fields in the message leader, to allow more than 64 IMPs and more than four hosts per IMP. We would also like to add other fields to the leader, or extend the meaning of existing fields. For example, the idea that the host interface should not be stopped without the host's consent means new message types must be defined.

We also feel that the host interfaces should be made to run as fast as possible. This is not so much to allow hosts to get higher throughput, since they will still be limited by network circuit bandwidths (though it will speed up intra-IMP messages), but to cut down delays on host lines. This would make it easier for us to make another change: not sending back the RFNM until the whole message is accepted by the host. Currently, the RFNM is generated after the host accepts the first packet. This is done for two reasons. First, it is assumed that the host-IMP interface is error-free and further that once the host begins to accept bits of the message it will accept it all. Secondly, to delay the RFNM until the end of the message would add tens or hundreds of milliseconds of delay to the roundtrip time. Speeding up the host interfaces would reduce this delay, and we would prefer to make the RFNM mean that all the message was delivered.

Again on the subject of reliability and accurate message accounting, we have mentioned that end-to-end checksumming would also be desirable. Perhaps error detection and retransmission over the IMP-host interface is the right way, perhaps the IMP should generate and verify checksums, perhaps the hosts should. This is really a subject for negotiation among the various parties. It is our basic belief that the subnetwork at least should provide such a message-processing facility, and that hosts can augment it if they desire.

We have also come around to believing that the more information that the IMP and host can exchange about the status of specific messages and their states generally, the better the network is to use. Therefore, the IMP now reports to the host as closely as it can why a message could not be delivered,, distinguishing about a dozen cases at present. Further, we have implemented 'host going donw' and 'IMP going down' messages, which carry information on why the system is going down, when it is scheduled to go down, and when it is scheduled to be back up. This information can also be applied on a message-by-message basis, to supplement any non-delivery diagnostics. Generally, we feel such information mechanisms are an important adjunct to the basic message-processing facilities. They greatly facilitate the use, maintenance, and debugging of the system associated with the ARPA network and we expect that we may see more of this kind of diagnostic information passed around at all levels of the network.

2.5 Host Level Protocols

The ARPANET subnet of IMPs and communication paths, as described in the previous sections, provides the physical mechanism which allows the transport of messages from one location to another. The IMP/Host interface, both hardware and software, provides the mechanism which allows the submission of a message to the network by one Host and the reception of the message from the network by another Host. Nevertheless, this set of abilities is insufficient to insure the attainment of meaningful communication among computer programs in the various Hosts.

An illustrative analogy may be made with the telephone system; the existence of the switching system, connecting handsets in (almost) every home and office, is insufficient to insure meaningful communication between an English speaker and a French speaker, or even among several English speakers connected in a "conference call". There must be rules of discourse to insure both a common choice of language, and a common mode of interaction, to make communication meaningful.

In the ARPANET the rules of discourse which govern the exchange of information between two (generally) similar entities are denoted protocols. Protocols deal not only with the details of information format, the common language, but also with the framework (or model) within which information is exchanged. Of course, any pair (or larger group) of cooperating entities may agree to a particular set of rules of discourse, but the protocols of interest are those which were adopted by substantial segments of the ARPANET community. This is not so because the "standard" protocols are necessarily better, but because implementation of a standard protocol admits a Host to a large community of communicating entities, while implementation of pairwise non-standard protocols allows communication only with other entities which follow the same non-standard protocol.

Naturally, there have been situations within the ARPANET community where development of an ad-hoc protocol was the most advantageous course for a limited user group. These situations commonly arise from a desire for increased efficiency along some dimension (e.g., implementation size or transmission sped) and the solutions chosen frequently obtain an efficiency by giving up some measure of generality provided by the standard protocol. A few examples of such ad-hoc protocols which have been implemented within the ARPANET environment include:

- COPYNET, an interim protocol for file transfer between PDP-10 TENEX systems in 36-bit word format. The protocol was discontinued with the adoption of a standard file transfer protocol.

- A file transfer protocol adapted to achieve high bandwidth between two Univac 418-III computers at Tinker and McClellan Air Force Bases. The existing Univac record structures were transformed directly into ARPANET messages.

- A file transfer protocol adapted to achieve low memory requirements for TIP implementation; implemented on TIPs equipped with the magnetic tape option.

- A real-time message transfer protocol used between seismic data collection sites and a centralized seismic data processor.

- A real-time message transfer protocol used for the experimental transmission of digitized speech.

Nevertheless, the vast majority of traffic which has been carried by the ARPANET has been handled by the standard protocols.

The ARPANET Host level protocol design has followed a strategy of layering, a strategy which is similar to the levels-of-abstraction approach to program development. Each layer, or level, of Host protocol builds upon the lower layers. This strategy has several important potential advantages. First, each layer of protocol is assigned a clear-cut set of functions; designers of each layer are presented with a bounded set of objectives and are able to think about solutions within this bounded context. It is reasonable to assume that this makes the design process easier, and hence leads to better overall design. Second, each layer of protocol can present a straight-forward functional interface to the next higher layer. This means that each layer may be tested independent of higher layers, which should lead to easier and quicker implementation. It should also mean that a functional capability contained within one layer can be redesigned and/or re-implemented without complications arising in other layers.

On the other hand, there is a danger that too many layers may be created; if data must cross many interfaces, each requiring some transformation, on its path from the original producer to the ultimate consumer then the system as a whole may be extremely inefficient. There have been some examples in the

ARPANET experience of inefficiency resulting from the layered design; for an example see Section 2.5.5.1 which briefly discusses the official RJE protocol.

The layering structure adopted for the ARPANET Host level protocols is based upon the model of the way in which a typical time-shared computer might be structured. At the lowest level is a monitor to monitor (or operating system to operating system) protocol, which is denoted the Host-Host protocol. This layer is envisioned as handling communication issues such as multiplexing many independent communications over a single interface, or managing I/O buffers.

The next layer is denoted the Initial Connection Protocol and addresses the issue of getting a Host's monitor (operating system) to enter a new user into the current user set. This is, obviously, a transitory issue; the "Initial Connection Protocol" is similarly a transitory layer. It is a procedural protocol rather than a formatting protocol.

The third layer of Host level protocol deals with the method of terminal-to-Host (or user to application program) communication. The protocol is denoted "TELNET", a word derived from the phrase "Telecommunications Network". Originally it was envisioned that this protocol would be one of many "user level protocols" in the same layer, but as the system model of the structure of a time sharing system gained strength it began to seem appropriate to use this protocol as the "command path" building block for other protocols, and thus TELNET became a layer in its own right.

The fourth layer of protocol is the File Transfer Protocol. This is, as indicated by the name, the specification of a mechanism for the transfer of complete files, not a mechanism for retrieval of some information subset. The File Transfer Protocol (FTP) uses the TELNET protocol to govern the exchanges of commands and responses which specify the transfer, and uses the Host-Host protocol directly for the actual movement of the file.

At a still higher layer is the ARPANET official Remote Job Entry (RJE) protocol, which is designed to handle the submission of batch jobs and retrieval of results. This protocol uses TELNET to govern the exchange of job control commands and responses, and uses FTP to govern the actual transfer of job inputs and outputs. In actuality, there are probably few (or no) implementations of the "official" RJE protocol; most ARPANET batch job submissions use the ad hoc RJS (Remote Job Service) protocol specified by UCLA for use with the IBM 360/91 system located there.

Figure 2.5-1 below illustrates the layering of protocol graphically. The following subsections briefly describe each of the Host level layers in greater detail.

Figure 2.5-11 Protocol Layering

2.5.1 Host-Host Protocol

The ARPANET Host-Host protocol is the lowest layer of the Host level protocols. As noted above, it is envisioned as addressing those issues relevant to the Host system taken as a whole, that is, the issues of operating system to operating system communication.

2.5.1.1 Basic Concepts

The Host-Host protocol for the ARPANET was designed to address the following five basic issues.

- Multiplexing: It was envisioned that there would be many processes within a single Host all trying to communicate through the network simultaneously. The Host-Host protocol was required to provide a way of multiplexing these communications onto the single Host-to-network interface.

- Error Control: The Host-Host protocol designers had to choose the appropriate level of error control, both in terms of bit errors, and also in terms of lost, duplicated, or disordered messages.

- *Flow Control*: The Host-Host protocol should include mechanisms which would prevent a fast transmitter from overrunning the processing or buffering capacity of a slow receiver.

- *Out-of-Band Signalling*: It was anticipated that in any communication system with flow control there may be occasions when critical information is prohibited from being transmitted due to the flow control rules. In such cases an "out-of-band" (i.e., not subject to flow control) signalling system may be desirable, if not essential.

- *Resynchronization*: Two communicating systems may occasionally get out of step; with the "state information" describing the current situation different in the two systems due to hardware or software failure somewhere. The Host-Host protocol should provide a way of resynchronizing the two systems.

In addition to addressing these five basic issues, which can be seen recurring in a wide variety of protocol design situations, the ARPANET protocol designers were influenced by several other goals and concepts. First, the protocol was designed specifically for the ARPANET environment. In practical terms, the concentration on this environment had three significant effects on the protocol design, as follows:

1) The protocol designers accepted at face value the claims of the subnetwork implementers that the network would be error-free. Because of this, no bit or message error detection or recovery mechanisms were included in the protocol.

2) The IMP/Host interface specification left a few bits in the message leader (i.e., the fixed format portion of the message in which the message address is included) to be assigned by the sending Host; this was originally called the "link" field and is now a portion of what is called the "message ID" field. The Host-Host protocol design attempted to insure that no information which wouldn't fit into this field would be required on a per-message basis, in an effort to avoid adding overhead.

3) The IMP subnetwork was specified to deliver a special control message (the Ready for Next Message, or RFNM) to a message sender to serve both for flow control at the

Host-to-IMP level and to acknowledge that the message had been successfully transported across the subnetwork. The Host chose to treat this as a surrogate end-to-end acknowledgement.

Each of these instances of reliance upon the (actual or assumed) characteristics of the IMP subnetwork had a positive effect on the ease with which the protocol could be initially specified or implemented, but has had some long-range negative effects, as will be discussed below.

A second concept which had a strong influence on the protocol design was the model chosen as a framework for communication. The model chosen was the familiar system of circuit switched telephone communication, in an implementation of what have come to be called "virtual circuits" in other contexts; the entities are called "connections" in the ARPANET context. Thus, the ARPANET subnetwork, which "switches" individual messages, is used by the Host-Host protocols as a base upon which to establish circuits. (It is interesting to note that earlier in the history of computer communications significant effort has been invested in developing techniques for building message switching systems on a base consisting of circuits; one obvious example is the development of "multipoint" leased line management.) An alternative choice of model (e.g., the postal system) might have led to a completely different protocol. In fact, some experiments with a message-based Host-Host protocol were suggested fairly early in the ARPANET development, but none were actually implemented.

Third, given a model of communication which was based on "connections", the protocol design was strongly influenced by a desire for dynamic reconnection, or hand-off of one end of a connection from one processes to another, with the previous termination point and the new termination point (potentially) in different Hosts. As described later, this goal led to the necessity for the complex Initial Connection Protocol and influenced the specification of simplex, rather than full-duplex, connections. In retrospect, this seems to have been a high price to pay, since it appears that the potential for dynamic reconnection has never been used by any Host.

Another fairly basic idea which grew out of the connection-based model was the concept of a "socket". A socket is a particular implementation of a more general problem; the creation of a name space for processes which is global in scope, It was obvious that each individual Host would have some internal scheme for naming processes, but these schemes were anticipated to be at least different, and possibly even incompatible. Since it would be impractical to try to impose a common internal process naming scheme, an external name space was created, with a portion assigned to each Host; this name space consisted of sockets, which were specified by concatenating a Host address with a 32-bit "socket number". Obviously, the same socket number at two different Hosts represented two different processes. Each Host was given the responsibility for mapping internal process identifiers into local socket numbers.

It was envisioned by the protocol designers that there would be at least two methods of assigning socket numbers to processes. Some socket numbers would be assigned permanently to processes which requested them. For example, a service provided by the Host would need at least one socket which remained constant so that potential users would know how to address requests for service. A programmer might be assigned a group of sockets which could be used by his programs; the programmer would be able to convey these numbers to other programmers at other sites who might want to communicate with his programs. On the other hand, some socket numbers might be assigned by an operating system on a transitory basis to processes which wanted to send service requests into the network. A socket assigned in this way could be relinquished explicitly when the process had no further communication needs, or automatically when the process terminated. In fact, both methods of socket assignment are common in the ARPANET.

A connection is specified by naming the two sockets which it connects. However, carrying 64 bits of socket number in each message was deemed by the protocol designers to be an excessive amount of overhead. In order to reduce the amount of overhead, it was decided that use of a connection should be preceded by a setup phase (in fact, it is primarily this setup phase which differentiates a connection from a stream of messages) during which an "abbreviation" for the 64 bits of connection name would be established. As noted above, the IMP/Host interface specified a message leader which contained an 8-bit field, (originally) called the "link", which was to be specified by the sending Host and which would be delivered by the IMP subnetwork along with the message text. It was clear that if a mapping from 64 bit connection identification to 8-bit link were constrained to be unique at any instant, the link could serve as an abbreviation for the connection identification, and the connection identification would then require no new message overhead. Thus, the Host-Host protocol is closely wedded to the concept of an IMP/Host link.

Unfortunately, the concept of the IMP/Host link was originally created for purposes of flow control (although, as has been noted in a previous section, it was unsuitable for this purpose and eventually abandoned by the IMPs). The IMPs therefore imposed the rule that a Host could not have two messages with the same link number in the network at the same time. This being the case, the RFNM (or less frequently, the Incomplete Transmission response, which identified a lost message) could refer to a link number and uniquely identify a message. Thus, use of the link to identify a connection, and relying on the RFNM/Incomplete as the only message-level error control, meant that a Host-Host connection could never have more than one message in flight at a time, even after the IMP subnetwork removed this restriction. This has been an extremely important impediment to achieving good bandwidth performance between Hosts. Waiting for a RFNM between messages typically limits connection bandwidth to about 10Kbs, in contrast to the 40 Kbs or so which is usually available from the subnetwork.

Another consequence of the decision to use the IMP/Host link field in this way was the decision to make connections simplex, rather than full-duplex. Since the link field was used to identify the connection to which a message belonged, it had to be unique. Since it would be impossible to guarantee that when two Hosts established a connection they would be able to find any link number which was free at both ends, it was clear that each Host should choose the link number which it would use to identify the conversation. However, since the IMPs required that not more than a few tens of links be in use at one time, link numbers were viewed as a scarce resource, and thus allocating one to an unnecessary data path seemed wasteful. For example, a file transfer would typically need only a simplex connection.

Of course, if each connection must go through a setup phase before it exists, there must be a mechanism in each Host which can manage the setup phase and, further, these mechanisms must be able to communicate before the first connection is set up. The ARPANET Host-Host protocol envisions a process called the Network Control Program (NCP) running in each Host; this process manages the IMP/Host interface (this is the sense in which it "controls" the network) and also manages the Host-Host protocol. It is the NCP which performs connection setup. Messages sent from one NCP to another for control purposes are referred to as "control messages" and are identified by having the link number set to zero; thus, link zero is denoted the "control link." It is useful to think of the control link as implementing a control connection which is a permanent, rather than a "switched", connection.

2.5.1.2 Original Specification

The Host-Host protocol is specified in detail by the specification of the effects of the control commands exchanged between NCPs. The specific functions of the NCPs which are effected by these commands are the establishment and termination of connections, the control of the rate of data flow across established connections, and the transmission of out-of-band signals (interrupts) associated with existing connections. It should be noted that an NCP has at least two interfaces: one to the external world, which is specified by the Host-Host protocol, and a second to the processes, running in the Host, which the NCP is serving. This internal interface was considered a purely Host-specific issue, and the standard protocol made no attempt to standardize it. Nevertheless, it is possible to describe typical instances of this interface in general terms; such a description is given in Section 2.5.1.2.6.

2.5.1.2.1 Connection Establishment

Two commands are defined to establish a connection; these are STR (sender-to-receiver) and RTS (receiver-to-sender), each of which takes three parameters:

STR (send socket, receive socket, byte size)

RTS (receive socket, send socket, link)

The STR command is sent from a prospective sender to a prospective receiver, and the RTS from a prospective receiver to a prospective sender. The send socket field names a socket local to the prospective sender; the receive socket field names a socket local to the prospective receiver. In the STR command, the "size" field contains an unsigned binary number (in the range 1 to 255; zero is prohibited) specifying the byte size to be used for all messages over the connection. In the RTS command, the "link" field specifies a link number; all messages over the connection must be sent over the link specified by this number. These two commands are referred to as requests-for-connection (RFCs). An STR and an RTS match if the receive socket fields match and the send socket fields match. A connection is established when a matching pair of RFCs have been exchanged.

With respect to a particular connection, the Host containing the end socket is called the "sending Host" and the Host containing the receive socket is called the "receiving Host". A Host may connect one of its receive sockets to one of its send sockets, thus becoming both the sending Host and the receiving Host for that connection. These terms apply only to data flow; control messages will, in general, be transmitted in both directions.

Hosts are prohibited from establishing more than one connection to any local socket. This restriction has led to rather severe difficulties in establishing contact with a general service, as exemplified by the Initial Connection Protocol (see Section 2.5.2). It has been argued that in general only one socket of the pair which specify a connection need be unique in order to avoid confusion; the stronger restriction stated for the ARPANET Host-Host protocol is a result of the desire for a dynamic reconnection mechanism which "always" works. As an example, consider the case in which sockets x and y in Host A were both connected to socket z in Host B. This situation causes no ambiguity. However, if all of the connections emanating from Host A needed to be temporarily reconnected to some new location, say socket w at Host C, then there would be two connections from w (at C) to z (at B); it would no longer be possible to distinguish them. The problem arises whenever it is necessary to reconnect connections terminating at two general service process, each of which "assumes" that the Host at the other end will take care to keep its end unique. To avoid ever making such an assumption, the stronger condition is imposed.

A Host sends an RFC either to request a connection, or to accept a foreign Host's request. Since RFC commands are used both for requesting and for accepting the establishment of a connection, it is possible for either of two cooperating processes to initiate connection establishment. As a consequence, a family of processes may be created with connection-initiating actions built-in, and the processes within this family may be started up (in different Hosts) in arbitrary order provided that appropriate queueing is performed by the Hosts involved.

There is no prescribed lifetime for an RFC. A Host is permitted to queue incoming RFCs and withhold a response for an arbitrarily long time, or, alternatively, to reject requests (see Connection Termination below) immediately if it does not have a matching RFC outstanding. It may be reasonable, for example, for an NCP to queue an RFC that refers to some currently unused socket until a local process takes control of that socket number and tells the NCP to accept or reject the request. Of course, the Host which sent the RFC may be unwilling to wait for an arbitrarily long time, so it may abort the request. On the other hand, some NCP implementations may not include any space for queueing RFCs, and thus can be expected to reject RFCs unless the RFC sequence was initiated locally.

The decision to queue, or not queue, incoming RFCs has important implications. Each RFC which is queued, of course, requires a small amount of memory in the Host doing the queueing. If each incoming RFC is queued until a local process seizes the local socket and accepts (or rejects) the RFC, but no local process ever seizes the socket, the RFC must be queued "forever." Theoretically this could occur infinitely many times (there is no reason not to queue several RFCs for a single local socket, letting the local process decide which, if any, to accept) thus requiring infinite storage for the RFC queue. On the other hand, if no queueing is performed cooperating processes will be able to establish a desired connection only by accident (when they are started up such that one issues its RFC while the RFC of the other is in transit in the network - clearly an unlikely occurrence).

Perhaps the most reasonable solution to these problems is for each NCP to give processes running in its own Host two options for attempting to initiate connections. The first option would allow a process to cause an RFC to be sent to a specified remote socket; with the NCP notifying the process as to whether the RFC were accepted or rejected by the remote Host. The second option would allow a process to tell its own NCP to "listen" for an RFC to a specified local socket from some remote socket (the process might also specify the particular remote socket and/or Host it wishes to communicate with) and to accept the RFC (i.e., return a matching RFC) if and when it arrives. Note that this also involves queueing (of "listen" requests), but it is internal queueing which is susceptible to reasonable management by the local Host. If this implementation were available, one of two cooperating processes could "listen" while the other process caused a series of RFCs to be sent to the "listening" socket until one was accepted. Thus, no queueing of incoming RFCs would be required, although it would do no harm.

2.3.1.2.2 Connection Termination

The command used to terminate a connection is CLS (close), which takes two parameters:

CLS (my socket, your socket)

The "my socket" field contains the socket local to the sender of the CLS command. The "your socket" field contains the socket local to the receiver of the CLS command. Each side must send and receive a CLS command before connection termination is completed and the sockets are free to participate in other connections.

It is not necessary for a connection to be established (i.e., for both RFCs to be exchanged) before connection termination begins. For example, if a Host wishes to refuse a request for connection, it sends back a CLS instead of a matching RFC. The refusing Host then waits for the initiating Host to acknowledge the refusal by returning a CLS. Similarly, if a Host wishes to abort its outstanding request for a connection, it sends a CLS command. The foreign Host is obliged to acknowledge the CLS with its own CLS; even though the connection was never established, CLS commands must be exchanged before the sockets are free for other use.

After a connection is established, CLS commands sent by the receiver and sender have slightly different effects. CLS commands sent by the sender indicate that no more messages will be sent over the connection; thus the CLS must not be sent if there is a message in transit over the connection. A CLS command sent by the receiver acts as a demand on the sender to terminate transmission. However, since there is a delay in getting the CLS command to the sender, the receiver must expect more input.

A Host should "quickly" acknowledge an incoming CLS so the foreign Host can purge its tables. However, there is no prescribed time period in which a CLS must be acknowledged.

Because the CLS command is used both to initiate closing, aborting and refusing a connection, and to acknowledge closing, aborting and refusing a connection, race conditions can occur. However, they do not lead to ambiguous or erroneous results, as illustrated in the following examples:

EXAMPLE 1: Suppose that Host A sends B a request for connection, and then A sends a CLS to Host B because it is tired of waiting for a reply. However, just when A sends its CLS to B, B sends a CLS to A to refuse the connection. A will "believe" B is acknowledging the abort, and B will "believe" A is acknowledging its refusal, but the outcome will be correct.

EXAMPLE 2: Suppose that Host A sends Host B an RFC followed by a CLS as in example 1. In this case, however, B sends a matching RFC to A just when A sends its CLS. Host A may "believe" that the RFC is an attempt (on the part of B) to establish a new connection or may understand the race condition; in either case it can discard the RFC since its socket is not yet free. Host B will "believe" that the CLS is breaking an established connection, but the outcome is correct since a matching CLS is the required response, and both A and B will then terminate the connection.

Every NCP implementation is faced with the problem of what to do if a matching CLS is not returned "quickly" by a foreign Host (i.e., if the foreign Host appears to be violating protocol in this respect). One naive answer is to hold the connection in a partially closed state "forever" waiting for a matching CLS. There are two difficulties with this solution. First, the socket involved may be a "scarce resource" such as the "logger" socket specified by an Initial Connection Protocol (see Section 2.5.2) which the local Host cannot afford to tie up indefinitely. Second, a partially broken (or malicious) process in a foreign Host may send an unending stream of RFCs which the local Host wishes to refuse.

Most implementers, recognizing these problems, have adopted some unofficial timeout period after which they "forget" a connection even if a matching CLS has not been received. It has been suggested that the network adopt some standard timeout period, but it has been proven impossible to arrive at a period which is both short enough to be useful and long enough to be acceptable to every Host. The difficulty with any timeout is that if a second connection between the same pair of sockets is later established, and a CLS finally arrives for the first connection, the second connection is likely to be closed. This situation can only arise, however, if one Host violates protocol in two ways; first by failing to respond quickly to an incoming CLS, and second by permitting establishment of a connection involving a socket which it believes is already in use.

2.5.1.2.3 Flow Control

After a connection is established, a sending Host sends messages over the agreed-upon link to the receiving Host. The receiving NCP accepts messages from its IMP and queues them for its various processes. Since it may happen that the messages arrive faster than they can be processed, some mechanism is required which permits the receiving Host to quench the flows from the sending Hosts.

The flow control mechanism requires the receiving Host to allocate buffer space for each connection and to notify the sending Host of how much space is available. The sending Host keeps track of how much room is available and never sends more data than it believes the receiving Host can accept.

To implement this mechanism, the sending Host keeps a bit counter associated with each connection. The counter is initialized to zero when the connection is established and is increased by allocate (ALL) control commands sent from the receiving Host as described below. When sending a message, the NCP of the sending Host subtracts the "text length", i.e. the product of the connection byte size and the number of data bytes in the message, from the bit counter. The sender is prohibited from sending if the counter would be decremented below zero.

The flow control mechanism does not pertain to the control link, since connections are never explicitly established over this link.

The control command used to increase the sender's bit counter was ALL (allocate), which took two parameters:

ALL (link, bit space)

This command is sent only from the receiving Host to the sending Host, and is legal only when a connection using the link number appearing in the "link" field is established. The "bit space" field is defined to be an unsigned binary integer specifying the amount by which the sender's bit counter is to be incremented. The receiver is prohibited from incrementing the sender's counter above 2 to the 32nd minus 1. In general, this rule will require the receiver to maintain a counter which is incremented and decremented according to the same rules as the sender's counter.

2.5.1.2.4 Out-of-Band Signalling (Interrupt)

The Host-Host protocol includes a mechanism by which the transmission over a connection may be "interrupted." The meaning of the "interrupt" is not defined at this level, but is made available for use by outer layers of protocol. The interrupt command sent from the receiving Host to the sending Host is INR (interrupt-by-receiver).

INR (link)

The interrupt command sent from the sending Host to the receiving Host is INS (interrupt-by-sender).

INS (link)

The INR and INS commands are legal only when a connection using the link number in the "link" field is established.

2.5.1.2.5 Other Functions

In spite of the effort made to insure that the Host-Host protocol functions did not require any additional bits to be transmitted on a per-message basis (using instead the IMP-Host protocol "link" field), it seemed clear that many Hosts might have difficulty beginning a message immediately following the 32-bit IMP/Host leader. For example, a PDP-10 Host, with a 36-bit word size, would find it much more convenient to begin the actual text of a message at a 36-bit boundary; beginning at a 32-bit boundary would be likely to require extensive expensive bit shifting. Therefore, the protocol defined a "marking" field to be included in each message between the IMP/Host leader and the first meaningful data bit. The marking consists of zero or more zero bits followed by a single one bit. Thus, the receiving Host NCP begins at bit 33 of a received message searching for a one bit. The bit following this one bit is the beginning of the meaningful data.

It is sometimes useful for one Host to determine if some other Host is capable of carrying on network conversations. The control command to be used for this purpose is ECO (echo).

ECO (data)

The "data" field may contain any bit configuration chosen by the Host sending the ECO. Upon receiving an ECO command an NCP must respond by returning the data to the sender in an ERP (Echo-reply) command.

ERP (data)

A Host should "quickly" respond (with an ERP command) to an incoming ECO command. However, there is no prescribed time period, after the receipt of an ECO, in which the ERP must be returned. A Host is prohibited from sending an ERP when no ECO has been received, or from sending an ECO to a Host while a previous ECO to that Host remains unanswered.

2.5.1.2.6 NCP/Process Interface

As previously noted, the interface between an NCP and the processes which use it within the same Host was considered to be an internal matter, not a subject for network-wide standardization. Nevertheless, the design of the Host-Host protocol assumed that certain facilities would be available to the processes which used the NCP, and these were documented in terms of a set of hypothetical system calls which NCP implementers might provide.

Of course, the type of interface which the NCP can offer is strongly influenced by the facilities supported by the Host operating system. An operating system consists of program modules which augment the hardware and provide an environment for processes. The interfaces between these operating system modules and user processes take the form of system calls and returns, and sometimes pseudo interrupts. System calls are implemented in a variety of ways, but often it is a special hardware instruction that involves a system call (e.g., SVC, UUO, JSYS, MME). In higher level programming languages a system call is often indistinguishable from a subroutine call. In some cases the form of the system call is different for each module of the system.

A process is a program in execution with its associated address space, a location counter, some general registers, and usually some open files (or devices). Processes may be created by users, though there are often processes which have been created as part of the system for particular functions, and some of these may be initialized by the system when it begins running. A process may have several entry points, each of these is called a Port. Special Ports which are designed in processes to handle unusual and error control conditions are called "code" returns.

One important aspect of an operating system that has a great impact on the implementation of network functional capabilities is the provision for interprocess communication. Generally processes are viewed as independent computational units that need interact only with the operating system in a few very constrained ways (i.e., via system calls). However, often it would be useful to build a new capability based on a combination of existing programs. One way of extending the usefulness of the process structure is to allow processes to communicate among themselves such that several processes may cooperate to accomplish a computational goal. The form of communication supported by an operating system very much influences the extent to which processes actually cooperate and, therefore, the extent to which use of the ARPANET is a natural extension of the programming environment.

A basic set of system calls which should be available to user written processes for use of the network via the Host-Host protocol are:

LISTEN (port, socket, code): This is a call to the NCP indicating that the process is willing to connect to any process which sends a request for connection (RFC) specifying this local socket. This requests the NCP to associate a local socket with a process and the specified port in that process. A return value is given in "code". If there is a pending call, the connection may be opened immediately and the process notified; if there is no matching pending call, the NCP will notify the process when a matching RFC arrives.

CONNECT (port, socket, foreign socket, code): The local socket of this process is associated with this call. The local process requests the NCP to initiate a connection on its behalf to a particular foreign socket from the specified local socket and associate the connection with the specified port. If there is a pending call matching these parameters, the connection is opened and the process so notified. A return value is given in "code". If there is no pending matching call, the NCP communicates this request to the foreign host and notifies the local process when a matching request is received and the connection is opened.

SEND (port, buffer, length, code): The local process requests the NCP to transmit the data starting at "buffer" and extending "length" bits over the connection associated with this port in accordance with the allocation values. "Code" is set with a return value.

RECEIVE (port, buffer, length, code): A local process requests the NCP to store data received on the connection associated with this port into the processes address space starting at "buffer" and extending for "length" bits. A return value is set in "code".

CLOSE (port, code): Activity on the connection associated with this port is stopped. A return value is set in "code".

INTERRUPT (port, code): The local process requests the NCP to send a special interrupt signal referring to the connection associated with this port outside the normal flow control constraints. A return value is set in "code".

STATUS (port, info, code): This is a call by a local process requesting the NCP to obtain the relevant status information from the connection table entry associated with the port and place it in the space specified by "info". A return value is set in "code". This allows a user program to monitor the state of a connection, of special interest are the flow control allocation values.

2.5.1.3 Subsequent Modifications

Within a few months of the specification of the Host-Host protocol, as the first few implementations began to take form, several shortcomings or inefficiencies came to light. A revised version of the protocol was developed to correct for these shortcomings; it is this revised version which has survived as the standard Host-Host protocol. The modifications made were in the areas of flow control, NCP synchronization, and text alignment ("marking"); each of which is described briefly.

Flow control changes: The early implementers identified two types of difficulty with the flow control mechanism. First, some Hosts with limited capabilities wished to allocate only a single buffer to each network connection, and to avoid the bit-packing which would be needed if multiple short messages (whose total length was less than the number of bits in the buffer) arrived. The solution adopted was to add a message count to the ALL command. This allows a Host to restrict the total number of bits it is willing to receive, the total number of messages, or both. Thus, the form of the ALL command is now

ALL (link, message space, bit space)

A second criticism of the flow control scheme is that it requires a Host to explicitly divide its total buffer space among all the connections which it may possibly participate in. It was argued that it would be better if a Host could assign its total buffer space to the set of existing connections (or even "overbook" its allocations), retracting some space allocation for reassignment if a new connection is established or, in the case of overbooking, if the space is being used up more rapidly than expected. To accomplish the retraction of allocation, two new commands were defined. The control command used to request that the sending Host return all or part of its current allocation is GVB (give-back), which takes three parameters:

GVB (link, message fraction, bit fraction)

This command is sent only from the receiving Host to the sending Host, and is legal only when a connection using the link number in the "link" field is established. The message fraction and bit fraction fields are defined as the fraction (in 128ths) of the current message space allocation and bit space allocation (respectively) to be returned. If either of the fractions is equal to or greater than one, all of the corresponding allocation must be returned. Fractions are used since, with messages in transit, the sender and receiver may not agree on the actual allocation at every point in time.

Upon receiving a GVB command, the sending Host must return at least the requested portions of the message and bit space allocations. A sending Host is prohibited from spontaneously returning portions of the allocations. The control command for performing this function is RET (return), which takes three parameters:

RET (link, message space, bit space)

This command is sent only from the sending Host to the receiving Host, and is legal only when a connection using the link number in the "link" field is established and a GVB command has been received from the receiving Host. The "message space" field and the "bit space" field are defined as unsigned binary integers specifying the amounts by which the sender's message counter and bit counter (respectively) have been decremented due to the RET activity (i.e., the amounts of message and bit space allocation being returned). NCPs are obliged to answer a GVB with a RET "quickly"; however, there is no prescribed time period in which the answering RET must be sent.

It is interesting to note that no Hosts actually adopted a buffer assignment strategy which requires the use of the GVB.

NCP Synchronization: Occasionally, due to lost control messages, system "crashes", NCP errors, the addition of new network Hosts, or other factors, communication between two NCPs will need to be synchronized. One possible effect of any disruption might be that neither of the involved NCPs could be sure that its stored information regarding connections with the other Host matched the information stored by the NCP of the other Host. In this situation, an NCP may wish to reinitialize its tables and request that the other Host do likewise; for this purpose the pair of control commands RST (reset) and RRP (reset-reply), neither of which take parameters, were added to the protocol.

The RST command is to be interpreted by the Host receiving it as a signal to purge its NCP tables of any entries which arose from communication with the Host which sent the RST. The Host sending the RST should likewise purge its NCP tables of any entries which arise from communication with the Host to which the RST was sent. The Host receiving the RST should acknowledge receipt by returning an RRP. Once the first Host has sent an RST to the second Host, the first Host is not obliged to communicate with the second Host (except for responding to RST) until the second Host returns an RRP. In fact, to avoid synchronization errors, the first Host should not communicate with the second until the RST is answered. If both NCPs decide to send RSTs at approximately the same time, then each Host will receive an RST and each must answer with an RRP, even though its own RST has not yet been answered.

Hosts are prohibited from sending an RRP when no RST has been received. Further, Hosts may send only one RST in a single control message and should wait a "reasonable time" before sending another RST to the same Host. Under these conditions, a single RRP constitutes an "answer" to all RSTs sent to that Host, and any other RRPs arriving from that Host should be discarded.

Text Alignment: It was quickly recognized that the marking convention adopted for text alignment completely solved the sender's problems, but didn't help the receiver at all. In view of the fact that all of the Hosts connected (or anticipated to be connected) to the ARPANET could conveniently handle units of 8, 18, or 36 bits, it was decided that all messages should begin with a "message header" of 72 bits (the least common multiple of these lengths) including the IMP leader. Since this adds 40 bits to the length of every message, it was decided to specify that 8 bits are used to restate the number of bits per byte (as established during connection setup) and 16 bits are used to specify the number of bytes in the message. These fields are used to locate the last bit of the text.

The protocol as originally specified, with the modifications described above, has been the "official" standard Host-Host protocol for the ARPANET since early 1971. Because of the large number of different implementations, and the tremendous amount of labor that would be required to change them (not to mention the coordination problems), the protocol has been highly resistant to further modification. However, there has been one set of ad hoc additions to the protocol made by many of the Hosts, notably all of the TIPs and TENEXes, since the additions were defined in early 1974.

The additions to the protocol were designed to deal with the lack of robustness in the flow control mechanism, and the lack of a protocol mechanism for resynchronizing individual connections. The problem perceived to exist with the incremental flow control mechanism is that it presumes a completely error-free transmission medium. However, low frequency Host software bugs, intermittent hardware bugs, and network failures can cause an ALL control command to be lost, with the result that the buffer space accounting at sender and receiver gets out of synchronization. If this creates a situation in which the receiver believes buffers are available, but the sender believes all buffers have been used, no further data can flow.

Use of the Host-Host RST (reset) command is inappropriate here, as it destroys all connections between the two Hosts. What is needed is a way to resynchronize only the affected connection without disturbing any others.

A second troublesome symptom of inconsistency in status information is the "half-closed" connection; after a service interruption or network partition, one NCP may believe that a connection is still open, while the other believes that the connection is closed (does not exist). When such an inconsistency is discovered, the "open" end of the connection should be closed.

To achieve resynchronization of space allocation, three new ad hoc control commands are defined: RAR (Reset Allocation by Receiver), RAS (Reset Allocation by Sender), and RAP (Reset Allocation Please), each of which takes one parameter, the link number for the connection.

The RAS command is sent from the Host sending on "link" to the Host receiving on "link". This command may be sent whenever the sending Host desires to resynchronize the status information associated with the connection and doesn't have a message in transit through the network.

Some circumstances in which the sending Host may choose to do this are:

1) After a timeout when there is traffic to move but no allocation (assumes that an allocation has been lost);

2) When an inconsistent event occurs associated with that connection (e.g., an ALL is received which increases the message or bit space allocation beyond the allowed maximum).

3) After the sending host has suffered an interruption of network service;

4) In response to a RAP (see below).

The RAR command is sent from the Host receiving on "link" to the Host sending on "link" in response to a RAS. It marks the completion of the connection resynchronization. When the RAR is returned the connection is in the known state of having no messages in transit and the allocations set to zero. The receiving Host may then start with a new allocation and normal message transmission can proceed. Since the RAR may be sent only in response to a RAS, there are no races in the resynchronization. All of the initiative lies with the sending Host.

If the receiving Host detects an anomalous situation, however, a way is needed to inform the sending Host that a resynchronization is desirable. For this purpose, the RAP command is provided. It constitutes a "suggestion" on the part of the receiving Host that the sending Host resynchronize; the sending Host is free to honor it or not as it sees fit. Since there is no obligatory response to a RAP, the receiving Host may send them as frequently as it chooses and no harm can occur. For example, if a message in excess of the allocate arrives, the receiving Host might send RAPs every few seconds until the sending Host replies, with no fears of races if one or more RAPs pass a RAS in the network.

An interruption of communication may result from a partitioning of the subnet or from a service interruption on one of the communicating Hosts. It is undesirable to tie up resources indefinitely under such circumstances, so the user is provided with the option of freeing up these resources (including himself) by unilaterally dissolving the connection. Here "unilaterally" means sending the CLS command and closing the connection without receiving the CLS acknowledgement.

When service is restored after such an interruption, the status information at the two ends of the connection may be out of synchronization. One end believes that the connection is open and may proceed to use the connection. The disconnecting end believes that the connection is closed (does not exist), and may proceed to re-initialize communication by opening a new connection using the same socket pair or same link.

The resynchronization needed here is to properly close the open end of the connection when the inconsistency is detected.

This can be accomplished by specifying consistency checks and adding a new pair of commands. The two new ad hoc commands are NXR (Non-existent Receive link) and NXS (Non-existent Send link), each of which takes as a parameter the link number.

The "missing CLS" situation described above can manifest itself in two ways. The first way involves action taken by the NCP at the "open" end of the connection. It may continue to send regular messages on the link of the half-closed connection, or control messages referencing its link. The closed end should respond with an NXS if the message referred to a non-existent transmit link (e.g., was an ALL) or NXR if the message referred to a non-existent receive link (e.g., a data message). On receipt of such an NXS or NXR message, the NCP at the "open" end should close the connection by modifying its tables (without sending any CLS command) thereby bringing both ends into agreement.

A second way this inconsistency can show up involves actions initiated by the NCP at the "closed" end. It may (thinking the connection is closed) send an RFC to open a connection using the same socket or link. The NCP at the "open" end should detect the inconsistency when it receives such an RTS or STR command. In this case, the NCP at the "open" end should close the connection (without sending any CLS command) to bring the two ends into agreement before responding to the RFC.

2.5.1.4 Conclusions

Since the definition of the current Host-Host protocol in early 1971, literally billions of bits have been successfully transferred between dissimilar ARPANET Hosts under its rules. Nevertheless, a number of problem areas have been identified during the intervening years, and subsequent protocol work in other ARPA programs, in other research networks, and in standards organizations have attempted to find solutions to many of these problems. There have also occasionally been suggestions to revise the ARPANET Host-Host protocol, but because of the large number of implementations which would be affected these have never met with widespread enthusiasm.

Some of the problem areas have been discussed in the previous paragraphs; these include the decision to make connections simplex (because of a "scarcity" of link numbers), the lack of robustness in the incremental allocation scheme, which has been addressed by some Hosts with an ad hoc synchronization scheme, and the need to resynchronize NCP data for single connections. The discussion of the Initial Connection Protocol in Section 2.5.2 will illustrate the class of difficulties which resulted from the decision to allow only one connection to a socket, in order to facilitate dynamic reconnection. Some of the other shortcomings are mentioned below.

The assumption that the ARPANET subnetwork provides a completely error-free transmission medium (i.e., not that it would never lose a message, but that it would always accurately return a RFNM or an Incomplete Transmission response) led to the design of a protocol that is not robust. Even if the assumption were true, there are still occasional errors in the IMP/Host interface and in the Host software which require correction. There is not any requirement for extensive error detection and correction at the Host level, but errors will occur and the protocol should be able to survive them. As discussed in the previous section, it is frequently unable to do so.

On the other hand, some of the more esoteric expected communication scenarios have never actually occurred, and the protocol complexities designed to support them thus have gone unused. In particular, the network usage has involved little or no dynamic reconnection, and thus the protocol restrictions designed to make it possible have proven unnecessary. It is probable that one reason why the intense initial interest in reconnection has not given birth to its embodiment in working systems is at least partly an accounting/authentication problem; no Host wants to provide service unless its accounting system can tell where to send the bill. This generally means, for existing Hosts, engaging in a dialog with the "logger" process. Development of a networkwide accounting system appears likely to be a prerequisite for dynamic movement of the locus of processing from Host to Host.

Another unimplemented complexity is the type of dynamic buffer allocation anticipation by the GVB/RET mechanism. In general, large Hosts have been able to dedicate enough buffering to each connection so that GVBs are unnecessary. Conversely, the smallest Hosts, which might best profit from dynamic buffer allocation, have found that the memory space (and processing time) for the code needed to operate a dynamic buffering scheme is greater than the anticipated average gain. In addition, "overbooking" schemes must be able to generate predictions of overallocation soon enough to account for the transmission and processing delay which will be incurred by a GVB request. Thus, each Host contains the code to process a GVB, but none has the code to generate one.

A major difficulty with the Host-Host protocol is its reliance on the IMP/Host RFNM as a surrogate end-to-end acknowledgement. First, this reliance violates what subsequent workers in the protocol field have called "the principle of thin-wire design". The point is that if a protocol is designed to work over a "thin wire", it can be adapted to other communications media, but if the protocol relies on the generation of auxiliary information (e.g., the RFNM) within the communication medium then it cannot easily be adapted to use a different medium. This locks the Host into use of a particular technology and implementation, an undesirable situation both technically and economically.

As previously mentioned, the subnetwork responses which the Host needed (RFNM or Incomplete Transmission) were uniquely identified by the link. Since the IMPs initially prohibited more than one message outstanding with a given link number, this restriction was carried into the Host protocol. However, when the IMPs discontinued this prohibition, the Hosts' use of the link number as the exclusive identifier of which connection a message belongs to made it impossible to relax the Host-Host protocol restriction. To illustrate this assertion, suppose that Host A sends two messages on link n to Host B. At some later time a RFNM(n) and an Incomplete(n) arrive at A, and a single message on link n arrives at B. Host A cannot decide which of the two messages should be retransmitted, and Host B cannot decide if the message just received immediately follows previous messages for the connection or if there is a gap. The Host-Host protocol restriction to a single outstanding message (per connection), as previously noted, restricts typical Host to Host bandwidth to about 10 Kbs, while the subnetwork can frequently deliver up to about 40 Kbs.

Another difficulty, already briefly mentioned in Section 2.5.1.2.2, is the fact that every NCP implementer is faced with the necessity for "timing out" certain protocol states in order to free valuable resources when a correspondent Host seems to be violating protocol, but the timeouts are not uniform between Hosts. This situation sometimes leads to confusion when two heavily loaded Hosts are attempting to communicate, with one "timing out" states, and the other not doing so.

A surprising omission in the design of a protocol within a research community is the lack of specification of any instrumentation of Network Control Programs. It is also somewhat surprising that almost no NCPs have been instrumented, even if no instrumentation was specified. But in fact it is generally impossible to obtain answers to questions such as:

- What are the statistics about the number of connections open at one time?

- What are the statistics about the number of messages sent or received over a connection?

- What amounts of bit and message space are typically allocated, and how is it used up?

- What is the ratio of control messages to data messages?

Finally, a very significant characteristic of the Host-Host protocol is that it is based on the concept of logical line switching in contrast to logical message switching. This has profound implications for its simplicity, flexibility, and robustness. A way to understand these implications is to consider the alternative message-switching approach to Host-Host protocol design. As an aid to visualizing the approach, one of the several specific proposals which has been made as an ARPANET alternative protocol is sketched briefly here.

Suppose that two processes, such as P and Q in Figure 2.5-2, wish to communicate; in particular, suppose P has a message to send to Q. Rather than setting up a connection (and its attendent flow control mechanism) with which to pass messages from P to Q, for each message Q is willing to receive, it sends a RECEIVE control message to Host A. When P has a message to send, it sends the data to its NCP. In whichever order these messages (RECEIVE or data) reach the Host A NCP, they eventually rendezvous; and, at this time, the data is sent to Host B and the relevant entries in the table in the Host A NCP are cleared. When the data gets to Host B, it matches a table entry left when the RECEIVE passed out of Host B. The table entries in Host B are then cleared and the data is passed to process Q.

The significant features of this is that connections are not maintained over a sequence of messages but instead are set up and expire on a message-by-message basis. There are several advantages to this approach:

1. There is no need for a large number of inter-NCP control messages to set up and break connections and consequently no need for a special control channel.

Figure 2.5-2 An Alternative message-switching approach

2. The RECEIVING process has the opportunity after each message to stop the flow of messages. Further, since the RECEIVE process itself can be required to provide a buffer, the NCP is relieved of the task of providing a large buffering capacity.

3. Because connections exist only fleetingly, relatively complex operations such as dynamically switching "connections" between a variety of processes are easy.

4. Errors have a minimal effect. For instance, if a RECEIVE message is lost, the RECEIVING NCP will time it out and the process can do another RECEIVE just as is normally done when the sender declines to send in response to a RECEIVE.

5. This message-switching protocol is suitable for implementation on small computers as well as large computers owing to its simplicity and the fact that there is no need for a large buffering capacity in the monitor.

2.5.2 Initial Connection Protocol

The Initial Connection Protocol (ICP) is a procedural protocol, rather than a formatting protocol, and applies only at the instant when a user attempts to make contact with a general service such as the "monitor" (or operating system) of a timesharing terminal-oriented system.

The ICP owes its existence to the protocol restriction that permits a socket to be involved in only one connection at a time, coupled with the necessity of being able to "advertise" the socket number at which a service (such as entry to a time-sharing system) may be found. Once the first user establishes a connection to the advertised socket, subsequent users will have to be turned away until the first user breaks the connection. Of course, the service-providing Host could advertise a long list of socket numbers, and allow a user to work down the list until a free socket was found, but if the service-providing Host hopes to derive some benefit (e.g., income or popularity) from each user, turning a potential user away may represent lost opportunity.

An analogy exists with the situation facing a business organization which makes sales via incoming telephone calls. The organization typically advertises a single telephone number, However, the organization procures many incoming telephone circuits, together with some type of automatic "ring up" equipment, so that if a call is made to a number which is already in use, the call will be switched (in a way invisible to the caller) to a free line. Naturally, at some point the capacity of the organization to handle more calls will be exceeded, but only at this point will the caller receive a connection refusal (i.e., a busy signal).

The ICP provides a function similar, although not identical, to the "ring up" device. The switching cannot be completely invisible to the calling Host, since that Host must know the actual socket number involved in the connection. Therefore the ICP specifies the following procedure to be followed by the server (called) and user (calling) Hosts (refer to Figure 2.5-3).

A server process makes available a well-advertised send-socket L and listens for a user (step S1). A user process initiates connection to sendsocket L from its receive-socket U specifying link "a" (U1). When the server receives an RTS at socket L, it confirms the connection with an STR from send-socket L to receive-socket U with a byte size of 32 bits (S2). The server then waits for an allocation from the user (S3). When the

<u>SERVER</u>		<u>USER</u>	
S1:	Listen on socket L	U1:	RTS (U, L, a)
S2:	STR (L, U, 32)	U2:	Wait for match
S3:	Wait for allocation	U3:	ALL (a, m, b)
S4:	Send as data the socket number S	U4:	Receive the socket number S as data
S5:	CLS (L, U)	U5:	CLS (U, L)
S6:	RTS (S, U+3, x)	U6:	STR (U+3, S, n1)
S7:	STR (S+1, U+2, n1)	U7:	RTS (U+2, S+1, y)

Figure 2.5-3: Steps in Initial Connection Protocol

user receives confirmation of the connection (U2), it allocates m (messages) and b (bits) for the connection using link "a" (U3). The server receives the allocation and sends, as regular data over the connection, the socket number S, which occupies one 32-bit connection byte (S4). The server can then initiate the closing of the connection between L and U (S5). Once the user system has received the data message containing S (U4) it can confirm (or initiate) the closing of the connection between L and U (U5).

The essence of the portion of the ICP described so far is to establish contact between the user and the advertised socket just long enough for the server to select an arbitrary free socket and convey its number to the user, then break the connection. Since this should be able to happen rapidly, a small amount of queueing in the server should be sufficient to hold any calls which come in for the advertised socket from other users during the connection's lifetime. The user has also received some assurance that the server is interested in establishing contact, as evidenced by the willingness of the server to provide a "callback" socket.

Now, concurrently with the closing of the connection between L and U, the user and server exchange commands opening the connection from socket U+3 at the user to socket S at the server using link "x" and byte size n1 (S6 and U6). Also concurrently, commands are exchanged opening a connection from server socket S+1 to user socket U+2 (S7 and U7).

As can be seen from the above discussion, the Initial Connection Protocol is relatively complex. On the other hand, implementations are typically not very large, and tens or hundreds of thousands of connections have been established using ity; it clearly is not too complex to be useful. However, most modern Host-Host protocol designs obviate the need for an ICP by requiring only one end of a connection to be unique. In view of the experience in ARPANET with dynamic reconnection, namely that there has been little or none, this is probably the proper course for protocol development.

2.5.3 TELNET Protocol

The TELNET protocol deals with the method of carrying out terminal-to-Host (or user to application program) communication. TELNET is derived from the phrase "Telecommunications Network". Since most of the Hosts initially connected to ARPANET provided some type of interactive, terminal-oriented service, there was intense interest in defining a protocol which would allow a human user, sitting at a keyboard/display (CRT or character printer), to access a remote Host as though the terminal and the Host were directly connected; TELNET is the result.

2.5.3.1 Basic Concepts

The original TELNET protocol is built around the ideas of a Network Virtual Terminal (NVT), a dichotomy between users and servers, and a set of TELNET control signals.

The concept of a Network Virtual Terminal is an attempt to reduce the mapping problems raised by interconnecting a variety of terminals to a variety of Hosts. Each different type of Host typically has a different internal character set and operating procedures. Similarly, each different terminal (usually even "compatible" terminals) has its own character set and/or timing and operating constraints. If there are "H" Hosts and "T" terminals, and they are all directly connected to each other, then there are H*T total connections, and hence H*T total mappings required.

Of course, in the ARPANET terminals are not connected directly to packet switches; terminals are connected to Hosts (Sometimes mini-Hosts such as the terminal-support portion of the TIP, but Hosts nonetheless). Thus, a terminal gains access to a remote Host via the mediating intelligence of a local Host process. The Virtual Terminal concept has the local process map from the terminal-specific characteristics into a network standard. Each Host is then required to implement only the mapping from its own internal characteristics to NVT characteristics. Thus the total number of mappings to be implemented is H+T, rather than the H*T required for direct mapping.

A second basic concept of the TELNET protocol is a dichotomy between "users" and "servers". For example, in a communication between a human at terminal (working through a local terminal support program) to a remote service process, only the human would have a preference as to where echoing should be done, or would want to generate an interrupt. This viewpoint led to design decisions which made it difficult to use TELNET to support terminal-to-terminal, or process-to-process, communication.

Finally, it appeared necessary to have the user's (local) terminal support process and the server's operating system exchange some control information pertaining to the operating of the TELNET connection. The decision made was to embed TELNET control in the data stream to which it pertained. A small set of TELNET control codes was defined for this purpose.

2.5.3.2 Original Specification

TELNET was initially defined in the spring of 1971, although much of the definition was conveyed from implementer to implementer as "folklore" until a complete specification was committed to paper almost a year later. However, this informality did not prevent TELNET from being rapidly and compatibly implemented at a wide variety of sites.

2.5.3.2.1 Network Virtual Terminal Organization

The Network Virtual Terminal (NVT) is a bi-directional character device. The NVT has no timing characteristics. The characters are represented by 8 bit codes. The character codes 0 through 127 are the USASCII codes (code values are given in decimal). The codes 128 through 255 are used for special control signals. The NVT is described as having a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data.

2.5.3.2.2 NVT Printer

The NVT printer has an unspecified carriage width. The printer can produce representations of all 95 USASCII graphics. Of the 33 USASCII control codes, 8 have specific meaning to the NVT printer, as shown in Figure 2.5-4. The remaining USASCII codes do not cause the NVT printer to take any action.

2.5.3.2.3 NVT Data Keyboard

The NVT Keyboard has keys or key combinations or key sequences for generating all of the 128 USASCII codes. Note that although there are codes which have no effect on the NVT printer, the NVT Keyboard is capable of generating these codes.

The use of a standard, networkwide, intermediate representation of terminal code between sites is intended to eliminate the need for using and serving sites to keep

NAME	MEANING
NULL (NUL)	A no operation.
BELL (BEL)	Produces an audible or visible signal.
Back Space (BS)	Backspaces the printer one character position.
Horizontal Tab (HT)	Moves the printer to next horizontal tab stop.
Line Feed (LF)	Moves the printer to next line (keeping the same horizontal position).
Vertical Tab (VT)	Moves the printer to the next vertical tab stop.
Form Feed (FF)	Moves the printer to the top of the next page.
Carriage Return (CR)	Moves the printer to the left margin of the current line.

Figure 2.5-4: NVT Printer Controls

information about the characteristics of each other's terminals and terminal handling conventions. This approach can be successful, but only if the user, the using site, and the serving site assume certain responsibilities.

1. The serving site must specify how the intermediate code will be mapped by it into the terminal codes that are expected at that site.

2. The user must be familiar with that mapping.

3. The using site must provide some means for the user to enter all of the intermediate codes, and as described below, special TELNET signals, as well as specify for the user how the signals from the serving site will be presented at the user terminal.

Since it is not known how the sites will specify the mapping between the network-wide standard code (7 bit ASCII in an 8 bit field) and the codes expected from their own terminals, it is necessary to permit the user to cause transmission of every one of the 128 ASCII codes.

2.5.3.2.4 End-of-Line Convention

The representation of the end of a physical line at a terminal is implemented differently on different network Hosts. For example, some use a return (or new line) key; the terminal hardware both returns the carriage or printer to start of line and feeds the paper to the next line. In other implementations, the user hits carriage return and the hardware returns the carriage while the software sends the terminal a line feed. The network-wide representation is carriage return followed by line feed. It represents the physical formatting that is being attempted, and is to be interpreted and appropriately translated by both using site and serving site.

Although TELNET defines the end of a line to be indicated by the ASCII character pair CR LF, several of the real devices in the world have only a single new line (NL) function. Several of the computer systems have in some programs used the CR and LF functions to have semantic meaning larger than the format effect they provide. Further, several computer systems allow the CR and LF functions to be used seperately (e.g., such that a line may be overprinted). One problem, for those TELNET (user) programs required to map the NVT into a device which only has a NL function, is how is the CR LF to be dealt with. A solution is to examine the character following the CR. If a LF is found, then perform the NL function; if anything else is found then back space to the beginning of the line. Another problem is the case of a computer system which locally uses period, ".", to cause the new line function and which uses, in some programs, CR and LF for semanticially significant operations. Suppose the user TELNET sends the sequence CR LF. Does this mean "new line" or the "CR operation" followed by the "LF operation"? The solution to this problem is to require that TELNET programs send a CR which is <u>not</u>

intended to be part of a CR LF pair as a CR NUL pair. Then the receiving program can always hold a CR and examine the next character to determine if a new line function is intended.

2.5.3.2.5 Break Signal

There is a special control signal on some terminals that has no corresponding bit pattern in ASCII, but is transmitted by a special electric signal. This control signal is Attn on a 2741 and Break on a Teletype. Some systems treat the Break as an extra code available for use in conjunction with the data stream. For example, one system uses Break as a special editing code meaning "delete the current line to this point." For this reason, the user must be provided with a way of generating this 129-th code from the NVT Keyboard. A representation of the Break code must also be specified for inclusion in the data stream; the TELNET control code "BREAK" (described in Section 2.5.3.2.8) is the defined signal.

2.5.3.2.6 Interrupt Signal

The user at a terminal connected to an interactive system generally has the capability of instructing the system to begin execution of user-specified programs. Since such programs are subject to all types of errors, including those errors which result in unending execution loops, the user is normally provided with the capability of generating an "attention signal" which instructs the system to terminate the execution of the current process. (Some systems use the Break or Attn Key in this way, rather than as the 129-th code described in the preceding paragraph.)

One of the functions performed by a terminal control program within an operating system is the scanning of an input stream for attention characters intended to stop an errant process and to return control to the executive. Terminal control programs which buffer input sometimes run out of space. When this happens to a local terminal's input stream, a "bell" or a question mark is echoed and the overflow character discarded, after checking to see if it is the attention character. This strategy works well in practice, but it depends rather strongly on the intelligence of the human user, the invariant time delay in the input transmission system, and a lack of buffering between typein and attention checking.

None of these conditions exists for interactive traffic over the ARPANET. The serving Host cannot control the speed (except to slow it down) or the buffering within the using Host, nor can it even know whether a human user is supplying the input. It is thus necessary that the terminal control program or server TELNET not, in general, discard characters from a network input stream; instead it must suspend its acceptance of characters via the Host-Host flow control mechanism. Since a Host may only send messages when there is room at the destination, the responsibility for dealing with too much input is thus transferred back to the using Host. This assures that no characters accepted by the using Host are inadvertently lost.

However, if the process in the serving Host stops accepting input, the pipeline of buffers between the user TELNET and remote process will fill up so that attention characters cannot get through to the serving executive. The solution to this problem calls for the user TELNET to send, on user-specified request, a Host-Host interrupt signal forcing the server TELNET to switch input modes to process network input for attention characters. The server TELNET is required to scan for attention characters in its network input, even if some input must be discarded while doing so. The effect of the interrupt signal to a server TELNET from its user is to cause the buffers between them to be emptied for the priority processing of attention characters. Thus the user must be provided with a way of generating the "send-an-interrupt" signal from the NVT Keyboard. It must be noted that attention characters are Host specific and may be any of the 129 characters (128 ASCII characters plus Break); the requirements that the user must be able to generate the Break signal and the interrupt signal are independent.

To flip an attention scanning server TELNET back into its normal mode, a special TELNET synchronization character (DATAMARK) is defined (see Section 2.5.3.2.8). When the server TELNET encounters this character, it returns to the strategy of accepting terminal input only as buffer space permits. It would not do to use the Host-Host signal alone in place of the signal-DATAMARK combination in attention processing, because the position of the DATAMARK character in the TELNET input stream is required to determine where attention processing ends and where normal mode input processing beings.

2.5.3.2.7 Local Functions

The ability of the user to cause the using site TELNET to send any combination of ASCII characters in a string, and only that combination, is viewed as important to the user utility of the TELNET ASCII conventions. Because of this, some user sites may find it necessary to implement a few additional function keys as part of the local NVT implementation. These function keys are intended to provide local control between the terminal user and the terminal handling process. Two function keys which are likely to be universally required are:

Transmit Now: Transmit all data entered and locally buffered now. Intended to be used with line mode operation.

Suppress end-of-line: Transmit all data entered and locally buffered now, and do not transmit the end-of-line immediately following this signal.

2.5.3.2.8 Control Codes

For control of the interaction between the user and server TELNET processes, a set of TELNET control codes are defined; these control codes are to be embedded in the TELNET data stream at appropriate points. In order to avoid interference with the normal data, the control codes are chosen from the code values 128-255. Eight control codes are defined as described below:

BREAK: This is the 129-th data code which the user may cause to be transmitted, as described in Section 2.5.3.2.5.

NOP: This is a byte which may be used as a filler in the data stream to align text into a buffer location more convenient for the sender.

TRANSPARENT and EBCDIC: These codes specify the escape to an alternate data and control code set. Since the control code set is also changed, there is no defined way to return to the normal ASCII data and control code set described here.

DATAMARK: This code is used in conjunction with the Host-Host protocol INS command to flip a TELNET server into and out of a special attention character scanning mode as noted in Section 2.5.3.2.6.

NOECHO: When this code is sent from user to server, it asks the server not to send echos of the transmitted data. This is the default (startup) condition. When this code is sent from server to user, it states that the server is not sending echos of the transmitted data. The server is permitted to send this code only as a reply to ECHO or NOECHO from the user, or to terminate the effect of HYI (see below). The server is prohibited from spontaneously sending NOECHO (or ECHO - see below) to avoid various race conditions which can develop if both user and server both try to select an echo mode simultaneously.

ECHO: When this code is sent from user to server, it asks the server to send echos of the transmited data. When this code is sent from server to user, it states that the server is sending echos of the transmitted data. The server is permitted to send this code only as a reply to ECHO or NOECHO.

HYI (Hide Your Input): The intention is that a server will send this code to a user system which is echoing locally (to the user) when the user is about to type something secret (e.g., a password). In this case, the user system is to suppress local echoing or overprint the input until the server sends a NOECHO code. In situations where the user system is not echoing locally, this code must not be sent by the server.

The HYI code presents some difficulty in that it is unclear how much is to be hidden. The server site usually knows how long the secret is but the user TELNET in general does not. Furthermore, if the user site cannot suppress the local echoing, there is a difficult implementation problem. One possibility is for the using site to overprint a full line with a mask, then have the user type his secret on the mask. If the secret were longer than one line, the use of the mask should be repeated.

The use of HYI might be avoided by having the serving site send a mask on which the user is to type the secret information. This also presents a difficulty, however, in that the user system may be performing a mapping from terminal code to NVT code which maps multiple keystrokes (each with a local echo) into single NVT characters. In addition, the user system may provide editing functions that map a deleted character into several (rather than zero) characters. Thus, if the server sends a mask which it believes to be just long enough, some typing will be unmasked. On the other hand, sending a mask which is longer than the secret as expressed in NVT code may hide data which the user wants to see. Of course, the issue of masking arises only for those terminals whose internal echoing cannot be turned off.

2.5.3.3 Subsequent Modifications

By early 1973 there was a considerable amount of discontent with the TELNET protocol. There were several underlying themes to the discontent.

First, the dichotomy between "user" and "server" which was reflected in the details of many TELNET control structures (e.g., echoing, hiding input, interrupting) made it impossible to take advantage of the universality of TELNET implementations in using

TELNET for terminal-to-terminal or process-to-process communication.

Second, there was continuing pressure for the inclusion of new controls in the data stream, but each new control code which might be added would require handling by each implementation. This had the effect of requiring unanimous "approval" of any addition; proposed additions never were able to muster unanimous support. As one example, the IBM 2741 terminal contains wired logic which gives control of the print head to either the keyboard or the computer. The Keyboard is physically locked when the computer has control, so that the user cannot intersperse his input with the computer's output. Since the source of output, the serving computer, knows when it is through sending output it would be useful if it could send a control code to the user computer at that point, so the user computer could relinquish control of the print head to the keyboard.

Third, the handling of where to generate echos was made entirely the responsibility of the user. However, each server is optimized to best handle terminals which either do their own echoing or do not, but not both. Therefore, the echoing conventions, which prohibit the server from initiating a change in echo mode, seemed overly confining. The servers are burdened with users who are in the "wrong" mode, in which they might not have to be, and users, both human and machine, are burdened with remembering the proper echoing mode, and explicitly setting it up, for all the different servers. Another echoing issue was the desire of some users, especially those whose network communication involved satellite links (e.g., from Hawaii or Europe), to have a highly interactive echoing mode in which the server system instructed the user system to echo and not echo classes of characters, with the definition of which classes to echo changing fairly dynamically. This type of echoing can avoid long delays for every character while retaining the best human engineering features of the server systems which work in a character-at-a-time mode.

Fourth, it was recognized that almost every server system, as part of its own terminal support facilities, provided simple functions such as delete the last character, delete the last line, ring a bell to show you're still working, and so on. Unfortunately, almost every system chose a different user input string to activate each function. Both to simplify the task of a user in switching from system to system, as well as to facilitate the activation of such functions in process-to-process communication, it seemed appropriate to define standard activation signals as part of the NVT.

For reasons of the kind described above, a "new" TELNET was defined in mid-1973 to replace the previous TELNET. This "new" TELNET was declared to be the official version, and a timetable for implementation was established calling for use of "old" TELNET to be discontinued in early 1974. However, the number of sites involved, the reduction in ARPA funding for protocol implementation work on a widespread basis, and a general reluctance to tamper with the working system caused this schedule to be extended indefinitely. Nevertheless, many of the ARPANET Hosts, including the TIPs, have implemented the new TELNET and operate it in parallel with the old version.

The new TELNET differs from the old in three significant ways, which are discussed in the following sections. These are the principle of option negotiation, an expanded view of the NVT, and a revised view of the embedded control codes.

2.5.3.3.1 Option Negotiation

The principle of negotiated options takes cognizance of the fact that many sites will wish to provide additional services over and above those available within an NVT, and many users will have sophisticated terminals and would like to have elegant, rather than minimal, service. Independent of, but structured within, the TELNET Protocol various "options" are sanctioned which can be used with the "DO, DON'T, WILL, WON'T" structure (discussed below) to allow a user and server to agree to use a more elaborate (or perhaps just different) set of conventions for their TELNET connection. Such options could include changing the character set, the echo mode, the line width, the page length, etc.

The basic strategy for setting up the use of options is to have either party (or both) initiate a request that some option take effect. The other party may then either accept or reject the request. If the request is accepted the option immediately takes effect; if it is rejected the associated aspect of the connection remains as specified for an NVT. Clearly, a party may always refuse a request to enable, and must never refuse a request to disable, some option since all parties must be prepared to support the NVT.

The syntax of option negotiation has been set up so that if both parties request an option simultaneously, each will see the other's request as the positive acknowledgment of its own.

The symmetry of the negotiation syntax can potentially lead to nonterminating acknowledgment loops with each party seeing the incoming commands not as acknowledgments but as new requests which must be acknowledged. To prevent such loops, the following rules prevail:

a) Parties may only request a change in option status; i.e., a party may not send out a "request" merely to announce what mode it is in.

b) If a party receives what appears to be a request to enter some mode it is already in, the request should NOT be acknowledged.

c) Whenever one party sends an option command to a second party, whether as a request or an acknowledgment, and use of the option will have any effect on the processing of the data being sent from the first party to the second, then the command must be inserted in the data stream at the point where it is desired that it take effect. Some time will elapse between the transmission of a request and the receipt of an acknowledgment, which may be negative. Thus, a site may wish to buffer data, after requesting an option, until it learns whether the request is accepted or rejected, in order to hide the "uncertainty period" from the user.

It is possible for requests initiated by processes to simulate a nonterminating request loop if the process responds to a rejection by merely re-requesting the option. To prevent such loops from occurring, rejected requests should not be repeated until something changes. Operationally, this can mean the process is running a different program, or the user has given another command, or whatever makes sense in the context of the given process and the given option. A good rule of thumb is that a re-request should only occur as a result of subsequent information from the other end of the connection or when demanded by local human intervention.

Option designers need not feel constrained by the somewhat limited syntax available for option negotiation. The intent of the simple syntax is to make it easy to have options by making it correspondingly easy to profess ignorance about them. If some particular option requires a richer negotiation structure than possible within "DO, DON'T, WILL, WON'T", the proper tack is to use "DO, DON'T, WILL, WON'T" to establish that both parties understand the option, and once this is accomplished a more exotic syntax can be used freely. For example, a party might send a request to alter (establish) line length. If it is accepted, then a different syntax can be used for actually negotiating the line length; such a "subnegotiation" perhaps including fields for minimum allowable, maximum allowable and desired line lengths. The important concept is that such expanded negotiations should never begin until some prior (standard) negotiation has established that both parties are capable of parsing the expanded syntax.

In summary, WILL XXX is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and DON'T XXX being its positive and negative acknowledgments; similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing option XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the NVT is what is left when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, all Hosts may implement their TELNET processes to be totally unaware of options that are not supported, simply returning a rejection to (i.e., refusing) any option request that cannot be understood.

2.5.3.3.2 The Expanded NVT

As with the old TELNET specification, the NVT has a character printer and keyboard. The data is 7-bit ASCII code in an 8-bit field. Echos will not, in the default NVT cross the network. The user is to be provided with a way to generate all 128 ASCII codes, plus the Break code, and the Interrupt signal. The end of line convention is as specified for old TELNET. There are no timing considerations for an NVT.

There are also five new keys defined (below) for the new NVT keyboard which map into five new NVT codes. The spirit of these extra codes is that they should represent a natural extension of the mapping that already must be done from "NVT" into "local". Just as the NVT data byte 104 should be mapped into whatever the local code for "uppercase D" is, so the EC character should be mapped into whatever the local "Erase Character" function is. Further, just as the mapping for 174 is somewhat arbitrary in an environment that has no "vertical bar" character, the EL character may have a somewhat arbitrary mapping (or none at all) if there is no local "Erase Line" facility. (Similarly for format effectors: if the terminal actually does have a "Vertical tab", then the mapping for VT is obvious, and only when the terminal does not have a vertical tab should the effect of VT be unpredictable.) The new codes are:

IP (Interrupt Process): This is the "attention signal" described in Section 2.5.3.2.6 as motivation for sending the Host-Host protocol interrupt. The interpretation of this character is generally to suspend, interrupt, abort, or terminate the process to which the NVT is connected.

AO (Abort Output): Allow the current process to (appear to) run to completion, but do not send its output to the user. Also send an interrupt signal (a Host-Host protocol INS and a DATAMARK) to allow the clearing of buffers external to the system receiving the AO and the location of subsequent data which should not be cleared.

AYT (Are You There): Send back to the NVT some visible (i.e., printable) evidence that the AYT was received.

EC (Erase Character): The recipient should delete the last preceding undeleted character or "print position" from the data stream.

EL (Erase Line): The recipient should delete characters from the data stream back to, but not including, the last "CR LF" sequence sent over the TELNET connection.

The definition of the NVT is also expanded by the addition of guidelines for the transmission of data. Although a TELNET connection through the network is intrinsically full duplex, the NVT is to be viewed as a half-duplex device operating in a line-buffered mode. That is, unless and until options are negotiated to the contrary, the following default conditions pertain to the transmission of data over the TELNET connection:

1) Insofar as the availability of local buffer space permits, data should be accumulated in the Host where it is generated until a complete line of data is ready for transmission, or until some locally-defined explicit signal to transmit occurs. This signal could be generated either by a process or by a human user.

   The motivation for this rule is the high cost, to some Hosts, of processing network input interrupts, coupled with the default NVT specification that "echos" do not traverse the network. Thus, it is reasonable to buffer some amount of data at its source. Many systems take some processing action at the end of each input line (even line printers or card punches frequently tend to work this way), so the transmission should be triggered at the end of a line. On the other hand, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line; therefore there must be methods of locally signalling that all buffered data should be transmitted immediately.

2) When a process has completed sending data to an NVT printer and has no queued input from the NVT keyboard for further processing (i.e., when a process at one end of a TELNET connection cannot proceed without input from the other end), the process must transmit the TELNET Go Ahead (GA) command. This rule is not intended to require that the TELNET GA command be sent from a terminal at the end of each line, since server Hosts do not normally require a special signal (in addition to end-of-line or other locally-defined characters) in order to commence processing. Rather, the TELNET GA is designed to help a user's local Host operate a physically half duplex terminal which has a "lockable" keyboard such as the IBM 2741.

2.5.3.3.3 Revised Control Code Structure

All TELNET commands consist of at least a two byte sequence: the "Interpret as Command" (IAC) escape character (code 255) followed by the code for the command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option referenced. This format was chosen so that as more comprehensive use of the "data space" is made (by negotiations from the basic NVT) collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and inefficiency, of "escaping" the data bytes into the stream. With the current setup, only the IAC need be doubled to be sent as data, and the other 255 codes may be passed transparently.

The defined TELNET commands are shown in Figures 2.5-5 and 2.5-6. These codes and code sequences have the indicated meaning only when immediately preceded by an IAC.

2.5.3.3.4 Defined Options

Because a participant in a TELNET conversation can refuse an option without understanding what it is, there is no pressure to limit the number of defined options, nor is there any pressure, other than the desire to provide more features, to implement defined options. By mid-1977 the 21 options had been defined as shown in Figure 2.5-7.

NAME	MEANING
IAC	Data byte 255
NOP	No operation
Data Mark	This should always be accompanied by an INS on the control link
Break	NVT 129-th character
Interrupt Process	The function IP
Abort Output	The function AO
Are You There	The function AYT
Erase Character	The function EC
Erase Line	The function EL
Go Ahead	The GA signal

Figure 2.5-5: TELNET Commands (Except Option) Negotiation

2.5.3.4 Conclusions

Old TELNET has proven eminently suitable for the connection of full-duplex terminals to server systems. It has been used, but with some difficulty, for communication between processes or between terminals. It does not provide a sufficient control structure to handle a basic half-duplex terminal, or to incorporate all of the optional facilities that subsets of users desire. New TELNET was designed to deal with these shortcomings.

NAME	**MEANING**
WILL (option code)	Indicates the sender desires to begin performing, or confirmation that the sender is now performing, the indicated option
WON'T (option code)	Indicates the refusal to perform, or continue performing, the indicated option
DO (option code)	Indicates the request that the other party perform, or confirmation that the sender is expecting the other party to perform, the indicated option
DON'T (option code)	Indicates the demand that the other party stop performing, or confirmation that the sender is no longer expecting the other party to perform, the indicated option
SB (option code)	Indicates that what follows is a list of subnegotiation parameters pertaining to the indicated option. The list will be terminated by IAC SE
SE	Subnegotiation End - indicates the end of a list of option subnegotiation parameters

Figure 2.5-6: TELNET Option Negotiation Commands

and appears to do so successfully. Nevertheless, the experience with TELNET illustrates the tremendous inertia associated with a

1. Switch to binary transmission mode
2. Send Echos over the TELNET connections
3. Reconnect the TELNET connection
4. Suppress transmission of GA (Go-Ahead)
5. Select a convenient message size
6. Provide current-status-of-options information
7. Send a timing-mark (to indicate a roundtrip over the data connections)
8. Enter a complex echoing mode based on classes of characters to be echoed at each end.
9. Select a convenient print line length
10. Select a convenient page length
11. Choose the best system to provide any necessary padding after carriage-return
12. Select horizontal tabstops
13. Choose the best system to properly handle horizontal tabs
14. Choose the best system to properly handle formfeeds
15. Select vertical tabstops
16. Choose the best system to properly handle vertical tabs
17. Choose the best system to properly handle linefeeds
18. Agree to use a particular "extended ASCII" character set
19. Log the user off the server system
20. Use a particular text-compaction scheme
21. Switch from operation of an NVT to a "virtual" Data Entry Terminal

Figure 2.5-7: TELNET Options

program that works. The discarding of old TELNET is now about four years overdue (out of a total age of less than seven years). An important conclusion, therefore, is that the first version of a protocol which must be implemented by many different groups will persist in spite of later improvements; it is therefore very important to approach perfection as closely as possible from the beginning. This inertia can be contrasted with the relative ease with which the IMP software, built by a single group on only one type of machine, has been drastically changed during the course of the ARPANET project.

A second conclusion to be drawn from the TELNET experience is the unifying power of the "virtual device" concept, coupled with a mandatory minimal implementation extended by negotiable options. The virtual device concept pervades modern terminal protocol development throughout the world.

2.5.4 File Transfer Protocol (FTP)

The File Transfer Protocol specifies a mechanism for the transfer of complete files (rather than, for example, groups of "records" retrieved from a file) between ARPANET Hosts. The primary function of FTP is to transfer files efficiently and reliably among Hosts and to allow the convenient use of remote file storage capabilities.

The objectives of FTP are:

1) to promote sharing of files (computer programs and/or data)

2) to encourage indirect or implicit (via programs) use of remote computers

3) to shield a user from variations in file storage systems among Hosts

4) to transfer data reliability and efficiently.

FTP, though usable directly by a user at a terminal, is designed mainly for use by programs. The attempt is to satisfy the diverse needs of users of maxi-Hosts, mini-Hosts, TIPs, and the Datacomputer, with a simple, and easily implemented protocol design.

The effort to develop a File Transfer Protocol began with the conceptualization of two levels: a "data transfer protocol" to govern the transmission of raw data, and a "file transfer protocol" which would control the data flow, establish the names and locations of files to be moved, user access rights, and so on. Preliminary specifications for these two levels were developed, but it eventually became clear that the two layers were actually only the format-specification and procedure-specification aspects of a single protocol, the current FTP.

2.5.4.1 Basic Concepts

As with TELNET, the FTP can be best understood in terms of a user and a server. Unlike TELNET, however, the protocol expects the user to be a process, rather than a human at a keyboard. Nevertheless, some care has been taken to make it possible, if not convenient, for the user to be a human.

Figure 2.5-8 provides a sketch of the overall model of the FTP system. The "Server FTP Process" can be further subdivided into a server Protocol Interpreter (PI) which interfaces to the command/response path via TELNET, and a Data Transfer Process (DTP) which moves the data between the data connection and the

Figure 2.5-8: ARPANET File Transfer Protocol

file system. Similarly, the "user FTP Process" can be subdivided into a Protocol Interpreter, a Data Transfer module, and a User Interface which mediates between the Protocol Interpreter and the local user support facilities.

In a typical session, the user would be expected to activate the User FTP Process. The user-protocol interpreter then initiates the TELNET connections. At the initiation of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the TELNET connections. (The user may establish a direct TELNET connection to the server-FTP, from a TIP terminal for example, and generate standard FTP commands himself, by-passing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the TELNET connections in response to the commands.

The FTP commands specify the parameters for the data connection (representation type, file structure, and transfer mode), and the nature of file system operation (store, retrieve, append, delete, etc.). The user-DTP or its designate should "listen" on the specified data socket, and the server initiate the data connection and data transfer in accordance with the specified parameters. It should be noted that the data socket need not be in the same Host that initiates the FTP commands via the TELNET connections, but the user or his user-FTP process must ensure a "listen" on the specified data socket. It should also be noted that two data connections, one for send and the other for receive, may exist simultaneously.

In another situation a user might wish to transfer files between two Hosts, neither of which is his local Host. He sets up TELNET connections to the two servers and then arranges for a data connection between them. In this manner control information is passed to the user-PI but data is transferred between the server data transfer processes.

FTP commands are "TELNET strings" terminated by the "TELNET end of line code". The command codes themselves are alphabetic characters terminated by the character <SP> (Space) if parameters follow and CR-LF as an end-of-line indication if not.

Replies to File Transfer Protocol commands were devised to ensure the synchronization of requests and actions in the process of file transfer, and to guarantee that the user process always knows the state of the Server. Every command must generate at least one reply, although there may be more than one; in the latter case, the multiple replies must be easily distinguished. In addition, some commands occur in sequential groups. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning.

An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intended that the three digits contain enough encoded information that the User-PI will not need to examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be server-dependent, so there are likely to be varying texts for each reply code.

There will be cases where the text is longer than a single line. In these cases the complete text must be bracketed so the User-process knows when it may stop reading the reply (i.e. stop processing input on the TELNET connection) and go do other things. This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to indicate the state of the transaction. The format chosen for multi-line replies which meets these constraints is that the first line begins with the exact required reply code, followed immediately by a hyphen, "-", followed by text. The last line begins with the same code, followed immediately by space <SP>, optionally some text, and CR LF. For example:

```
123-First line
Second line
 234 A line beginning with numbers
123 The last line
```

The user-process then simply needs to search for the second occurrence of the same reply code, followed by <SP> (Space), at the beginning of a line, and ignore all intermediary lines. If an intermediary line begins with a 3-digit number, the Server must pad the front to avoid confusion.

The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated response by the user-process. The first digit denotes whether the response is good, bad or incomplete. An unsophisticated user-process will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A user-process that wants to know approximately what kind of error occurred (e.g. file system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information.

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply: The requested action is being initiated; expect another reply before proceeding with a new command. This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult.

2yz Positive Completion reply: The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply: The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply: The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g. the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz Permanent Negative Completion reply: The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g. after the spelling has been changed, or the user has altered his directory status.)

The following function groupings are encoded in the second digit:

x0z Syntax: These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.

x1z Information: These are replies to requests for information, such as status or help.

x2x Connections: Replies referring to the TELNET and data connections.

x3x Authentication and accounting: Replies for the logon process and accounting procedures.

x5x File system: These replies indicate the status of the Server file system vis-a-vis the requested transfer or ther file system action.

The third digit gives a yet finer gradation of meaning in each of the function categories.

2.5.4.2 Original Specification

There are five areas of FTP specification. Three of them (representation type, file structure, and transmission mode) deal with the data transfer aspects of the protocol, while the other two (data error recovery, command structure) deal with the procedural aspects of the protocol. Each of these five areas is described briefly below.

2.5.4.2.1 Data Representation Types

Data is transferred from a storage device in the sending Host to a storage device in the receiving Host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. For example, NVT-ASCII has different data storage representations in different systems. PDP-10's generally store NVT-ASCII as five 7-bit ASCII characters, left-justified in a 36-bit word; 360's store NVT-ASCII as 8-bit EBCDIC codes; Multics stores NVT-ASCII as four 9-bit characters in a 36-bit word. It may be desirable to convert characters into the standard NVT-ASCII representation when transmitting text between dissimilar systems. The sending and receiving sites would have to perform the necessary transformations between the standard representation and their internal representations.

A different problem in representation arises when transmitting binary data (not character codes) between Host systems with different word lengths. It is not always clear how the sender should send data, and the receiver store it. For example, when transmitting 32-bit bytes from a 32-bit word-length system to a 36-bit word-length system, it may be desirable (for reasons of efficiency and usefulness) to store the 32-bit bytes right-justified in a 36-bit word in the latter system. In any case, the user should have the option of specifying data representation and transformation functions. FTP provides for very limited data type representations; the following types are defined:

ASCII Format: This is the default type and must be accepted by all FTP implementations. It is intended primarily for the transfer of text files, except when both Hosts would find the EBCDIC type more convenient. The sender converts the data from his internal character representation to the standard 8-bit NVT-ASCII representation as defined in the TELNET specification. The receiver will convert the data from the standard form to his own internal form. In accordance with the NVT standard, the <CRLF> sequence should be used, where necessary, to denote the end of a line of text.

EBCDIC Format: This type is intended for efficient transfer between Hosts which use EBCDIC for their internal character representation. For transmission the data are represented as 8-bit EBCDIC characters. The character code is the only difference between the functional specifications of EBCDIC and ASCII types. End-of-line (as opposed to end-of-record) will probably be rarely used with EBCDIC type for purposes of denoting structure, but where it is necessary the <NL> character should be used.

A character file may be transferred to a Host for one of three purposes: for printing, for storage and later retrieval, or for processing. If a file is sent for printing, the receiving Host must know how the vertical format control is represented. In the second case, it must be possible to store a file at a Host and then retrieve it later in exactly the same form. Finally, it ought to be possible to move a file from one Host to another and process the file at the second Host without undue trouble. A single ASCII or EBCDIC format does not satisfy all these conditions and so these types have a second parameter specifying one of the following three formats:

Non-print: This is the default format. The file need contain no vertical format information. Normally, this format will be used with files destined for processing or just storage.

TELNET Format Controls: The file contains ASCII/EBCDIC vertical format controls (i.e., <CR>, <LF>, <NL>, <VT>, <FF>) which the printer process will interpret appropriately. <CRLF>, in exactly this sequence, also denotes end-of-line.

Carriage Control (ASA): The file contains ASA (FORTRAN) vertical format control characters. In a line or a record, formatted according to the ASA Standard, the first character is not to be printed. Instead it should be used to determine the vertical movement of the paper which should take place before the rest of the record is printed.

Image: The data are sent as contiguous bits which the receiving site must store as contiguous bits. The structure of the storage system might necessitate the padding of the file (or of each record, for a record-structured file) to some convenient boundary (byte, word or block). This padding may occur only at the end of the file (or at the end of each record) and there must be a way of identifying the padding bits so that they may be stripped off if the file is retrieved. The padding transformation should be well publicized to enable a user to process a file at the storage site. Image type is intended for the efficient storage and retrieval of files and for the transfer of binary data.

Of course, a file must be stored and retrieved with the same parameters if the retrieved version is to be identical to the version originally transmitted. Conversely, FTP implementations must return a file identical to the original if the parameters used to store and retrieve a file are the same.

2.5.4.2.2 File Structures

FTP recognizes two file structures: a "pure" file structure with no internal substructure, and a record structure in which the file is composed of a series of records. Of course, these two recognized structures represent only a small percentage of the total number of file structures which have been defined, but they are sufficient to handle most of the files which ARPANET Hosts wish to exchange.

The "natural" structure of a file will depend on which Host stores the file. A source-code file will usually be stored on an IBM 360 in fixed length records but on a PDP-10 as a stream of characters partitioned into lines, for example by <CRLF>. If the transfer of files between such disparate sites is to be useful, there must be some way for one site to recognize the other's assumptions about the file.

With some sites being naturally file-oriented and others naturally record-oriented there may be problems if a file with one structure is sent to a Host oriented to the other. If a text file is sent with record-structure to a Host which is file oriented, then that Host must apply an internal transformation to the file based on the record structure. Obviously this transformation should be useful but it must also be invertible so that an identical file may be retrieved using record structure.

In the case of a file being sent with file-structure to a record-oriented Host, there exists the question of what criteria the Host should use to divide the file into records which can be processed locally. If this division is necessary the FTP implementation should use the end-of-line sequence, <CRLF> for ASCII, or <NL> for EBCDIC, text files as the delimiter. If an FTP implementation adopts this technique, it must be prepared to reverse the transformation if the file is retrieved with file-structure.

2.3.4.2.3 Transmission Modes

The final consideration in transferring data is choosing the appropriate transmission mode. There are three modes defined in FTP; one which formats the data and allows for restart procedures; one which also compresses the data for efficient transfer; and one which passes the data with little or no processing. In this last case the mode interacts with the structure attribute to determine the type of processing. In the compressed mode the representation type determines the filler byte.

All data transfers must be completed with an end-of-file (EOF) which may be explicitly stated or implied by the closing of the data connection. For files with record structure, all the end-of-record markers (EOR) are explicit, including the final one.

The following transmission modes are defined:

Stream: The data is transmitted as a stream of bytes. There is no restriction on the representation type used; record structures are allowed. In a record structured file EOR and EOF will each be indicated by a two-byte control code of whatever byte size is used for the transfer. The first byte of the control code will be all ones, the escape character. The second byte will have value 1 for EOR and value 2 for EOF. EOR and EOF may be indicated together on the last byte transmitted by the value 3. If a byte of all ones was intended to be sent as data, it should be repeated in the second byte of the control code.

If the file does not have record structure, the EOF is indicated by the sending Host closing the data connection and all bytes are data bytes.

For the purpose of standardized transfer, the sending Host translates its internal end of line or end of record denotation into the representation prescribed by the transfer mode and file structure, and the receiving Host performs the inverse translation to its internal denotation. Since these transformations imply extra work for some systems, identical systems transferring non-record structured text files might wish to use a binary representation and stream mode for the transfer.

Block: The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines: last block in the file (EOF), last block in the record (EOR), restart marker (see Section 2.5.4.2.4), or suspect data (i.e., the data being transferred is suspected of errors and is not reliable). This last code is not intended for error control within FTP. It is motivated by the desire of sites exchanging certain types of data (e.g., seismic or weather data) to send and receive all the data despite local errors (such as "magnetic tape read errors"), but to indicate in the transmission that certain potions are suspect).

Record structures are allowed in this mode, and any representation type may be used. There is no restriction on the transfer byte size. With this encoding more than one descriptor coded condition may exist for a particular block. As many bits are necessary may be flagged.

The restart marker is embedded in the data stream as an integral number of 8-bit bytes representing printable characters in the language being used over the TELNET connection (e.g., default is NVT-ASCII).

Compressed: The file is transmitted as series of bytes of the size specified by the BYTE command. There are three kinds of information to be sent: regular data, sent in a byte string; compressed data, consisting of replications or filler; and control information, sent in a two-byte escape sequence. If n bytes of regular data are sent, these n bytes are preceded by a byte with the leftmost bit set to 0 and the other bits containing the number n.

If n replications of the data byte d are to be transmitted, they can be compressed into two bytes. The leftmost two bits of the first byte are set to 10 and the remaining bits contain the number n. The second byte contains the value d.

A string of n filler bytes can be compressed into a single byte, where the filler byte varies with the representation type. If the type is ASCII or EBCDIC the filler byte is space. If the type is Image or Local byte, the filler is a zero byte. The compressed filler is a single byte with the leftmost two bits set to 11 and the remaining bits containing the number n.

The escape sequence is a double byte, the first of which is the escape byte (all zeroes) and the second of which contains descriptor codes as defined in Block mode. The descriptor codes have the same meaning as in Block mode and apply to the succeeding string of bytes.

Compressed mode is useful for obtaining increased bandwidth on very large network transmissions at a little extra CPU cost. It is most efficient when the byte size chosen is that of the word size of the transmitting Host.

2.5.4.2.4 Error Recovery

There is no provision for detecting bits lost or scrambled in data transfer. However, a restart procedure is provided to protect users from gross system failures (including failures of a Host, an FTP-process, or the IMP subnet).

The restart procedure is defined only for the block and compressed modes of data transfer. It requires the sender of data to insert a special marker code in the data stream with some marker information. The marker information has meaning only to the sender, but must consist of printable characters in the code of the TELNET connection. The marker could represent a bit-count, a record-count, or any other information by which a system may identify a data checkpoint. The receiver of data, if it implements the restart procedure, would then mark the corresponding position of this marker in the receiving system, and return this information to the user.

In the event of a system failure, the user can restart the data transfer by identifying the marker point with the FTP restart procedure. The following example illustrates the use of the restart procedure.

The sender of the data inserts an appropriate marker block in the data stream at a convenient point. The receiving Host marks the corresponding data point in its file system and conveys the last known sender and receiver marker information to the user, either directly or over the TELNET connection, depending on who is the sender. In the event of a system failure, the user or controller process restarts the server at the last server marker by sending a restart command with the server's marker code as its argument. The restart command is transmitted over the TELNET connection and is immediately followed by the command (such as Store or Retrieve) which was being executed when the system failure occurred.

2.5.4.2.5 FTP Commands

There are three general classes of FTP commands. First, there are commands which deal with the issues of access control and accounting; logging on and logging off the server system.

Second, there are the commands which deal with the data transfer issues such as choosing a representation type, file structure, and transmission mode. Commands to establish socket numbers and a byte size for the data connection are also included. Defaults exist for all of the data transfer parameters, so these commands are needed only if the default are to be changed.

Third, there are the service commands which define the file transfer or the file system function requested by the user. The argument of an FTP service command will normally be a pathname. The syntax of pathnames must conform to server site conventions (with standard defaults applicable), and the language conventions of the TELNET connection. The commands may be in any order except that a "rename from" command must be followed by a "rename to" command and the restart command must be followed by the interrupted service command. The data, when transferred in response to FTP service commands, is always sent over the data connection, except for certain informative replies. The commands to specify FTP service include:

Retrieve - causes the server to send a copy of the named file over the data connection.

Store - causes the server to accept the data transferred over the data connection and store it with the given pathname.

Append - causes the server to accept the data transferred via the data connection and append it to the named file, or create the named file if it does not exist.

Allocate - reserve space for file storage.

Restart - accept a restart marker (see Section 2.5.4.2.4) and prepare to resume a data transfer.

Rename - a "from" and a "to" command, used in pairs, to rename a file stored at the server.

Abort - any action of the last-given unfinished command, especially the transfer of data.

Delete - a file stored at the server.

List - depending on parameters, send a list of all the files belonging to the user or stored in a named directory, or details about the storage of a named file.

Status - causes the server to send a status response over the TELNET connection; in general this is used to obtain the server's view of the status of a data transfer which is in progress.

2.5.4.3 Conclusions

As previously noted, the official FTP was developed fairly late relative to the Host-Host and TELNET protocols, and thus was able to incorporate features to deal with some of the shortcomings experienced during use of these protocols. The two most interesting ideas in the FTP are the effort to separate the data and control functions so completely (i.e., with separate connections using different formatting/encoding), and the structuring of replies so that they can be used either by human or automaton.

The separation of data and control was seen to be successful at the Host-Host level; it is far less successful in FTP, where synchronization problems (both over the network and within a server system) have forced some control to be embedded in the data stream, and have also in general prohibited the user system from having multiple commands outstanding.

The use of responses (and commands) which can be generated and interpreted both by humans and programs (even on radically different systems) is probably the major technical contribution of the ARPANET FTP to the body of protocol ideas. The concept has been proven to work quite well in practice, and has been copied in the development of protocols for other networks.

On the other hand, the FTP has been criticized for attempting to obtain generality by the method of including a little of something for everyone; witness the six representation types (ASCII and EBCDIC each with three subtypes), two file structures, and three transmission modes - a total of 36 combinations. It has been suggested by several protocol workers that a more rational approach is to define a "Network Virtual File System"; a single, general, simple, albeit inefficient protocol that could be used by all casual users of file transfer and similar functions. This could even be embedded in TELNET. All communication between pairs (or other natural groups) of serious users should be done using special-purpose protocols. While this may represent an extreme view, it would have been much easier to implement and would have encouraged users with large volumes of data, and a desire for efficiency, to use a special-purpose conversion programs, well-matched to the specific application, rather than a general-purpose conversion at each end.

2.5.5 Other High-Level Protocols

There are a number of additional high-level protocols, some "official" and others ad hoc, which have been designed for the ARPANET. This section contains brief descriptions of three such protocols: the official Remote Job Entry protocol, the "competing" ad hoc Remote Job Service protocol, and the official Graphics protocol.

2.5.5.1 Remote Job Entry Protocol

Remote job entry is the mechanism whereby a user at one location causes a batch-processing job to be run at some other location. This protocol specifies the Network standard procedures for such a user to communicate over the Network with a remote batch-processing server, causing that server to retrieve a job-input file, process the job, and deliver the job's output file(s) to a remote location. The protocol uses a TELNET connection for all control communication between the user and the server RJE processes. The server-site then uses the File Transfer Protocol to retrieve the job-input file and to deliver the output file(s). This mechanism is illustrated in Figure 2.5-9. It should be noted that the RJE user need not be located at the Host which sends the input or receives the output (i.e., the Host containing the FTP server).

Figure 2.5-9: ARPANET Official RJE Protocol

There are two types of users: direct users (persons) and user processes. The direct user communicates from an interactive terminal attached to a Host. This user may cause the input and/or output to be retrieved/sent on a specific socket at the specified Host (such as for card readers or printers on a TIP), or the user may have such files transferred by file-name using File Transfer Protocol. The other type of user is an RJE User-process in one remote Host communicating with the RJE Server-process in another Host. This type of user ultimately receives its instructions from a human user, but through some unspecified indirect means. The command and response streams of this protocol are designed to be readily used and interpreted by both the human user and the user process.

A particular user site may choose to establish the TELNET control connection for each logical job or may leave the control connection open for extended periods. If the control connection is left open, then mulitple job-files may be directed to be retrieved or optionally (to servers that are able to determine the end of each logical job and form several jobs out of one input file) one continuous retrieval may be done (as from a TIP card reader). This then forms a "hot" card reader to a particular server with the TELNET connection serving as a "job monitor". Since the output is always transferred a job at a time per connection to the output socket, the output from this "hot" reader would appear when ready as if to a "hot" printer. Another possibility for more complex hosts is to attach an RJE User-process to a card reader and take instructions from a lead control card, causing an RJE control TELNET to be opened to the appropriate Host with appropriate log-on and input retrieval commands. This card reader would appear to the human user as a Network "hot" card reader.

The RJE commands and responses follow the same general rules as defined for FTP. In fact, the reply codes are primary those defined for FTP, with only some minor additions. The command structure is somewhat more complicated than exists for FTP, for three reasons. First, the RJE server site contains an FTP user which must typically "log in" to the Host which is the source or destination of the job data. Thus, the RJE user must not only supply accounting and authentication data to gain entrance to the RJE server; the user must also supply authentication and accounting information to be used by the RJE server to gain access to the files. Second, the user must be able to specify a Host system from which the RJE server is to obtain the input and a system to which the RJE server is to send the output; these systems need not be the Host at which the user is located, nor need they be the same as each other. Third, the user must specify whether the RJE server is to actively try to send the output after it becomes available or wait for the user to retrieve it. The user must also specify whether the RJE server is to discard or retain the output after it has been sent to the destination Host.

Although the official RJE protocol was specified by late 1972, it has probably never been implemented. This is further testimony to the inertia of a working existing system. The competing system in this case is the Network Remote Job Service protocol specified as an "interim" measure a year and a half earlier; the users and servers with a real need to perform remote batch computing implemented this protocol immediately and never bothered to change.

2.5.5.2 Network Remote Job Service

NETRJS is the protocol for the remote job entry service on the IBM 360 Model 91 at the UCLA Campus Computing Network (CCN). NETRJS allows the user at a remote Host to access CCN's RJS ("Remote Job Service") subsystem, which provides remote job entry service to real remote batch (card reader/line printer) terminals over direct communications lines as well as to the ARPANET.

To use NETRJS, a user at a remote Host needs a NETRJS user process to communicate with one of the NETRJS server processes at CCN. Each active NETRJS user process appears to RJS as a separate (virtual) remote batch terminal (VRBT). A VRBT may have virtual card readers, printers, and punches. Through a virtual card reader a user can transmit a stream of card images comprising one or more OS/360 jobs, complete with Job Control Language, to CCN. These jobs are spooled into CCN's batch system (OS/360 MVT) and run according to their priority. RJS will automatically return the print and/or punch output images which are created by these jobs to the virtual printer and/or card punch at the VRBT from which the job came (or to a different destination specified in the JCL). The remote user can wait for the output, or can sign off and sign back on later to receive it. Figure 2.5-10 illustrates the processes and connections involved in the use of NETRJS.

The VRBT is assumed to be under the control of the user's console; this serves the function of an RJS remote operator console. To initiate a NETRJS session, the remote user must establish a TELNET connection and sign into RJS. Once signed in, he can use his console to issue commands to RJS and to receive status, confirmation, and error messages from RJS. Different

Figure 2.5-10: NETRJS Protocol

VRBTs are distinguished by 8-character terminal id's. There may be more than one VRBT using RJS simultaneously from the same remote Host.

When a VRBT starts a session, it has a choice of two ICP sockets, depending upon whether it is an ASCII or an EBCDIC virtual terminal. An EBCDIC virtual terminal transmits and receives its data as transparent streams of 8 bit bytes (since CCN is an EBCDIC installation). It is expected that a user at an ASCII installation, however, will want his VRBT declared ASCII; RJS will then translate the input stream from ASCII to EBCDIC and translate the printer stream back to ASCII. This will allow the user to employ his local text editor for preparing input to CCN and for examining output. The punch stream will always be transparent, for outputting "binary decks". The choice of code for the operator console connections is independent of declared terminal type; in particular, they always use ASCII under TELNET protocol, even from an EBCDIC VRBT.

NETRJS protocol provides data compression, replacing repeated blanks or other characters by repeat counts. However, when the terminal id is assigned by CCN, a particular network terminal may be specified as using no data compression. In this case, NETRJS will simply truncate trailing blanks and send records in a simple "op code-length-data" form, called "truncated format".

A job stream for submission to RJS at CCN is a series of logical records, each of which is a card image. A card image may be at most 80 characters long, to match the requirements of OS/360 for job input. The user can submit a "stack" of successive jobs through the card reader channel with no end-of-job indication between jobs; RJS recognizes the beginning of each new job by the appearance of a JOB card. For each job successfully spooled, the user will receive a confirming message on his console. At the end of the stack, he must send an End-of-Data transaction to initiate processing of the last job. NETRJS will then close the channel (to avoid holding buffer space unnecessarily). At any time during the session, the user can re-open the card reader channel and transmit another job stack. He can also terminate the session and sign on later to get his output.

The user can abort the card reader channel at any time by closing the channel. NETRJS will then discard the last partially spooled job. If NETRJS finds an error it will abort the channel by closing the connection prematurely, and also inform the user via his console that his job was discarded (thus solving the race condition between End-of-Data and aborting). The user needs to retransmit only the last job, however, he could retransmit the entire stack (although it would be somewhat wasteful) since the CCN operating system enforces job name uniqueness by immediately "flushing" jobs with names already in the system.

If the user's process, NCP, or Host, or the Network itself fails during input, RJS will discard the job being transmitted. A message informing the user that this job was discarded will be generated and sent to him the next time he signs on. On the other hand, those jobs whose receipt has been acknowledged on the operator's console will not be affected by the failure, but will be executed by CCN.

The user may wait to set up a virtual printer (or punch) and open its channel until a STATUS message on his console indicates output is ready; or he may leave the output channel(s) open during the entire session, ready to receive output whenever it becomes available. He can also control which one of several available jobs is to be returned by entering appropriate operator commands.

When RJS has output to send to a particular (virtual) terminal and a corresponding open output channel, it will send the output as a series of logical records. NETRJS will send an End-of-Data transaction and then close an output channel at the end of the output for each complete batch job; the remote site must open a new channel to start output for another job. This gives the remote site a chance to allocate a new file for each job without breaking the output within a job.

If the user detects an error in the stream, he can issue a Backspace command from his console to repeat the last "page" of output, or a Restart command to repeat from the beginning of the job, or he can abort the channel by closing his socket. If he aborts the channel, RJS will simulate a Backspace command, and when the user re-opens the channel the job will begin transmission again from an earlier point in the same data set. If the user's process, NCP, or Host, or the Network itself fails during an output operation, RJS will act as if the channel had been aborted and the user signed off.

Although the NETRJS protocol was defined by UCLA CCN as an interim protocol for use to "get started" while the official RJE protocol was being designed, as noted in the previous section, NETRJS has remained in operation and the official RJE has gone unimplemented. Further, any site wishing to compete with CCN for the batch service "market" is essentially obliged to implement NETRJS because the batch user processes use this (rather than the RJE) protocol. The lesson is quite clear: in an effort-limited world the first specification to be implemented has a very high probability of becoming the de facto standard.

2.5.5.3 Graphics

The aim of a graphics protocol is to allow users with various different kinds of display hardware at different sites in a network to make use of common graphics applications programs. One approach is a collection of special-purpose protocols. Each application program could publish a description of the protocol needed to drive the program and to view the output. The prospective user would then write a program to interpret the published protocol and to drive his display (probably making use of existing graphics programming facilities at his site). This might permit a convenient division of labor between the computer executing the application program and the computer driving the display. The disadvantage of this approach is that the user must write a new program to interface his display to each different application protocol. In addition, there is no guarantee that the protocol required by the application program can actually be implemented within the user's operating system and display hardware.

Another approach is to develop one general-purpose protocol such as a Network Virtual Graphics Terminal that tries to provide facilities that a large number of application programs could use and that a large number of user sites could interpret as with TELNET and the NVT. Thus, one user program could be used to interface to a number of application programs. The disadvantages of this approach are that the generality may preclude adequate response through the network, or that some application programs will find the general-purpose protocol too restrictive to be used at all. In addition, the design of such a protocol is not easy; attempting to provide a common device-independent framework for driving hardware of qualitatively different capabilities may be very difficult.

The network graphics protocol attempts a middle course. It is not intended to satisfy all graphics needs, for all terminals, now and in the future. It is limited to calligraphic pictures, to moderate interaction demands, and to "best effort" attempts to generate the required graphics. Certainly, the impact of video technology will increase demand for variable character sets, shading, and maybe even animation techniques and it is anticipated that special-purpose protocols will be necessary for these (and certain present) applications. In such cases, the general-purpose protocol can perhaps be used as a starting point for development of special-purpose protocols. The general-purpose protocol should be used whenever possible, however, so that separate user programs need not be written for each user site.

A user session with a graphics application program, as illustrated in Figure 2.5-11, is modeled on a TELNET session with a program; the user must log into a server system, execute several system commands, initiate the program, communicate with the program as it is running, perhaps interrupt a running program, log out, etc. A graphics session certainly requires all of these features; in addition it will require a pathway for transmitting graphics protocol information in both directions.

The application program, when it wishes to produce graphical output or to request graphical input, makes use of a server program (SP) which may simply be a subroutine package. The job

Figure 2.5-111 Graphics Protocol

of the SP is to interface to the network graphics protocol on one side and to a "graphics language" on the other.

The protocol transmitted between the SP and the user program (UP) is the graphics protocol. It provides facilities so that:

1. The SP can cause images to appear on the user's display screen.

2. The UP can report to the SP any interactions that the user initiates, such as a key depressed on the keyboard or stylus interactions.

3. The SP can discover various special properties of the user's display terminal, and can take advantage of them in conjunction with the application program.

These types of protocol transmission are termed "output," "input," and "inquiry."

The job of the UP is to interpret the protocol and to generate appropriate device-dependent information so that the image shown on the user's display corresponds to that specified by the SP using the protocol. It also implements the input and inquiry aspects of the protocol. The UP may have many "local options" that are not covered by the protocol. For example, the UP might contain facilities for creating hard copies of the image on the display without engaging in any protocol exchanges.

Essentially the UP can be viewed as either an "interpreter of structured picture definitions" or an "interpreter of transformed picture definitions". In the protocol, the UP tells the SP which kinds of format it can implement. It is perfectly possible for the UP to implement both -- the display images due to each of the formats are merged onto the screen.

These two different kinds of output format can be summarized as follows:

Transformed: The protocol is used to build and modify a set of "segments" (sometimes called "records") of a transformed display file, stored in the user Host. A segment is a list of graphical primitives that specify lines, dots, and text to be displayed at specific positions on the display screen. Individual segments may be deleted or replaced; they may be added to or removed from a list of segments to actually display on the screen (this is called "posting" and "unposting" segments). If a picture is composed of many segments, changes to the picture can often be made by replacing one or two segments; thus segmenting the display file helps to reduce the amount of information that must be transmitted through the network to effect change. Considerable experience with this type of picture definition has demonstrated that device independence can easily be achieved.

One advantage of transformed format is that the UP can be kept very simple; no transformations need to be performed in the User Host. A UP along the lines should be able to be implemented in an IMLAC. The burden of transformation is left to the server Host, presumably a large computer quite capable of being programmed to do transformations.

Structured: The protocol is used to build and modify "figures"; each figure is a collection of "units". A unit may be a list of graphical primitives, such as lines, dots and text; or it may specify a "call" on another figure, together with a transformation to apply to the called figure. The protocol can replace individual units; altering a figure that is called in several places may cause widespread changes to the visible display. The structured format requires (in principle) even less network bandwidth for updates than does the transformed format; many updates may involve changing a single transformation (e.g., to change the viewing transformation for a three-dimensional object).

Although a structured picture definition can be interpreted directly by a few display processors that have transformation ability, most UPs that implement structured format will perform the transformation in software. This implementation is relatively difficult, and certainly requires a fairly powerful user Host computer.

A remaining issue is when the display-generation software is to run in order to update the display. A useful technique is for the SP (and application program) to signal the UP whenever a collection (or batch) of changes is complete and the display should be updated. This technique has the following advantages:

1. Transformation software is executed only when necessary in order to update the display; never needlessly repeating transformations.

2. Screen erasures on storage tubes can be minimized; the screen will be erased a maximum of once per batch of updates, rather once per update.

3. All changes to the display appear "instantly". Network speed means that display-file updates will arrive at the UP over a longish period of time. If the effect of these changes is delayed until a batch is finished, the display will appear to change "all at once", a much more satisfying effect than many slow changes. This is particularly true if an image is being replaced by another image that is a scaled-up version of the original.

To take full advantage of this technique, the application program should specify when the screen should be updated to represent precisely what is specified in the display file. These "end batch updates" commands should precede each request for new user input -- thus the user will see an up-to-date image before formulating his response.

The problem of providing input facilities is even harder than that of providing output facilities. The difficulties are chiefly those of device independence and of adequate performance. The difficulty of achieving device independence is easily described: display hardware can have a large number of very different kinds of input gadgets attached (light pens, tablet and styli, joysticks, knobs, buttons, etc.) that have different properties and different methods of reporting their output (e.g., periodically, on computer demand, or "when something changes"). In addition, operating systems at user sites often enforce restrictions on the use of input equipment in order to avoid undue system degradation. Further, if each input must be shipped to the server Host, processed by the application program and SP, then any display updates shipped to the user Host and processed by the UP before the user sees the response, it would be impossible to use many interactive graphical techniques.

The device independence issue is solved by inquiry: the UP reports to the SP a list of available devices. The SP and application program can then collaboratively arrive at an acceptable set needed for operating the application program. If the set of input devices is insufficient, the application program can perhaps engage in a dialog with the (human) user to seek remedies. Perhaps another version of the UP can be run which implements the required device or, if the application program and SP are sufficiently flexible, perhaps the command language of the application program can be altered dynamically to permit its operation with the available devices. For example, if the UP responds that it has no coordinate input device (and that it is not willing to simulate one with, say two knobs) then the AP might want to use a keyboard-based interaction sequence and to organize the entire command system differently.

The performance difficulties are addressed by permitting the SP to ask the UP to use some particular interactive technique in conjunction with an input device, and to report the results of the interaction. Such interaction techniques are termed "events". The techniques often involve providing "local feedback" so that the user sees the results of his interaction without a long network delay. Examples of local feedback are: displaying a "tracking dot" at the current location of a coordinate input device, or displaying a trail of "ink" behind the tracking dot, etc. Examples of the special "events" that the protocol provides are:

Positioning: providing a pair of x-y coordinates to specify the position of something.

Pointing: giving the coordinates of a displayed object in order to identify it (i.e., associate it with some other object or attribute).

Stroking: reporting a stream of coordinates identifying an arbitrary curve.

Dragging: using a coordinate (or other) device to cause some portion of the the display image to move in synchronism with the coordinate device.

The protocol provides the SP with two basic methods for dealing with input devices: (1) to request and obtain the state of an input device, e.g., the current position of a coordinate input device, and (2) to enable various "events", and to obtain a report describing the "events" resulting from user actions.

The protocol has no set of standard features; there is thus no Network Virtual Graphics Terminal as viewed by the protocol. The inquiry function is used to transmit to the SP a certain amount of detailed information about the terminal in use and the UP that drives it. This information will probably be requested by the SP when a session is initiated.

The information transmitted by the inquiry response is in part for information only. However, some of the information returned is essential in order for the SP to transmit legal protocol to the UP. In outline, the information returned is:

List of implemented protocol commands: This list tells the SP whether the UP implements transformed format, or structured format, or positioned text, or any combination of them, and so forth. In addition, this report tells which optional parts of the protocol are implemented by the UP.

Coordinate information: This information is necessary for the SP to carry out transformations that generate coordinates in the coordinate system used by the terminal.

Parameters: These describe available character sizes, available intensity resolution, available line textures, etc.

A list of available input devices and events: A "device number" is specified for each device; this is used when reading the state of a device. Similarly, an "event number" is specified for each event, and is cited when enabling or disabling it.

An ASCII text string: This describes the terminal, e.g., "IMLAC PDS-1 in room 22".

The information in the inquiry response (that transmitted from UP to SP) that is not essential to further protocol operation may still be useful to the SP in order to drive the terminal intelligently. For example, inquiry can determine the kind of display being used, not so as to send device-specific code to it, but so that the application program does not try to use a graphic technique on a terminal that cannot handle it (e.g., some sort of dynamics on a storage tube).

As may be obvious from the foregoing, the graphics protocol designers were unable to design a reasonable "virtual" device; the variety in available hardware and software is too great, and no class of devices or techniques dominates the graphics field. Thus, as with FTP, there is a little of everything in the protocol. It seems clear that the development of a compact and useful graphics protocol is still very much a subject for research.

2.6 Performance (incomplete section)

In this section, we discuss the many aspects of ARPANET performance.

2.6.1 Introduction

explanation of terms

short history

2.6.2 Theoretical Models, Measurements, Simulations

Kleinrocks' model

IMP measurement package/UCLA work

various experiments and measurement runs

2.6.3 Delay

In this section we consider what delay performance characteristics are possible in a given network. The topics discussed below include the identification of the components of delay, an analysis of the minimum delay possible for a round trip through a network, the reason for breaking messages up into packets, a brief look at queueing delay, and a discussion of delays for interactive traffic.

Components of Delay. We will first discuss the delay experienced by a single packet transmitted over a single hop. The components of delay which we will take to be fundamental variables are as follows:

1. L = propagation delay or speed of light latency.
This is the delay for the first bit of the packet to traverse the circuit.

L = (circuit length [mile])/(signal propagation rate [mile/sec]).

2. T = transmission delay.
This is the time for the bits of a packet to be clocked out on the circuit.

T = (number of bits in packet [bits])/(transmission rate [bits/sec]).

In the analysis below we will use Tp to denote T for a packet and Tr to denote T for a RFNM, which we will assume is a minimum length packet.

3. C = nodal processing delay.
This is the time it takes the node to process the packet. It has two fixed components, corresponding to the store operation and the forward operation, or receive and transmit, plus a random component due to queued tasks of higher priority. This component measures the interference experienced by packets queued for processor service. (In the following, we ignore delays at the source and destination nodes due to message processing.)

C = Cr[sec] + Ct[sec] + Cq[sec].
receive transmit queueing = random

A typical range of C is probably 1 to 10 milliseconds for an IMP-like node.

4. Dq = queueing delay
This is the time that the packet must wait for the transmission of the packets which precede it on the output queue, including

the output time of the packet currently being transmitted. Thus, Cq measures input queueing delay, waiting for central processor service, while Dq measures output queueing delay, waiting for circuit service.

5. Dr = delays due to retransmissions.

This is the time that the packet must wait in the event that its first transmission is unsuccessful. This may happen if it was in error or if the other node refused it for some reason, or, in the case of broadcast circuits, there was a collision with another packet.

In general, we will assume that the first two components are much greater than the last three. Tables 1 and 2 give some representative values for L and T. (Note that a 50 Kbs circuit can transmit 1000 bits in 20 ms, and that the first bit can travel about 4000 miles in that time. So a bit is about 4 miles long at 50 Kbs!)

Minimum Round Trip Delay. Now we can examine the minimum round trip delay, by taking the case of Cq = Dq = Dr = 0. Consider a message with P packets traveling over a path of H hops. If the delay at hop i is

D(i) = L(i)+T(i)+C(i)

we can define the natural quantity D(ave), average hop delay, as follows:

D(ave) = (1/H)*zzzz(L(i)+T(i)+C(i)), (1)

We can also define the less obvious variable D(max):

D(max) = zzzz(T(i)+Ct(i)), (2)

With these definitions, we can make the following two statements:

1. The first packet experiences delay equal to H*D(ave).

2. The remaining P-1 packets follow through the network, each packet at most one hop behind the preceding packet, and these packets add (P-1)*D(max) to the total delay.

This analysis can be illustrated by the following numerical example:

P = 3

H = 3

Ct = Cr = 0.5

HOP

1 2 3

L 0 1 3

T 2 3 2

Total delay = 21, as shown in Figure 2-35a.

Note that the bottleneck hop, in this case hop 2, has the largest T(i) + Ct(i), here equal to 3.5, but not the highest total delay. That is, hop 3 has total delay of 6, compared to the delay of 5 over hop 2. Note also that more than one packet is being transmitted at the same time, giving a pipelined effect, and reducing the total delay for the message.

This means that the delay for a single packet message is

D(SP) = zzzz(L(i)+Tp(i)+C(i)). (3)

and the delay for a multipacket message is

D(MP) = (P-1)*Max[i=1,H](T(i)+Ct(i)) + D(SP). (4)

The delay for a RFNM is

D(RFNM) = zzzz(L(i)+Tr(i)+C(i)). (5)

Therefore, the minimum round trip delay for a message of P packets over H hops is

Figure 2-35 Example of Delays

D(MRT) = zzzz(2*L(i)+2*C(i)+Tp(i)+Tr(i)) (6)

+ (P-1)*Max[i=1,H](Tp(i)+Ct(i)).

If the values of L, C, Ct, and T are the same for each hop, then we have a simplified minimum round trip delay

D(MRTS) = H*(L+Tp+C) + (P-1)*(Tp+Ct) + H*(L+Tr+C). (7)

first packet subsequent packets RFNM

Some curves are given in Figure 2-36 which illustrate the minimum round trip delay through a network for a range of message lengths and path lengths, for two sets of line speeds and lengths over paths of 1 to 6 hops.

The Rationale for Packetizing Messages. As a slight digression, we now consider the rationale for breaking up messages into packets in order to reduce delay. It will be shown that this rationale is similar to that for any pipelining technique, and the variables of interest will be identified. We will carry through this discussion for the simplified case of identical values of L, C, Ct, and T at each hop. First we rewrite equation 7 as

D(MRTS) = (P-1)*(Tp+Ct) + H*(2*(L+C)+Tp+Tr)

Figure 2-36 Min Round Trip Vs. Message Length

Note that the only one of the variables L, C, Ct, and T which is a function of packet length is T. Let us suppose that instead of a message of P packets, each of which has a transmission delay of T, the message is sent as a whole, with a transmission delay of P*T. The total delay can be calculated as above, considering this new entity as a long single-packet message. Using the equation for D(MRTS), we get an equation for minimum round trip delay, simplified , not packetized

$$D(MRTSNP) = H*(L+P*Tp+C) + H*(L+Tr+C) \qquad (9)$$

We can rewrite this as

$$D(MRTSNP) = (P-1)*H*Tp + H*(2*(L+C)+Tp+Tr) \qquad (10)$$

Subtracting equation 8 from equation 10 we get a difference in delay

$$D(MRTDIF) = (P-1)*((H-1)*Tp-Ct) \qquad (11)$$

We make the following observations:

1. Clearly, if P=1, D(MRTDIF) = 0
There is no difference between the two techniques for single-packet messages.

2. Assuming P>1, if H=1, D(MRTDIF) = -(P-1)*Ct < 0

That is, if H=1 and P>1, it involves more delay to break a message into packets because of the added processor overhead (generally small).

3. Assuming Tp>Ct, that the processor time is small, if P>1 and H>1, then D(MRTDIF) > 0.

This is the usual case, and constitutes the basic reason to break messages into packets.

We can return to the numerical example used earlier with the equivalent conditions for a single packet message as long as the 3 packet message considered above:

P = 1

H = 3

Ct = Cr = 0.5

HOP				
	1	2	3	total
C	1	1	1	3
L	0	1	3	4
T	6	9	6	21
	--	--	--	--
total	7	11	10	28

Total delay = 28, as opposed to 21 for the packetized case (see Figure 2-35b).

Queueing Delay. We have examined the minimum round trip delay as a function of message length, network path length, and the packetizing strategy. It is appropriate at this point to analyze the effects of additional delays which may be present. To perform the minimum delay analysis, we made 3 assumptions, each of which should be re-examined at this point:

1. Cq = 0. The packet arriving may have to wait on an input queue before it is serviced by the processor. This time is generally quite small, but it is a random variable which may take on large values if there are time-consuming high-priority tasks in the system.

2. Dq = 0. This is the major assumption which must be modified in the analysis of actual delays in a network. A great deal of theoretical work has been done in studying queueing delay, particularly by Kleinrock, who has used both analysis and simulation in this regard. For the simplified discussion at hand, it is sufficient to note that each packet on the queue adds an additional T+Ct to the delay of all packets behind it on the queue.

3. Or = 0. The accuracy of this last assumption varies widely with the type of network and line being considered. For some wideband circuits, particularly satellite channels, the error rates are very low and few retransmissions may be necessary for reasons of errors. However, satellite links may be used in a broadcast competition mode so that some packets are lost in collision with others. Finally, there is always some chance that the adjacent node will refuse to accept a packet for lack of processing resources.

Low Delay for Interactive Traffic. Perhaps the most important consideration about delay in a network is this: some traffic consists of interactive, high-priority messages and this traffic must be delivered to its destination as rapidly as possible. This is in contrast to bulk transfer traffic which is not so delay-critical. The most obvious case of such interactive traffic is most man-computer dialogue, which consists of rather short messages between the computer and a man at a terminal. Here there is a definite threshold for delay. Below this threshold, delay is acceptable, and above it delay is unacceptable. There is no added benefit if the delay is considerably below the threshold, and it is likely that once the delay is much above the threshold, almost any value of delay is equally unacceptable.

Another way of looking at the bimodal nature of network traffic is to consider that much of the delay for an interactive message is in the network itself. That is, it is generated and quickly sent into the network. When it is delivered, it will be processed quickly at the destination Host. Bulk transfers on the other hand may experience lengthy delays outside the network due to buffering considerations and the very size of the data (secondary storage or tapes or cards may be involved in the data transfer, greatly increasing delay).

2.6.4 Throughput

In this section we consider what throughput performance characteristics are possible in a given network. The topics discussed below include an analysis of nodal processor bandwidth, an examination of circuit overhead, a quantification of the buffering in packets required for a network line and the buffering in messages required for a network path, the throughput requirements of bulk transfer traffic, and the tradeoff between delay and throughput.

The Effective Bandwidth of the Node Processor. We will first examine the effective processing capability of the node computer in a network with variable message length. Some numerical examples will be given to support the intuitive notion that the processor is most effective for long messages, given some very general assumptions about the packet processing involved.

We begin by defining some new quantities of interest in studying throughput in networks. The quantities which we will take as fundamental variables are as follows:

1. Bd = the number of host data bits in a packet.

2. Bs = the number of software overhead bits per packet. These bits include header information such as address, and identifying information such as message number.

3. Bh = the number of hardware overhead bits per packet. These are typically framing bits for the circuit, and error detection bits such as checksums or redundant information.

4. P = the number of packets per message, as above.

5. Btot = the total number of host data bits per message.

Btot = P*Bd - unused bits in the last packet.

Now we will examine how long it takes the node processor to store and forward a message. As noted above, we ignore the processing at the source node and at the destination node. The time required to process a store-and-forward message is a function of the following parameters:

6. C = the packet processing time, as above.

We assume that C is independent of packet length or type. To the extent that this is not true, the length-dependent component of C can be accounted for in item 8 below.

7. BWPo = the fraction of the bandwidth of the processor taken by overhead.

Due to certain necessary periodic processes within the node, notably the routing computation, effective processor bandwidth is reduced.

8. BWIO = the I/O rate of the node in bits/sec.

We assume here that BWIO is a linear function of the number of bits in the message. In most I/O architectures, it is probably a function of the number of computer words in the message, which is identical apart from unused bits in the last word. We are also assuming that the I/O transfer steals cycles from the processor, reducing its effective bandwidth.

9. I = the I/O transfer time in seconds. We will denote the I/O time for a packet, a message, and a RFNM as Ip, Im, and Ir respectively:

Ip = 2*(Bd+Bs)/BWIO

Im = 2*(Btot+P*Bs)/BWIO

Ir = 2*Bs/BWIO

10. MT = the total time taken to process a message.

MT = C*(P+1)*(1+BWPo) + Im + Ir.

11. BWPd = the maximum data bandwidth that the node can support.

BWPd = Btot/MT.

This the the number of host data bits per second that the node can process.

12. BWPl = the maximum line bandwidth that the node can support.

BWPl = (Btot +(P+1)*(Bs+Bh))/MT

This represents a processing capability limit on the number and speed of the circuits that can be connected to a node. The difference between the two quantities, BWPl-BWPd, is a measure of the line overhead at a given message length.

At this point, a numerical example may be illustrative. In the ARPANET the processor bandwidth of the IMP, both BWPd and BWPl, can be plotted as functions of the message length, and some results are given in Figure 2-37. (The discontinuities indicated in the curves are the result of packetizing messages.)

Figure 2-37 Processor Bandwidth vs. Message Length

The Effective Bandwidth of the Network Circuits. In this section we will consider some of the factors acting as overhead to reduce the effective circuit bandwidth for network lines. That is, we wish to catalog all the kinds of transmission that take place on the network lines that are not actual host data bits, and from this accounting determine which factors are the dominant ones. There are three basic kinds of line overhead:

1. Line overhead in bits/packet.

We have detailed two components of this overhead earlier, Bs and Bh.

2. Line overhead in bits/message.

In the terminology we have been using, the RFNM is overhead on a message basis, and under our assumption that it is a minimum length packet, it contributes a number of bits equal to Bs+Bh.

3. Line overhead in bits/second of network system traffic.

Some examples of this kind of traffic can be given, along with the approximate order of magnitude for the traffic rate in the ARPANET:

routing	1000 bits/sec
line alive/dead	100 bits/sec
NCC status reports	10 bits/sec
core reloads and dumps	1-10 bits/sec

We will make the assumption that routing messages are the primary source of traffic other than messages between hosts. For this reason, we will define the variables

Br = the number of bits in a routing message

Fr = the frequency of routing messages (1/sec)

$BWCr = Br*Fr$ = the bandwidth of the circuit used for routing

and ignore the other components of periodic overhead.

We can now total the overhead from all considerations, by converting to a common dimension, bits/sec. In order to do this, we must introduce another variable,

Fp = the number of packets per second.

We then can express the total bandwidth of the circuit given to overhead as

$$BWCo = Br*Fr + (Bs+Bh)*Fp + (Bs+Bh)*Fp/P$$

routing packets RFNMs

We can rewrite this as

$$BWCo = BWCr + (Bs+Bh)*Fp*(P+1)/P$$

In comparison with this overhead rate is the actual data rate on the circuit,

$$BWCd = Bd*Fp$$

We can evaluate the fractional overhead percentage as

$$BWCo/BWCd = (BWCr)/(Bd*Fp) + (Bs+Bh)*(P+1)/(Bd*P)$$

Another quantity of interest is the maximum data rate that can be attained for given values of the systems parameters. Given a circuit with bandwidth BWC, the maximum data rate occurs when Fp is at a maximum,

$$Fp(Max) = (BWC-BWCo)/Bd$$

Substituting the expression for BWCo above, we get

$$Fp(Max) = (BWC-Br*Fr)/(Bd + (Bs+Bh)*(P+1)/P)$$

The numerator indicates that the routing message bandwidth comes off the top, leaving a reduced effective bandwidth, which is then used for both data bits and packet and message overhead bits. The graphs in Figure 2-38 show the behavior of some of these variables; the variable plotted is Fp(Max)*Bd, that is BWC-BWCo, which is the maximum data rate for a given message size.

The Packet Buffering Required for Network Circuits. We now turn to an examination of the number of packet buffers required to keep a communications circuit fully loaded. This number is a function not only of line bandwidth and distance but also of packet length, nodal delays, and acknowledgment strategy. We will assume that the node buffers each packet that it transmits until it receives an acknowledgment, meanwhile transmitting other packets to utilize the circuit efficiently. If it does not receive the acknowledgment in the expected time, it retransmits the packet. We also assume that packet buffers are a fixed size large enough to hold the largest packet. The expected time for an acknowledgment to return is the sum of:

1. Tp = the transmission time for the packet, a function of line bandwidth.

2. L = speed-of-light delay for the first bit of the packet to arrive at the other node, a function of line length.

Figure 2-38

3. C = Cr+Ct = the processing delay in the other IMP, to receive the packet and return the acknowledgment.

4. Dq = queueing delay for the returning acknowledgment, the time it waits for any other transmissions ahead of it.

5. Ta = the transmission time for the acknowledgment.

6. L = speed-of-light delay for the first bit of the acknowledgment to arrive at the first node.

7. Cr = the processing delay for the acknowledgment.

Our first simplifying assumption is that the processing times are small compared to the other delays and can therefore be ignored:

Cr = Ct = 0.

We can then state that the minimum number of packet buffers needed to keep a circuit fully loaded is

BFp = (Tp + L + Dq + Ta + L)/Tp

This can be rewritten in somewhat more meaningful form as

BFp = 1 + 2*L/Tp + (Dq+Ta)/Tp

This expression indicates that one buffer is always necessary, to account for the packet transmission time itself. More buffers may be required if the circuit is long compared to the packet transmission time, or if the acknowledgment transmission takes a long time compared to the packet. Stated differently, the number of buffers needed to keep a line full is proportional to the length of the line and its speed, and inversely proportional to the packet size, with the addition of a constant term.

In order to proceed further with the analysis, we need to introduce two new terms:

Ts = the transmission time of the shortest allowable packet

Tl = the transmission time of the longest allowable packet

We also need to postulate a traffic mix of long and short packets, with x/y the ratio of short packets to long packets in the channel. Now we can define Dq and Ta in terms of Ts and Tl. We make a worst-case assumption for Dq:

Dq = Tl, the acknowledgment has highest priority (equivalently, it piggybacks on all packets), but it must wait for the transmission of a maximum-length packet which has just begun.

The assumption for Ta is rather arbitrary:

Ta=(Ts+Tl)/2, the acknowledgment piggybacks on an "average" length packet.

We now state the result for the number of packet buffers required given the above set of assumptions:

BFp = 1 + (2*L + Tl + (Ts+Tl)/2)/Tp

Using the ARPANET values of Ts and Tl given above in Table 2, and choosing a variety of line lengths and traffic mixes (shown as the ratio of short packets, S, to long packets, L), we can present some numerical results as a family of curves shown in Figure 2-39. Note that the knee of the curves occurs at progressively shorter distances with increasing line speeds. In fact, if we define the knee to occur when the linear term is equal to half of the constant term, then the knee occurs when

L = (Ts + 2*Tp + 3*Tl)/4,

or for a line length of 225*10**6/BWC miles. The constant term dominates the 9.6 Kbs case, and it is almost insignificant for the 1.4 Mbs case. Note also that the separation between members of each family of curves remains constant on the log scale, indicating greatly increased variations with distance.

Figure 2-39

The Message Buffering Required for Network Paths. This section takes up a topic which closely parallels that of the last section. Here we will examine the number of messages needed to obtain full bandwidth over a network path of many lines. That is, we will compute how many messages must be in flight between two nodes in order to keep all the intermediate lines fully loaded. Actually, the best one can do is to keep all the lines in the path of the lowest circuit bandwidth fully loaded. It turns out that this analysis is quite simple given all the definitions of the preceding sections. The number of message buffers needed is computed by taking the round trip delay for a message and dividing it by the time taken to transmit a single message. That is,

$$BFm = D(MRT)/(P*(Tp+Ct))$$

Using equation 6 in section 2.2.1, we can obtain a more detailed expression for the simple case of equal delay at each hop:

$$BFm = (P-1)/P + H*(2*(L+C)+Tp+Tr)/(P*(Tp+Ct))$$

It is clear from this expression, and on intuitive grounds, that BFm is a minimum for maximum length messages, that is for large P. The curves in Figure 2-40 show the dependence of BFm on line characteristics and on the length of the network path, for P=8

Figure 2-40

and ARPANET values of the parameters. For each of four line speeds, the buffering requirements are plotted for network paths made up of a number of land lines (the length of the lines is given with each curve). Also shown are the requirements for the same network paths with the addition of one satellite link running at the same bandwidth as the land lines.

The consequences of the above discussion are several. For the communications subnetwork, it means that the nodes must be able to do bookkeeping on several messages in flight between two nodes. Further, the network must buffer these messages in the memory of the source or destination node for the duration of their flight, in addition to the packet buffering that takes place instantaneously at the intermediate nodes along the path. This has some important ramifications for the design of the software for the node computer. A second set of issues is the effect of message buffering on host computers. It is clear that the communication protocols that the hosts use must also be engineered to the parameters of the network, if they are to obtain full throughput levels.

<u>High Throughput for Long Data Transfers.</u> The several topics examined in this section all point to a single conclusion: the larger the packets in a message, and the larger the messages in a

data transfer, the higher the level of throughput that is potentially attainable. For reasons of processor overhead, circuit overhead, and buffering considerations within the nodes, it is always better to have long packets and messages if high data rates are desired.

The Tradeoff Between Low Delay and High Throughput. It is clear that the two goals of low delay and high throughput are often in conflict because the node has limited resources with which to service its hosts and lines. It is difficult to guarantee both low delay and high throughput to several competing sources. The approach taken in the ARPANET is as follows: the IMP program has been designed to perform well under bimodal traffic conditions. It provides quick delivery for short interactive messages and high throughput rates for long files of data. This optimization of the program for a specific model of traffic behavior occurs at many levels, and is essential for balanced performance characteristics. (Other conflicts exist, such as the conflict between high throughput and congestion/deadlock control mechanisms.)

2.6.5.1 IMP Techniques

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

A dead line is detected by the sustained absence (approximately 3.2 sec) of IHY messages on that line. No regular packets will be routed onto a dead line, and any packets awaiting transmission will be rerouted. Routing tables in the network are adjusted automatically to reflect the loss. Receipt of consecutive I-heard-you packets for about 30 seconds is required before a dead line is defined to be alive once again. The IMP program takes into account the fact that an IMP may have only one working line to the network. In this case, a line may make twice as many errors as is usually permitted before it is declared unusable. This mechanism is an attempt to improve network availability from singly-connected sites.

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

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

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

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

The IMPs use the technique of software checksums on all transmissions to detect errors in packets, protecting the integrity of the data and isolating hardware failures. The end-to-end software checksum on packets, without any time gaps, works as follows (refer to Figure 2-41).

Figure 2-41 Software Checksums

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

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

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

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

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

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

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

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

It is useful to provide a brief look at some of the problems encountered in the ARPANET with the reliability of routing. In 1971, there was an IMP which crashed every few days in such a way as to cause all the other IMPs in the network to malfunction as well. It was finally determined that its core memory was faulty and sometimes the routing messages read out from memory by the modem output interfaces were all zeroes. The adjacent IMPs interpreted such an erroneous message as stating that the IMP in question had zero delay to all destinations - that it was the best route to everywhere! Once this information was propagated to the other IMP's, the whole network was operating with incorrect routing information, and normal operations came to a halt. The problem here is that the information that one node intended to transmit was not successfully received by another node.

A different problem happened in 1973, with similar symptoms of the whole network being affected. It was traced to an IMP having an incorrect instruction, due to a memory failure, in the middle of its routing program. This had the effect that the IMP computed an incorrect pointer to where it thought the input routing data was in memory, and so it proceeded with the routing computation on the basis of random data, and then propagated this information to the adjacent IMPs. This meant that the routing throughout the network was continuously changing, oscillating wildly, as the failed IMP sent out the incorrect routing data. Here the problem was that the node was not running the correct routing process, but it too had global effects.

A final problem happened when one IMP, at Aberdeen, suffered a dropped bit in memory in its route directory. The entry in the directory for the affected destination IMP, UCLA (many hops and 3000 miles away), was then zero. A zero entry in the directory at an IMP means that the entry is for the IMP itself. Thus, the Aberdeen IMP took all traffic for the UCLA IMP as traffic for itself. The dropped bit in the Aberdeen IMP meant that part of the network could not communicate with the UCLA IMP, though there were no failures in that part of the network or in the UCLA IMP. Note that the route directory is a completely internal table; it is not exchanged with the adjacent IMPs, and yet a failure in this table had global effects.

There is one lesson to be drawn from such incidents:

> *The routing algorithm is extremely important to network reliability, since if it malfunctions, the network is useless. Further, a distributed routing algorithm has the property that all the nodes must be performing the routing computation correctly for the algorithm to be reliable. A local failure can have global consequences.*

Having described some of the problems that can arise in the course of actual network operations, we now turn to a more systematic evaluation of the problem areas and possible solution techniques.

In attempting to catalog the kinds of errors that can be introduced into routing data, it is useful to recall the system components that may cause the errors, and how these components fail. The processor, memory, I/O interfaces, and the circuit between nodes are all hardware components that may have either a solid or intermittent failure, and the routing program or the routing data may be affected. In transferring a set of data from one node to an adjacent node, there are the following possibilities for error:

1. Incorrect data stored in memory at the source: The program may store it in the wrong locations in memory; the memory may fail at the data locations.

2. Data changed between creation and transmission: The program may mistakenly change some of the data, due to hardware or software failure; the memory may fail.

3. Incorrect transfer of data from memory to line interface at the source: The program may direct the hardware to send the data from the wrong locations; the hardware may make the data transfer with errors.

4. Incorrect transmission of the data over the circuit: The circuit may be noisy, and make data errors; the circuit may be unusable.

5. Data changed between reception and use: The program may mistakenly change some of the data, due to hardware or software failure; the memory may fail.

6. Incorrect transfer of data from the line interface to memory at the destination: The program may direct the hardware to read the data into the wrong locations; the hardware may make the data transfer with errors.

7. Incorrect data read from memory at the destination: The program may read the wrong locations in memory; the memory may fail at the data locations.

We now explain briefly the measures taken in the ARPANET to protect against the failures cataloged above. The central aims of the approach taken in the ARPANET are:

1. To detect errors of any kind that may affect the routing computation.

2. To perform error checking often enough and in enough places to provide fault isolation in the event of failure.

3. To report as much data as possible about the failure to the Network Control Center for later diagnosis.

4. To allow the node which detects a failure to continue to function, whenever possible. This includes taking remedial action as appropriate.

5. Most important, to localize the effects of a failure by ensuring that incorrect routing information is never exchanged between nodes.

The basic technique used in the ARPANET to improve the reliability of the IMPs and enhance their ability to provide diagnostic information is checksumming. The original ARPANET design included checksum hardware in the modem interfaces, to detect errors in packets due to noisy circuits. This concept has been extended greatly as follows:

1. Routing messages carry a special checksum (computed in software in the present IMP, but there will be special hardware for this purpose in the next generation IMP) which is distinct from the checksum on transmissions over the circuits.

a. The process which constructs each routing message builds the checksum. The checksum refers to the intended contents of the message, not the actual contents. That is, the program which generates the routing message builds its own checksum as it proceeds, not by reading what has been stored in the routing message area, but by forming the checksum of the intended contents for each entry as it computes them. This must be done in software, though the next two can be done in hardware.

b. The process which sends out routing messages always verifies that the checksum is correct before it transmits a message. This detects intra-node failures such as memory failures, processor failures, and random changes made to the data.

c. The process which accepts routing messages always verifies that the checksum is correct before marking it as acceptable data. This detects inter-node failures such as memory transfer problems, interface noise, and so on.

2. All routing programs are checksummed before every execution, to verify that the code about to be run is correct. The checksum of the program includes the preliminary checksum computation itself, the routing program, any constants referenced, and anything else which could affect its successful execution. These programs include:

a. The process which constructs routing messages.

b. The process which transmits routing messages.

c. The process which accepts routing messages.

d. Any periodic processes which affect the routing data.

In each case, if an error is detected, the program is immediately reloaded from an adjacent node, and a diagnostic message is sent to the NCC indicating the kind of failure.

3. Experience in Use and Operation

The actual network, the objectives and design of which have been discussed in such detail in the preceding sections, has changed during its lifetime from a computer communications research vehicle to an operational system supporting a wide variety of diverse users, pursuing independent computational goals in their daily work. This section briefly discusses some of the communities of interest which have been involved in ARPANET utilization, some of the novel systems which have been built to take advantage of network facilities, and some of the ongoing communications experiments which take the existence of the ARPANET as a base upon which to build. The section concludes with a discussion of the operation of the network, since the network operation and maintenance is heavily reliant upon the network itself.

A discussion of the Host experience with the ARPANET should surely begin with mention of some of the first network uses. Undoubtedly the very first use of the network for a task which was not some aspect of monitoring the network itself was the conversion of the Stanford Research Institute (SRI) On-Line System (NLS) software from an XDS-940 to a PDP-10. Conversion began several months in advance of the delivery of the DEC PDP-10 system to SRI with the modification of the XDS-940 compiler to produce PDP-10 object code. By early 1970 this modification had been performed and object code was ready for checkout. The initial checkout was carried out by sending the object code through the ARPANET to the PDP-10 system at the University of Utah, using a precursor of the standard Host-Host protocol. The code was executed at Utah and a record of its execution history (traces, console outputs, etc.) stored in the file system for later transmission back to SRI. This network use was important both because it showed an early, significant, payoff from the investment in the network, and because it demonstrated the feasibility of communication between dissimilar Host computers. In fact, it should be noted that there was a PDP-10 computer installed elsewhere within SRI at the time this network use was taking place, but it proved to be much more convenient to use the Utah PDP-10 because both it and the SRI 940 were connected to the ARPANET, while the SRI PDP-10 was not.

Several early "demonstration project" uses of the ARPANET were documented in mid-1970 [Carr70] and early 1971 [Metcalfe71]. These included a number of different examples of terminals directly connected to one Host being used to access a timesharing system on some other Host (e.g. SRI XDS-940 to Utah PDP-10, UCLA Sigma-7 to SRI 940, PDP-6/10 to Multics at MIT, and MIT PDP-6/10 to Harvard PDP-10). A more ambitious experiment involved sending graphics programs and 3D data from Harvard's PDP-10 to the MIT Dynamic Modeling/Computer Graphics complex which included an Evans & Sutherland Line Drawing System, a quite fancy graphics processor. The data was processed repeatedly on the E&S system and 2D scope data was returned to Harvard's PDP-1 for display. This setup was used to run a simulated landing of an airplane on an aircraft carrier under control of user console input data. The importance of these experiments was that they demonstrated the ability of the network to support interactive traffic without excessive delays, although the aircraft carrier simulation was bandwidth limited. It was noted in [Metcalfe71], in a description of the Harvard/MIT console connection, that:

> "...it was found that the response from the Harvard system at MIT-DM/CG was seemingly as fast as could be expected from one of their own consoles. This fact is particularly exciting to those who don't have a feel for network transit times when it is pointed out that such response was generated through two

time-sharing systems, three user level processes, and three IMPs, all connected in series."

Experiments such as these provided the impetus needed to hasten the development of protocols (and NCPs), while also providing valuable tests of some of the original protocol concepts and of the subnetwork of IMPs.

Since the time of these experiments network use has increased tremendously. The following sections briefly describe some of the details of this use. It should be noted that these sections draw heavily on the facts and concepts presented in the cited references, and in some cases incorporate text as published in the references without explicit indication.

3.1 Server Systems

It has become common when discussing the ARPANET Hosts to categorize them as either "servers" or "users" depending, obviously, upon whether they tend to provide services to other sites or to provide access to the network so that local people can access remote services. Of course, many server systems also provide an access path for local users to reach the ARPANET, so that there are almost no pure server systems (e.g., the Datacomputer and the ILLIAC IV). Similarly, there are few user systems which offer no services, other than the TIPs. However, most systems can be fairly unambiguously categorized as servers or users.

It is worth noting that size alone does not determine whether a system is a server or a user. For example, the PDP-11 based Seismic Input Processor is a pure server system which acts as buffer between continuous low speed inputs and occasional high speed transfers to the Datacomputer. On the other hand, the CDC 6500 at Fleet Numerical Weather Central is an ARPANET user system which is used only to gain access to remote network services.

Among the network server systems there is quite evident functional specialization, with surprisingly little competition within each specialty. Thus, it is appropriate to think of the network servers as a complementary set of resources, among which choices are made on the basis of the job to be performed. The primary exception to this situation is in the area of general purpose time-sharing with an emphasis on text handling. In this field there is some competition between the various PDP-10 systems on one hand and the Multics systems on the other; there is also competition among the various PDP-10 centers, notably the TENEX centers at BBN and ISI.

Some of the network server systems were in existence (or, at least, were coming into existence) prior to the construction of the ARPANET and are "incidentally" connected to it; such systems include the UCLA Campus Computing Network (CCN) IBM 360 Model 91, the MIT Multics system, and the BBN Research Computer Center (RCC) TENEX systems. Others, such as the Datacomputer, included ARPANET connection as a relatively important part of their planning. (In addition, although its development was started well before the ARPANET, the planning for access to the ILLIAC IV came to rely increasingly on the ARPANET.) Thus, some servers were relatively indifferent to whether they attained significant outside usage or not, some had a ready-made "captive" network user community, and some actively sought new users. Further, some of the centers were run on a strictly economic basis, while others were as much (or more) influenced by research goals as by economics. For all these reasons, the server organizations' interfaces to the users varied widely, from eager helpfulness through relative indifference.

Regardless of the attitude of the server organizations, however, none of them had significant prior experience dealing with a user community spread over a very wide geographic area. The response modes which had been developed to provide user services on a campus, or within a few office buildings of a single company, or within a single city and its suburbs proved inappropriate to a user community spread across the entire continent (not to mention England, Norway, and Hawaii). In particular, the mode of hardcopy documentation which assumes a computer center bulletin board where "recent changes" are posted, a consultant's office nearby, and other system users in nearby offices, proved to be unworkable.

One of the first issues to be addressed by the ARPANET community was the question of how to discover what programs and databases existed at what servers. An "ARPANET Resource Handbook" was generated at BBN and later taken over by the Network Information Center (NIC) at SRI. This handbook is loosely analogous to the "yellow pages" published by the telephone company, providing a single place where the providers of service can announce what is available. The Resource Handbook is a logical companion to the ARPANET Directory, also produced by the Network Information Center, which is analogous to the telephone company's "white pages", a listing of how to contact individuals by address, telephone, and (more importantly) via "network mail" (see Section 3.4).

The Resource Handbook, however, only provides pointers to existing resources; it doesn't provide much help in learning to use them, nor does it help in case of difficulty. The server systems have generally adopted similar strategies for dealing with these continuing user-assistance issues; strategies which are extensions of ideas which originated in local timesharing systems. They include the concepts of "linking" two terminals (e.g., a user and an operator), "mailboxes" where requests for help can be left and answers collected, "system messages" sent to the console of each user, and on-line documentation.

In addition, some common strategies have been developed for the cases when human interactions are desirable. At the urging of the NIC, each site nominated a "Technical Liaison", competent to provide answers to system questions to a fairly detailed level, whose name was publicized in both the Resource Handbook and the ARPANET Directory. Most server sites also expanded their ability to provide consulting via telephone; some sites, including the Network Information Center, provided toll-free voice telephone service for user consulting. Some of the servers, notably CCN, also encouraged user visits to the server site at least once, and provided periodic staff visits to the larger users [Kehl73].

3.1.1 UCLA - Campus Computing Network

The Campus Computing Network (CCN) of UCLA has been operating a large server system on the ARPANET since mid-1971. CCN operates an IBM 360/91 CPU with 4 megabytes of high-speed memory and a large secondary-storage configuration. For a number of years, this was the largest and fastest general-purpose computing system on the ARPANET.

As an ARPANET server, the 360/91 has been primarily used for large-scale numerical computation or "number crunching", in a batch-processing mode. The CCN system also provides TELNET access to the IBM general-purpose timesharing system TSO and implements the File Transfer Protocol; both of these have been most useful as adjuncts to batch processing. In addition to its computational power, the CCN system offers users an extensive application program library encompassing a wide variety of fields, many of which are unavailable elsewhere on the ARPANET.

CCN's chance to obtain a connection to the ARPANET was a result of the presence at UCLA of Professor L. Kleinrock and his students, including S. Crocker, J. Postel, and V. Cerf. This group was not only involved in the original design of the network and the Host protocols, but also was to operate the Network Measurement Center (NMC). For these reasons the first delivered IMP was installed at UCLA, and ARPA was thus able to easily offer CCN the opportunity for connection.

CCN reportedly had several interests in becoming an ARPANET Server System, including:

1. The opportunity to become involved in an important new aspect of computing technology, helping CCN to obtain a lead in technical expertise.

2. The prospect of additional income, coming at a time when the recession was causing cutbacks of federal and state funding, but an early point in the 360/91 life cycle with only about 50% loading generated locally. Further ARPANET users were expected to need batch number-crunching, an ideal load for CCN because of its characteristics of being CPU-bound and suitable for off-shift scheduling.

3. The expected local benefit to students and faculty of having new application packages and programs installed on the 360 which would not otherwise be available.

4. The benefits to local users of access to remote resources, both computational and human, especially from the point of view of becoming linked into a major research community.

5. The intangible but real benefit of the prestige accruing to sites associated with the ARPANET.

Of course, many of these reasons pertain to other server systems as well.

Although ARPA provided the UCLA IMP and paid for the necessary CCN interface hardware in order to encourage the availability of number-crunching service, CCN incurred significant costs in developing the necessary Host software. These costs could only be recovered by attracting new users. Thus, CCN was especially interested in seeing that the ARPANET protocols developed to support the types of service CCN offered were efficient, reliable, and easy to implement as well as being specified early. This frequently led to conflicts with other groups involved in the design of protocols, who may have exhibited more of a desire for elegance and generality, even at the cost of more difficult implementation and less efficient operation. The CCN representatives in the protocol design process felt that fundamental differences in approach arose from the "paper tape" orientation of most of the designers (the XDS 940, PDP-10, TENEX tradition) as opposed to the "unit record" orientation of the IBM systems. For example, this difference made development of a usable File Transfer Protocol very long and difficult, and the result was never fully satisfying to anyone. The "paper tape" machines generally used ASCII encoding and character-at-a-time echoplex operation of terminals, compared to the EBCDIC code and line-at-a-time operation of the 360/91. Finally, most ARPANET designers and systems programmers were heavily oriented towards interactive service, while CCN primarily offered batch processing.

A result of the dominant orientation of the ARPANET community toward interactive computing was that CCN had difficulty providing user access to its batch processing capability. A number of potential user groups might have accessed CCN through the ARPANET if they could have dialed in with their batch job entry terminals. However, neither the common user Hosts (e.g., TIPs) nor the majority of the other service Hosts provided access for the synchronous types of batch terminals which are in common use. One solution to this problem was to make the IBM time-sharing system TSO available via TELNET, allowing remote users to create job stream files at CCN and submit them for batch processing. A second solution, for a user Host system, was to program a NETRJS user process (see Section 2.5.5). This happened slowly, but eventually seven different Host systems implemented NETRJS user programs, as shown in Table 3-1. However, since NETRJS was a "non-standard" protocol, CCN never felt entirely comfortable in basing its user services upon the implementation of NETRJS at other Hosts.

With the exception of the official RJE protocol and graphics, CCN attempted to implement completely every user-level protocol. An outstanding example is CCN's File Transfer Protocol (FTP) Server, which is the most complete implementation of that

SITE	OPERATING SYSTEM
RAND	OS/MVT on 370/158 (originally on 360/65)
UCLA-NMC	SEX on Sigma 7
Illinois	ANTS on PDP-11
MIT-DMCG	ITS on PDP-10
Harvard, Utah	DEC 10/50 on PDP-10
UCSB	OS/MVT on 360/75
ISI, BBN, NIC, I4	TENEX on PDP-10

Table 3-11 Sites Supporting NETRJS Use (Braden73)

protocol on the ARPANET, since CCN believed that efficient and reliable transfers of large files would be very important to its users. The effort was largely frustrated by the generally minimal efforts invested by most other Hosts. For example, no other site has implemented the restart capability or the high-efficiency block modes.

It is obviously impossible to identify all the uses made of the CCN 360 via the ARPANET. Some of the major computing efforts which were performed are:

1. The Rand Corporation's meterological group, CCN's first large user, operated an atmospheric circulation model, which included programs requiring 850K and 1200K bytes of memory. In four CPU minutes, the 360/91 simulates six hours of real time. It was estimated that the same computation would have required 40 CPU minutes to complete on Rand's 360/65. Since their usual practice is to simulate 90 days of real time for their runs, it can readily be seen why they needed access to the bigger, faster machine.

2. At the University of Illinois, both the Laboratory for Atmospheric Research and the Center for Advanced Computation make use of CCN's 360/91 through a variety of terminals and output devices at their computer center. These include a Gould electrostatic printer, an IMLAC programmable terminal, a Computek storage tube, and a penntype data plotter. As part of their work, they have produced displays which present a three-dimensional plot of pressure distribution within a cloud.

3. The ILLIAC IV users group at NASA Ames Research Laboratory has used CCN's 360/91 to test programs to be run on the ILLIAC IV. When the programs are successfully run at CCN,

they are then run on the ILLIAC IV, and the results are compared to validate the results produced by the ILLIAC IV.

4. Professor Hearn of the University of Utah used CCN over the ARPANET to do the formal algebraic manipulation of problems occurring in high-energy physics. Professor Hearn ran his REDUCE processor to evaluate and expand algebraic expressions, using the 360/91 because of its speed and memory size.

5. Teledyne-Geotech in Alexandria, Virginia, has used CCN for signal processing (data reduction) in support of seismic research. They were using their IBM 360/44 to perform fast Fourier transform analysis on their data. The data involved exceeded the capacity of the 360/44.

Overall, in spite of some feeling of alienation as a batch, "unit record", service center system in a time sharing "paper tape", research system world, CCN's experience as an ARPANET server has been favorable. From a financial point of view, ARPANET usage of CCN computing services ranged from 10% to nearly of 20% of CCN's income between 1970 and 1975, providing an important aid to balancing CCN's budget. Obviously, CCN was still primarily a local service center for student and faculty needs. However, from the Government viewpoint, CCN was an excellent resource. The 360/91 hardware was a half a generation beyond the 360 technology, and remained price-competitive with other cost recovery centers until recently. At ARPA's suggestion CCN modified its accounting formulas to provide equitable charging for large-memory jobs; the Rand meteorological project proposed to run jobs requiring 2400K bytes for hours at a time. This change was useful to both ARPA and UCLA.

There have been many other benefits to UCLA from CCN's involvement with the ARPANET. For example, programs installed at CCN by or for ARPANET users were also available to all other CCN users, including EISPACK, SPEAKEASY, REDUCE, NASTRAN, and PL/M.

The ARPANET services that CCN performed were often related to, or at least similar to, existing faculty projects. For example, the Rand Climate Dynamics project was dependent upon the models and codes created by professors Mintz and Arakawa of UCLA. In another notable example of remote collaboration, Professors Donchin at the University of Illinois and Vidal of the UCLA Computer Science Department used the ARPANET through CCN to collaborate on collecting and reducing experimental data.

The ARPANET provided a mechanism for several CCN staff members to interact with a community of bright system designers and builders, and provided the stimulus of new ideas and varied backgrounds. Furthermore, CCN management has observed that the ARPANET software design for the 360/91 was an interesting and challenging project, undoubtedly a factor in keeping several of the best systems programmers on the staff. Many programers on the systems and user services staffs of CCN gained valuable exposure to the non-IBM world through ARPANET involvement, and gave the staff new insights into how much better software can be. This contact has had many subtle benefits and raised the level of system design and user interfacing at CCN.

3.1.2 TENEX Systems

The TENEX system is a virtual memory system built around the DEC PDP-10. The TENEX operating system, together with the associative memory page-mapping hardware which is used to support it, were developed at Bolt Beranek and Newman. TENEX implements a virtual processor with a 256K word (36-bit words) memory for each user process. In addition, it provides a multiprocessor job structure with software interrupt capabilities, an interactive command language, and advanced file handling capabilities. Files are generally interpreted as character stream devices; thus real terminals, Network Virtual Terminals, and other network connections can easily be viewed by user processes as files.

The TENEX system has proven quite popular within the ARPA research community, perhaps due to its early (about 1970) implementation of virtual memory at quite low cost, and its highly human-engineered user interface which grew out of the tradition of the XDS 940 Project Genie Software. This popularity is reflected in the fact that 17 TENEX systems are listed as servers in the 1976 ARPANET Directory. Further, nine of these systems are located in only two centers, at the University of Southern California's Information Sciences Institute (ISI) and at BBN's Research Computer Center (RCC). In fact, ISI and the RCC are the two major suppliers of TENEX service to the ARPANET.

Although TENEX is a general-purpose time-sharing system, with the ability to run almost all software written for the PDP-10, it is probably true that the vast majority of ARPANET use of TENEX is related to document production (editing, formatting), computer mail (reading, sending, filing), or artificial intelligence work using Interlisp (Interlisp is a dialect of the list-processing language LISP, the development of which has been distributed among people at several ARPANET sites). Generally speaking, TENEX systems have been specialized to be either more suitable for text processing and mail or to be more suitable for running LISP.

Both the ISI and RCC centers are heavily involved in system software development, computer science research, and networking experiments; thus each site initially appeared to place less emphasis on stability, high availability, and reliability than more service-oriented sites (e.g. CCN). Both sites approached the problem of integrating their research with provision of stable service by the acquisition of additional hardware to construct a "service" system and a "development" system, with the development system also available for backup use. Once this step was taken, these two sites were able to offer significantly better performance than other, single system, TENEX sites and this fact probably contributed significantly to the gravitation of new users to these sites, supporting additional systems, achieving increased economy of scale in operation, and thus adding to their competitive margin.

It is interesting that neither ISI nor BBN provides much user support in the form of consultants or hardcopy documentation. The TENEX sites tend to supply small amounts of computational service to very large numbers of users, in contrast to the service pattern typical of a batch center. For this reason, user support of the type offered by, for example, CCN appears to be prohibitively expensive. For this reason, the TENEX service sites have tended much more to online documentation and "consulting" via mailbox services.

In spite of the fact that the RCC and ISI are in some ways competitors for TENEX business, it is worth noting that they are quite different types of organizations and thus true competition does not really exist. The RCC is part of a commercial enterprise which has furnished its own computers and is attempting to make a profit in selling services. ISI is part of a non-profit organization which is operating government-furnished computers and is being reimbursed by the government for operational expenses. Therefore, direct government TENEX use tends to take place at ISI; other ARPANET use, for example by private contractors working on government projects, is somewhat more competitive.

3.1.3 Multics

Multics is a general-purpose computer collaborative effort by the MIT Laboratory (formerly Project MAC), Honeywell Informa the General Electric computer division) Laboratories. Multics was designed wit being a prototype "computer utility", concepts and philosophy of earlier tim several directions. The ability to share d of the data sharing, is handled in a Multics structure.

Multics at MIT is a virtual memory sys H6180 system with two CPUs, 384K word microsecond core, an 2 million words microsecond core). The hardware archit segmentation and paging. A hierarchy protection "rings" are used to limit relatively flexible way and provides a basi build a secure (in the military sense Machine resources and accounting are u software that gives users who manage control over consumption patterns within th TENEX, there has been extensive human-engi interface (although the interfaces are quit

During the stage of development of Host protocols, the Multics representatives, like those from CCN, tended to feel that they were seriously "outnumbered" by the representatives of the PDP-10 sites, and the TIP designers. The major points of friction included:

1) Multics was a line-at-a-time system and paid a fairly stiff penalty in system overhead for fielding terminal input on a character-at-a-time basis. Nevertheless, the TELNET protocol design meetings were strongly oriented to character-at-a-time operation.

2) Multics distinguishes between upper case and lower case characters; it is important to use the correct case in, for example, specifying a user name in a message. Most other systems, notably the PDP-10 systems, mapped lower case into upper case during the specification by the user of file or user names, as an aid to users with Teletype Model 33 terminals. Considerable pressure was put on Multics (not TENEX) by the ARPA office to "make it easy for Multics to accept TENEX-generated network mail."

3) Protocol for file transfers, especially for the transfer of mail, took a rather casual approach to access control (i.e.,

authentication and system security). Because of the Multics emphasis on being a prototype computer utility, and the type of fairly stringent access controls which that implies, it was difficult for Multics to accept this casual approach.

Multics development and operation is marked by an organizational division which is rather unique in the ARPANET community, and which has led to less use of Multics via the network than would have been predicted based on the interest in, and capabilities of, the system. The development of the ARPANET hardware and software was carried out by the Laboratory for Computer Science (LCS), with a staff of graduate students and researchers. The Multics server, however, is operated by MIT Information Processing Services (IPS), a service organization which has had difficulty in generating internal enthusiasm for use of this hardware and software in a production environment. At least partly for this reason, Multics has been less aggressively promoted than other systems. Nevertheless, it is estimated that 15 to 20 percent of Multics usage is via the ARPANET.

It is worth noting that IPS has vigorously pursued the goal of insuring that local users do not indirectly subsidize network users. This has led to a policy of handling as little of the protocol as possible at system level (i.e., interrupt level); this policy, in turn, has in some cases interfered with the capabilities which were intended by the protocols. As an example, performing Host-Host protocol flow control in the user's space, yet reading and writing the IMP interface in system space introduces an extra level of data transfer, and to some extent masks the effectiveness of the flow control mechanisms. Other network servers have been somewhat more relaxed about distributing the overhead associated with the network among all users, just as the cost of other I/O devices are distributed.

3.1.4 ILLIAC IV

The ILLIAC IV is a very large scale parallel processor, with a control unit and 64 processing elements. Each processing element has a local memory of 2K words (64-bit words); the control unit can address all 128K words. A considerable amount of secondary disk memory is provided in close association with the machine, and a very large UNICON laser storage device is included in the ILLIAC complex, although it is not yet operational. Actually the 64 processing elements represent one "quadrant" of the original design, but the remaining three quadrants were never built.

Access to the ILLIAC is via a number of PDP-10 and PDP-11 computers which serve the role of peripheral I/O processors and front-ends. These, in turn, are connected to a number of communications systems, including the ARPANET. In fact, although the beginning of the development of the ILLIAC predated the ARPANET by several years, the concept of user access to ILLIAC via the ARPANET has been central to ILLIAC site planning since 1970; at times it was anticipated that all access to the ILLIAC would be via the ARPANET, and several different IMPs were installed to make this possible.

An IMP was first installed at the University of Illinois' Center for Advanced Computation to provide for ILLIAC access after the machine had been delivered to that location. When the Illinois campus began to seem unsuitable as an eventual site for the ILLIAC, the search for a new site was conducted with the knowledge that users of the system would be able to access it via the ARPANET, and thus the geographic locations of the system's users did not need to have a first order effect on site selection. In early 1971 an IMP was installed at the Burroughs' Paoli, Pennsylvania facility where the ILLIAC was being constructed; this IMP provided access to the B6500 which was originally intended as the ILLIAC I/O interface. Both an IMP and a TIP were eventually installed at the Institute for Advanced Computation at the NASA Ames Research Center when this site was selected for ILLIAC installation.

The ILLIAC connection, and the anticipation that all user access to both the processor and the associated "trillion bit" laser store would be via the ARPANET, had some effects on the ARPANET which are not completely obvious. It was expected that huge volumes of data, requiring relatively high transfer rates, would be funnelled into the vicinity of the ILLIAC; the direct result of this expectation was the initiation of the Pluribus IMP project, with a design goal of handling connections operating at over a million bits per second. Two of the ARPANET inter-IMP circuits terminating at the ILLIAC's IMP were upgraded from 50 to 230 Kbs, and one of them (from the Moffett Field IMP site) was ordered to be further upgraded to over one megabit speed. Eventually, however, it was realized that much of the bulk transfer traffic to ILLIAC would be classified, and modes of access not involving the ARPANET were chosen as a result.

From a system software standpoint, the ILLIAC is rather primitive; a user takes control of the entire system, runs his own software, and then relinquishes the system in a single-programmed, batch-type mode of operation. A variety of special languages including CFD, Glypnir, and Tranquil have been written to assist programmers in taking advantage of the computational parallelism provided by the system.

Use of ILLIAC and the associated facilities has been administratively limited to ARPA-authorized and NASA-authorized user groups. The customer support problems faced by the ILLIAC staff are thus limited to interactions with a relatively small and knowledgeable community.

It has been reported [Falk76] that NASA Ames researchers use about 20 percent of the ILLIAC's operating time (which totals 60 hours a week) solving two-dimensional aerodynamic flow partial differential equations, and that they would like to process three-dimensional flows but have run into processor speed and memory size limitations. Other uses of the ILLIAC include global climate simulations from Rand and other NASA weather simulations, signal processing, seismic research, and linear programming; in general these uses are carried out via the ARPANET.

3.1.5 The Datacomputer

Like the ILLIAC and the 360/91, the Datacomputer serves as a specialized resource to a widespread community. However, rather than specialization as a computing engine, the Datacomputer is specialized to be a large-scale data management and storage utility.

Much more than the servers discussed previously, the Datacomputer was designed specifically for use in the ARPANET environment. The Datacomputer designers at the Computer Corporation of America (CCA) acted on the belief [Marill75] that within a resource sharing network there is a natural tendency toward specialization of network nodes, such that the factoring of problems into their constituents, the assignment of these constituents to the appropriate machines, and the recombination of results will tend to become an automatic process. In the limit, specialized network nodes become what may be termed "utilities", that is, machines which perform a restricted range of functions solely for the benefit of the other machines. The Datacomputer is a network utility in this sense. It is entirely specialized for the performance of data management and storage functions. The Datacomputer designers further speculate that the trend toward specialized network utilities will continue, and that the traditional stand-alone general-purpose machine will eventually disappear from the scene. The computer world envisioned in such a speculation might consist of a network containing a few very large Datacomputer-like systems, a few very large computational utilities ("number crunchers"), and a large number of small human-interaction units (such as intelligent terminals), having limited computational power and local storage.)

Physically the Datacomputer consists of a processor, three levels of data store, and a connection to the ARPANET. The processor is a PDP-10 TENEX system, with primary core memory storage of 336K words (36-bits). Secondary storage consists of six spindles of IBM-2314-equivalent disk (24 million words total) and four spindles of IBM-3300-equivalent disk (60 million words total) which are used as a staging area for tertiary storage. The tertiary storage device is an Ampex Tera-Bit Memory (TBM) system, which consists of control units built from DEC PDP-11 processors and video tape drives with a recording density of a megabit per square inch. In the CCA configuration there are four tape drives, each of which contains 50 billion bits of storage, and transfers data at 6 megabits per second. A high speed seek from one end of a tape to the other takes approximately 45 seconds. The TBM can be expanded to a total of 64 tape drives for 3.2 trillion bits of on-line storage.

The primary anticipated advantage of specialized network utilities is economy of scale, and the Datacomputer project fits this model well. The base cost of a TBM unit is very large; the cost of the TENEX front-end is not small. Yet the cost per bit of the TBM is about $1 per megabit, about a factor of 20 down from the cost of disk storage. Thus, while few individual installations could afford the price of a TBM, the pooling of many users' requirements allows significant savings for all. In addition, of course, the cost of software development, maintenance and improvement can be spread over a large number of users.

The existence of the ARPANET motivated the development of the Datacomputer; the characteristics of the network shaped that development. The heterogeneous computer population of the ARPANET lead to extensive provision in the Datacomputer for a variety of byte sizes, character sets, numeric representations, and data stream structures. In most cases, automatic synthesis of, or conversion between, the alternatives is available. For example, one user could input a stream composed of variable length EBCDIC character strings terminated by a delimiter character. Another could extract the same data as a stream of variable length ASCII strings each preceeded by a length count, The Datacomputer communicates with programs that run on remote machines. The fact of remoteness precludes the use of simple subroutine calls or similar means of communication conventionally used within a single machine. The communication, furthermore, is not with people at terminals, who can be expected to make intelligent responses when failures or unusual circumstances occur, but with programs. Hence, all synchronization messages, error messages, language statements, and file descriptions must be creatable and readable by programs; likewise, a facility for checkpointing by user programs is required.

Datalanguage is the language in which all requests to the Datacomputer are stated. Datalanguage includes facilities for data description, for database creation and maintenance, for selective retrieval of data, and for access to a variety of auxiliary facilities and services. A basic characteristic of datalanguage is that all data is described. Descriptions are stored in the Datacomputer directory and are available to the user program in machine-readable format. A description contains the information needed to interpret the data, that is, information on data representations and structure.

The bandwidth available over the ARPANET, generally less than 40,000 bits per second, is two orders of magnitude less than typical local disk bandwidth. This means that different strategies and facilities are called for in using the Datacomputer over a network than for a user accessing data stored on a local disk. In particular, it is necessary for the user program to be able to send self-contained requests and to synchronize with and keep track of the remote task doing the user's bidding at the Datacomputer.

As an example of a self-contained request, one might want to update an employee file to indicate an across-the-board salary increase of 10%. With most local disk systems one can read each record and write it back updated. This would be relatively slow in a network environment. However datalanguage is powerful enough to specify this sort of operation to be performed entirely within the Datacomputer, insuring no network delays for data transmission.

The user has extensive control over the internal format of the data for efficiency reasons and flexible access controls are provided. At each directory level down to the individual file, data access and control privileges can be granted or denied to classes of users. The classes can be specified in terms of the

Host number and the Host-Host protocol socket number from which the user communicates with the Datacomputer. The more usual password and login name restricted user classes are also available, as well as combinations of these with Host and/or socket number.

Quite aside from the control of access by password and address, the Datacomputer environment provides a stronger type of access control to data than is usually possible in a general-purpose machine. By definition, a general-purpose environment allows the programs within it enormous latitude in the functions they can perform, and it appears that programs can often be written to circumvent existing access regulation procedures by taking advantage of errors that arise in unexpected circumstances. Such hostile programs are sometimes able, without authority, to access data, delete data, or crash the system and prevent other users from legitimately accessing data.

In the environment of the Datacomputer, the situation is quite different, since the system is logically a closed, dedicated, special-purpose box, which responds only to a limited set of commands and does not provide a general-purpose computing facility. A hostile user program cannot be run on the box because the box does not run user programs. The approach can inherently provide stronger guarantees that programs without proper access authority will not be able to access or damage data contained in the Datacomputer.

The TBM device was not available for use until late in 1976, but major use of Datalanguage and the secondary storage facilities of the Datacomputer began several years earlier. Some of the most significant Datacomputer uses include:

1. Seismic Data: The seismic network project is covered in more detail in Section 3.3. A significant aspect of the project is the transmission of real-time raw seismic data to the Datacomputer at rates of 7-12 kilobaud, around the clock. By the end of 1976 nearly 70 billion bits of seismic data (raw readings, event summaries, instrument status reports, and various historical files) had been stored. If stored on conventional disk storage, this volume of data would have required more than 85 spindles of 100 megabyte drives. As the seismic database grew, researchers began retrievals against it, initially to develop and test procedures for use in ongoing seismic studies.
2. ARPANET Subnetwork Statistics: The ARPANET Network Control Center has been an established user of the Datacomputer, storing statistics on performance and usage of the IMPs and

circuits which implement subnetwork. Usage had grown to 600 megabits by the end of 1976, with steady retrieval activity against the data.

3. Host Surveys: Another well-established use of the Datacomputer is the SURVEY application, carried on in conjunction with MIT's Laboratory for Computer Science. Current survey data on the status of Hosts on the ARPANET continued to be stored at the rate of about 10,000 probes per day.

   The SURVEY database has been used not only by those interested in the data per se, but also by researchers in the Very Large Data Base project at MIT. They used the Datacomputer's processing of requests against 2 quarters' worth of SURVEY data to test their work in estimating the cost of query processing in very large databases.

4. ERDA: Several of ERDA's national laboratories have begun investigations of the Datacomputer; most active were groups at the Argonne and Lawrence Berkeley Laboratories. The major project is installation of a climatological database by personnel at Argonne. This database contains 16 files, each with a year's worth of hourly readings for some U.S. city; the data are used by a number of sets of programs which model energy usage in buildings and communities.

3.2 User Systems

Just as there are many kinds of server systems connected to the network, so there are many kinds of systems which provide access to the ARPANET for terminal users. However, the vast majority of such systems are either Terminal IMPs or are DEC PDP-11's running the ANTS or ELF terminal support systems. Each of these is described briefly below.

3.2.1 The Terminal IMP

Initial terminal access to remote services on the ARPANET was via connection to a local Host. However, it quickly became obvious that use of a general-purpose time-sharing system as a simple access port into the network was quite expensive. In addition, there was considerable interest in accessing the network from locations where there was no local Host in existence. These two observations led to the development of the Terminal IMP (or TIP) [Ornstein72] to provide direct network access for certain classes of terminals.

Logically a TIP consists of two independent machines, a normal IMP and a terminal-handling mini-Host. Physically, however, these two logical machines are combined in a single

Honeywell 316 computer identical to the 316-based IMP, but with the addition of 16K words (16-bit words) for terminal-handling program and buffers and a sophisticated Multi-line Controller (MLC) which provides the common logic for 63 bit-serial, character asynchronous terminals.

The TIP was designed to provide an inexpensive, reliable method for direct network access, and as such attempted to meet rather limited goals. First, it was felt that the TIP should serve human-operated interactive terminals, implying little need for flow control or error checking mechanisms between the terminal and the TIP. Second, it was assumed that elaborate terminals, such as graphics displays, would either contain a Host or access the network through a Host. Third, any computational requirement that might arise should not be handled in the TIP, but rather either by "intelligent" terminals or by the remote Hosts. The resulting TIP was therefore intended to provide access for users with "simple" terminals.

Compared to normal computer installations, TIP installations are a bit peculiar. The TIPs are delivered without any systems documentation for the software system; in fact, since the TIP is part of an IMP, site personnel are forbidden to touch the system lest they jeopardize the integrity of the subnetwork. As with IMPs, problems are handled by the operators at the Network Control Center. Furthermore, since there is nobody locally who can be asked to modify the TIP system according to local desires, all requests for modification must be submitted to the TIP development group at Bolt Beranek and Newman.

One type of continual pressure for modification is the steady introduction of new terminals to the market, and the desire on the part of the TIP user community to attach these terminals. In general, such terminals are "Teletype compatible", but each tends to have slightly different characteristics, for example in timing of print head movements to a new line, beginning of line, etc. Although differences like these may seem minor, each needs its own little bit of code to work properly or even to work at all. The general approach in the TIP is to provide each type of device with a piece of straight-line program to handle it. This maximizes TIP throughput by requiring the TIP to perform only the necessary per-character processing for any given device. The TIP actually handles Teletype Model 33's, IBM 2741s, Odec line printers, Memorex line printers, Execuports, and any terminals compatible with them.

At one point after the TIP had been implemented, it was thought desirable to have a magnetic tape drive option for the TIP. Since this was clearly a device which could not be handled by the MLC, a standard Honeywell drive was used, and an ad hoc protocol was implemented to transfer tapes between TIPs so equipped. However, it was impossible to smoothly integrate the tape equipment into the TIP structure, because its characteristics differed so widely from those assumed during the original design.

Other services for which there has been demand are things such as elaborate device status information, on-line server schedules, "news" and "complaint" facilities, and even access control and accounting facilities. It is clearly beyond the capacity of a TIP to handle these sorts of features by itself. However, since these are the capabilities normally associated with a service Host, it was logical to seek to provide such facilities at a larger Host and construct the mechanisms which could allow a TIP user to gain access to them in a relatively transparent manner. The actual implementation of this concept makes use of the RSEXEC (see Section 3.5) on several TENEX systems (the reliability of the service is thus increased through redundancy), and an ad hoc "broadcast Initial Connection

Protocol" which asks for a connection from each instance of the service and accepts the connection from the most responsive [Cosell75]. The ability to choose the proper location to implement a new function, as exemplified by such facilities, is clearly one of the advantages of a network environment.

It now appears that the original assumption that computing power should be in the terminal and not in the TIP is an oversimplification. Although it is reasonable to claim that a terminal access computer should not perform rotation calculations for graphics terminals, a terminal with a computer (such as an Imlac) would prefer to follow a more sophisticated communication protocol than is sufficient for Teletypes. For instance, the terminal may not always be prepared to accept characters when the TIP sends them and, conversely, the TIP is frequently not able to accept characters at the rate the terminal can send them. The MLC, designed for interactive terminals, does not lend itself to sophisticated flow control. One cannot merely say that the computational power should be in the terminal and not in the TIP; the very fact that the terminal has extra computational power requires the TIP to have some degree of matching power. However, the addition of relatively little computational power to the TIP can permit a large amount of additional computational power in the terminal.

3.2.2 ANTS

ANTS is an acronym for "ARPA Network Terminal System", a system which was developed at the Center for Advanced Computation at the University of Illinois. The ANTS design is much more ambitious than that of the TIP; it was developed to provide network access support not only to simple character terminals but also to devices such as synchronous batch terminals, card readers, line printers, tape drives, and sophisticated graphics display terminals [Bouknight73]. The separation of terminal support functions from the network node permits additional capability in the system in the form of a disk, a wider variety of peripherals, more memory, etc., and thereby provides a limited amount of on-site processing power for the user to do housekeeping functions such as network accounting, on-site card deck-to-printer listings, data storage on magnetic tape and disk, etc. It also provides him with a higher level interface into the various protocol streams than is provided by the TIP.

In addition to the terminal support and local housekeeping functions, the ANTS design was intended to offer a significant alternative to direct interfacing of large computer systems to the ARPANET by acting as an "intelligent interface". An "intelligent interface" is one which can be programmed to interface between dissimilar systems and map the input-output characteristics of one to the other. In such an instance, ANTS might be used to map terminal data streams into a form expected by the data-communications front-end of a large computer system such as the CDC 6600.

Internally, all devices which are available for use are grouped in classes for control purposes. In a typical ANTS configuration, there are several more or less standard classes:

1. Network connections
2. Files
3. Terminals
4. Non-file structured peripherals
5. "Intelligent interface" ports

Each class (or in some cases sub-class) is controlled by a device manager. The device managers in ANTS are "virtual machines". They receive and execute "instructions" for passing data to and from their associated devices and the rest of the system. Other "instructions" provide for assignment of devices to system or user tasks.

Internally, all data is moved through the system in strings of 8-bit bytes called messages. Messages are transmitted along simplex "data paths". Each data path has a capacity for a specific number of messages and a specified number of bytes which may be in transit over the data path at any given time. Thus a data path enforces implicit flow control. A second important capability of the data path is performance of data transformation operations. Such functions as code conversion, record compression and expansion, or aggregation of many small messages into large composites can be dynamically inserted in the data path.

The first ANTS system went into prototype operation at the Center for Advanced Computation of the University of Illinois in late 1971. By late 1973 it included the following hardware:

PDP-11/20 CPU
28K words core
256K word head-per-track disk
ROM bootstrap
Real-time-clock
Interval timer
Paper Tape Reader/punch
4 DECtape drives
2 9-track magnetic tape drives
36" drum plotter
GOULD 4800 Printer/Plotter
Card Reader
2 Computek Storage Scopes
IMLAC Graphics Display
8 Dial-in interfaces (110-300 Baud)
12 2400 Baud CRT terminals
2 TTY37 Teletypes

The ANTS project obviously was undertaken with a rather ambitious set of goals, aiming to support local editing, the handling of complex terminals, and the "intelligent interfacing" of complex Host computers, all with a small processor. Additionally, in order to make it possible for user organizations to modify and maintain their own systems it was decided to implement ANTS in an ALGOL-like language designed at the University of Illinois specifically for the purpose of implementing ANTS. Thus, the programmers and system designers were faced not only with the task of trying to understand and implement a large number of extremely complex interfaces while maintaining high program bandwidth, but also the problem of using a new language and its compiler. For reasons which probably included these, the ANTS project ran into serious difficulties, both with schedules and performance. Because of these difficulties, only a small number of the potential ANTS user sites actually installed the system on an operational basis.

3.2.3 ELF

ELF is a multiprogrammed operating system for the DEC PDP-11 (ELF is German for "eleven", and is recognizably similar to IMP) which currently has much the same goals as ANTS [Retz75]

[Retz76]. However, it was originally designed to provide a multi-user interface to computing resources via the ARPANET for solving problems requiring interactive signal processing capabilities.

ELF development was started in early 1973 at the Speech Communications Research Laboratory in Santa Barbara, California. In early 1974 that system was tested at several network sites and a decision was made to expand the system capabilities. An initial version of the expanded system was operational at the beginning of 1975. It is somewhat ironic that the decision to expand the system to provide more of the (intended) ANTS capabilities was made at least partly because the ELF system became operational so much more quickly than ANTS, but a major reason for the large difference in the rate of development was the huge difference in initial goals.

The ELF system has a hierarchical structure. At the center of the system exists a set of modules, collectively referred to as the kernel, which perform tasks of resource management in a multiprogramming environment. An ELF process may be thought of as an autonomous stream of instructions (i.e., a program), having an associated program counter, general-purpose registers, stack storage, and a linked list structure called an event queue. Processes may be in one of two states, ready or blocked. The kernel provides a set of primitive calls for outer-level procedures, performing such tasks as the creation of processes, process synchronization, storage allocation, and sharing of an interval timer.

System procedures outside of the kernel perform tasks such as the interpretation of user commands from terminals, maintenance of the system file structure, and control of communication with the ARPANET. The portion of the system which provides terminal support in ELF is referred to as the EXEC (or Executive), and is patterned after the user interface provided by the TENEX operating system. The ELF Network Control Program (NCP) controls the communication between processes in ELF and those processes residing in remote operating systems on the network.

The ELF system is being used in a variety of applications which require a set of operating system components for network communication. The ELF Kernel provides a base for a variety of software configurations which are tailored according to system requirements. The Kernel, EXEC, and NCP modules support a number of users who access the network by running the TELNET sub-system of the EXEC.

Additional processes may be added to the system for peripheral support. For example, a simple process which uses the ELF NCP to await a remote request for connection may be included in the system. The process accepts a connection, receives a stream of data, and outputs it to a local line printer. A status indication from the NCP signals that a remote process has closed the connection, and the peripheral control process then returns to the "listening" state.

The support of real-time data acquisition functions for speech research has been one of the goals in the design of the ELF Kernel. An EXEC sub-system performs functions of real-time data sampling and file I/O for digitization of speech waveforms. The system may thus be used as a data acquisition station while simultaneously providing a terminal support function for access to remote systems.

The ELF design has placed much more emphasis on development and maintenance via the ARPANET than ANTS. In addition, partly because of the limited initial ELF objectives, ELF is written in a macro assembly language rather than a high-level language.

This admittedly makes modification and expansion more difficult, but did probably make it easier to meet the real-time signal processing requirements. Cross-compilers for the BCPL and L1011 languages which generate PDP-11 code are available on network TENEX systems for production of ELF user-level code.

Several debuggers which run on a stand-alone PDP-11 have been modified to run in an ELF user address space. An additional debugging mechanism involves the interpretation of debug commands received from the network by a system debugging process. This approach minimizes space requirement in a processor being debugged while taking advantage of facilities available on large systems to provide a comfortable user language and a symbolic representation of addresses and instructions. A system debug process which resides in ELF utilizes the breakpoint signal facility provided by the Kernel and is responsible for debugging of a number of other processes. ELF system software is distributed as a set of source modules which are accessible in a directory at one of the network TENEX service sites. Users may access the source files by means of the network file transfer protocol.

The ELF system has been quite successful in the ARPANET, both as a terminal support system and as a front-end. By late in 1976 there were between 20 and 30 ELF systems in operation in the network.

3.3 Specialized Communities of Interest

This section discusses three of the hundreds of specialized user communities which make up the patterns of network use. These are the climate dynamics research community, the seismic research community, and the "secure" user community; they are of special interest for quite different reasons. The climate dynamics research, carried out primarily under the direction of the Rand Corporation, is interesting because it used several large computers and was one of the first major activities making serious use of the ARPANET. The seismic research activities are of interest because of their heavy realtime component and their heavy demands for network bandwidth. Finally, the "secure" use of the ARPANET is of interest for the techniques used to make it possible.

3.3.1 Climate Dynamics

The Climate Dynamics Program at the Rand Corporation studied the effects on climate of large-scale changes in the environment by running global simulation models of the oceans and atmosphere. Experiments might study the effects of changing the mean temperature of an ocean, melting a polar ice cap, or placing a large lake in the middle of the Sahara Desert.

A typical experiment scenario was to first run a simulation of 90 days of the unmodified environment as a control standard, followed by a 90 day simulation beginning from the same initial data with the exception of the desired environmental modification. Two of the programs which are part of the modeling system require large regions of core, about a million 8-bit bytes, and have huge computational requirements. The model required about 40 minutes on the Rand 360/65 to simulate six hours of real time; the same simulation requires only four minutes on the UCLA CCN 360/91. Thus, the first requirement of the program was to use the ARPANET to access the 360/91 and later, when it became available, the ILLIAC IV.

However, vital as access to the large processors was for running the simulations, the greater part of the programming effort was involved in the post-analysis and display of the large data files that were generated as results. One 30-day simulation provides a full IBM 2314 disk pack, 28 million bytes of data. To be able to analyze this data meaningfully, a meteorologist must be able to scan the resulting weather picture and focus in on areas of interest. He needs to look at such diverse items as the change in temperature or pressure at a given point over time; or at the effect of an air pocket over the Pacific on the pressure at the east coast of the United States; or on the humidity in Seattle. The post-analyses were primarily performed by large batch jobs, but also involved some interactive jobs. The displays were large printouts (100 or so pages) and static graphics (magnetic tapes were generated at UCLA and sent to outside vendors for the generation of microfilm and printer capabilities not available on the ARPANET) as well as interactive, over-the-net graphics from both ILLIAC and UCLA to a TEKTRONIX terminal at Rand. The display techniques used were not dynamically interactive, but did allow the researcher to choose the data and describe the display to be presented. The remote running of the graphics programs was a result of the connection of the terminal to the network via a 370/158 at Rand which was not set up operationally to handle interactive user applications. The Rand computer was used primarily for storage of source programs, text editing, and listing output. It was felt the editing functions were best kept local to avoid interference with these functions when the network or remote servers were down.

Batch jobs were submitted to UCLA via the NETRJS protocol; some result retrieval from UCLA and all interactions with the ILLIAC system used the File Transfer Protocol. However, since no site except UCLA (including Rand) implemented the compressed mode of data transmission or the restart markers specified in FTP, there were annoying problems because of the amount of time involved and unreliability of the Hosts [Mobley77]. When a transmission "broke in the middle" the only recovery was to restart at the beginning. Another difficulty for the weather researchers was the non-standardization of common commands (LOGON vs SIGNON or LOGOFF vs LOGOUT) from one system to another, and interpretation of responses, especially when there were problems with the remote Host or its network interface. In addition, the network graphics protocols were insufficient for this group, which defined a standard low-level software interface for each graphics device, so that existing higher level routines could remain the same.

Nevertheless, the researchers involved in the climate dynamics project report that "we made extensive use of the ARPANET and the advantages were tremendous. It was an RJE facility. It was a connection to remote interactive systems. It was our means of moving data between remote computers. It reduced turn-around time and increased the efficiency of our use of programmers' time a great deal [Mobley77]".

3.3.2 The Seismic Community

The seismic data activity involves the collection, storage and processing of seismic waveform information (seismograms) as measured by seismometers installed throughout the world. The data will assist seismologists in exploring techniques for detecting seismic events, pinpointing their location, and recognizing the causes of these events. A major application of the work is the detection of underground nuclear tests in preparation for future Strategic Arms Limitation Treaties. By establishing an on-line, real-time database of seismic information from a world-wide network of monitoring sites, a great deal of data can be made easily available to computers in the network for seismic analysis and other purposes [Dorin77].

The computers at various sites involved in the gathering and subsequent analysis of seismic data are known as the Vela Seismological Network or Velanet. Many of the computers are on the ARPANET, which therefore was chosen as the most appropriate communications medium available for much of the system. The Velanet consists of two sites sending seismic waveform information in real-time, the Large Aperture Seismic Array (LASA) in Montana, and the Norwegian Seismic Array (NORSAR). LASA data is transmitted via leased telephone lines to an intermediate processor (the Communications and Control Processor, or CCP) at the Seismic Data Analysis Center (SDAC) in Alexandria, Virginia. NORSAR data arrives at SDAC via a satellite communications link of ARPANET.

Data which is not transmitted in realtime arrives at SDAC on magnetic tapes from the Iranian Long Period Array (ILPA), as well as from other instrument clusters throughout the world. Non-array seismic data is sent by magnetic tape from various locations around the world to the Albuquerque Seismological Laboratory (ASL) of the U.S. Geological Survey. Both the real-time and non-real-time data arriving at SDAC, as well as the data concentrated at ASL, are forwarded through the ARPANET to the Datacomputer (see Section 3.1) at CCA. The seismic data traffic to the Datacomputer is 12 kbs around the clock or approximately 30 billion bits per month. There are plans for additional sites which may boost the traffic volume to 35 kbs.

Processors throughout the Velanet can retrieve the seismic data. Processors at SDAC, Lincoln Labs Applied Seismology Group (LL-ASG) and possibly elsewhere will be used by seismologists for this purpose.

There are two basic categories of files stored on the Datacomputer for the seismic project. First, the raw data and status files provide complete seismic readings at various instruments and arrays around the world, as well as associated information on the status of these instruments so that the raw readings can be properly interpreted. Second, there are the derived event summary and associated seismic waveform files. These consist of a processed distillation of the first category into likely seismic events, and the time segments of raw data that are associated with these events. The derivation of the second category of files from the first is performed by processors at the Seismic Data Analysis Center.

The real-time data from LASA and NORSAR is, as noted above, routed first to the CCP and from there to the Datacomputer. Processing and buffering constraints in the arrays and their associated computers, the ARPANET, and the CCP provide a very short interval of elasticity (less than 30 seconds) between the time the data is generated and the time it must be accepted at CCA. This small interval of elasticity has not only network implications, which are discussed below, but also implications for the operation at CCA.

The Datacomputer system is implemented within a general purpose time-sharing system with a varying load depending on time of day and other factors. It cannot guarantee the required responsiveness. In addition , both the basic computer system and its peripherals, including the TBM , require regular preventative maintenance, and hence cannot be operated continuously.

To provide the required round-the-clock responsiveness, a small, dedicated, reliable system known as the Seismic Input Processor (SIP) has been implemented. The SIP is a PDP-11/40 with two IBM 3300-like disks. It accepts the data stream from the CCP, buffers it on its disk, reformats the data, and periodically transmits it to the Datacomputer. At the current bandwidth, 26 hours of buffering are provided per disk pack. Thus the SIP completely isolates the real-time data stream from Datacomputer downtime or delay.

The SIP uses standard Host-Host protocol to communicate with the Datacomputer but uses a special protocol for the real-time path to the CCP. This special protocol eliminates the normal Host-Host handshaking and connection setup overheads and results in simpler, more efficient communication. Furthermore, the special protocol eliminates the standard protocol's requirement that no more than one message be in the network in one direction at a time. This modification increases bandwidth and decreases network blocking. The special protocol also maximizes network efficiency by packing logical messages into full size physical messages and uses sequential message ID numbers.

Seismic networking activities have placed considerably more strain on the operational ARPANET than most other uses. For obvious economic reasons there has been only one trans-Atlantic path included in the network, and this path is rated at only 9.6 kbs rather than the usual ARPANET rate of 50 kbs. Even more significant is the IMP-to-IMP acknowledgement strategy which normally allows only eight packets to be outstanding on a circuit. Since the trans-Atlantic circuit is provided by a satellite with a minimum round-trip delay of about onehalf second, this strategy restricts traffic on this circuit to about 16 packets per second, regardless of the packets' sizes. These factors combined to make it difficult or impossible for the Velanet to maintain the necessary flow rate from NORSAR to SDAC. One factor revealed by the analysis of the difficulty was that, because of the 16 packet per second restriction, five TIP terminal users in Norway and England, each typing one character per second and operating in remote echo mode, would saturate the channel (although achieving only a 64 bit per second data rate). Thus this usage pattern, if it actually existed, would be likely to reduce the Velanet bandwidth to unacceptable levels. This analysis lead to a modification of the inter-IMP strategy to allow more outstanding packets over the satellite channel.

The mode of operation at CCA also led to intra-IMP bandwidth problems. The SIP needed to continuously absorb about 12 kbs of traffic from the CCP, and from time to time unload the accumulated data to the Datacomputer. A reasonable data rate for this latter transfer is on the order of 100 kbs. At the same time, of course, other network store-and-forward traffic, and other Datacomputer users, must be accommodated by the CCA node. The TIP originally installed at CCA had insufficient internal bandwidth to meet these requirements while also providing terminal support. The TIP was first replaced with an IMP, but the IMP was required to support a VDH connection which usurped about one-third of the normal IMP buffering, so in spite of marginally sufficient bandwidth the IMP's internal end-to-end protocol buffering requirements resulted in continued unsatisfactory performance. Eventually, the VDH connection was moved to another network node, and finally a higher-capacity Pluribus IMP was installed. (The ARPANET's first Pluribus IMP was installed at SDAC to meet the requirement for more external connections, Hosts and inter-IMP circuits, than the Honeywell IMPs could handle as well as the anticipated bandwidth requirements.)

Yet another effect of the Velanet use of the ARPANET has been the necessity to pay special attention to the large-volume seismic data flows when designing the network topology. For example, because the requirements for source-IMP and destination-IMP buffering to support a given bandwidth increase in proportion to end-to-end delay, and delay increases in proportion to path length (number of hops), it is desirable to have path lengths from NORSAR to SDAC, SDAC to CCA, and ASL to CCA as short as possible. These considerations resulted in the relocation of a previously-existing Washington to Boston circuit to SDAC at the Washington end and to CCA at the Boston end. A Gunter to Mitre circuit was also partially motivated by Velanet needs.

3.3.3 Secure ARPANET use

As the ARPANET moved from a computer communication research tool to an operational service device there began to be interest in using it to carry classified data in a secure manner. The usual technique for transporting such data through an uncontrolled ("black") environment is "end to end encryption"; the data passes through an encryption device (a Key Generator) as it enters the communication system and through a decryption device as it leaves. This technique is not directly suitable for a packet network such as the ARPANET because each IMP needs to process a portion of the data (the destination address). A solution to this problem in military message switching is known as "link encryption"; the message is encrypted as it passes into each communication circuit and decrypted as it enters each switch. This solution, however, is only acceptable if the switches themselves are in a secure ("red") environment, clearly not the case in the ARPANET.

The initial solution chosen is a device known as a Private Line Interface (PLI) which acts as an interface between a secure Host and the non-secure ARPANET. The PLI drives the Host-supplied data through an approved Key Generator (KG) unit. A fixed address, supplied by the operator at PLI initialization time, which is unencrypted ("clear") is then appended to the encrypted data and delivered to the IMP.

In detail, the PLI is a Pluribus computer which appears to the Host to be an IMP, and to the IMP appears as a Host. The PLI is really two computers in series (See Figure 3-1). The secure Host communicates with the red-half PLI via a normal Host-IMP interface. The red-half PLI signals the KG unit to generate and send a key-sequence. The black-half PLI supplies clocking to the KG. It scans the incoming data stream for a key-sequence, compresses the key, and stores it as the first few bytes of an ARPANET message. Then the red-half sends a message segment through the KG unit. The segment is padded with SYN characters if needed to bring it to a fixed size. The black-half adds this data block to the compressed key, assigns a message identifier and a message header (including destination) then transmits it to the IMP. It also notifies the red-half of the message identifier assigned via a uni-directional data and status link (from black to red). When a RFNM or other status indicator is received from the IMP the red-half is notified over this same data link.

When the black-half receives an encrypted message from the IMP, it starts the KG, expands the key-sequence in the message, then transmits the data through the KG unit to the red-half. Since the black-half provides a clocking signal, the red-half must be prepared to accept the decrypted data as fast as it is

Figure 3-1: PLI Configuration

sent. There is no way for the red-half to indicate overrun or other errors to the black-half, although errors are reported to the Host. This requires the Host-Host protocol to effect retransmission should errors of this type occur.

The first pair of operational PLI's were installed in the ARPANET in early 1976 in Sunnyvale and San Diego, California. As a result of the experience gained from initial operation of this pair, approval for some additional control signals (Red Receiver Full, Black to Red Word Empty, Reset) from the red-half to the black-half was sought from the National Security Agency and later received. Approval of such signals is necessary because they contain the potential for passing classified data into the black environment without encryption through accident, malfunction, or malicious design.

The most serious difficulty with the PLI design for the ARPANET environment is that a Host using the PLI is constrained to communicate only with the one other Host whose address is set in the black-half PLI at system start-up. This deficiency was accepted in the initial design in order to shorten the development and approval cycle, but with the approval and installation of the first pair of units, work began on the design of a "multi-address" PLI. The intent of this device is to incorporate a low-bandwidth unencrypted data path from red to black which would allow a few address bits (5) to be associated with each encrypted data block. The black PLI could use these bits as an index into a short table of prespecified destination addresses to select the one to be attached to the data.

The most serious obstacle faced by this plan is the need for NSA to be able to certify the correct operation of the red-half PLI program, so as to insure that classified data is not transmitted through the unencrypted path. The language chosen for implementation of this program is Meta-BCPL; this choice was made to assist NSA in their verification procedure in order to increase the probability of achieving certification.

At the same time, the red-half PLI was expanded to be able to handle up to four Hosts. Thus, assuming final approval of the system by NSA (expected in 1977), a secure Host connected to the ARPANET through the Multi-address PLI will potentially be able to address 32 other PLI's, each with 4 Host, or a total of 128 remote destinations.

3.4 Network Mail

One of the most remarkable outcomes of the existence of the ARPANET has been the explosive growth of what has been called "network mail", "message services", and so on. This growth is the more remarkable because it was essentially unplanned, unanticipated, and mostly unsupported in its early growth.

Early in the course of ARPANET development it became obvious that the researchers involved with, or connected to, the network would want to communicate with each other in various ways. As part of its initial work in facilitating the communication between Host-Host protocol developers, the Network Information Center made use of the SRI On-Line system (NLS) available to this group; NLS [Englebart73] provided a relatively sophisticated text management system which assisted individual researchers to exchange notes, collaborate on document production, create new text structures (essentially files and records) from fragments of previous structures, and so on. In fact, NLS was, although not described in those terms at the time, a serious "automated office" system.

Unfortunately, there were some rather serious drawbacks to ARPANET use of NLS. First, the system was optimized for use from a very elaborate work station, with high-bandwidth graphic output and some rather esoteric input devices. Most network users didn't have the appropriate I/O devices, and their simulation on ordinary keyboard/printers was quite awkward. Second, use of NLS required logging-on to the single computer on which it was implemented, a system which was not always available and which, in any case, had the capacity to support only a modest number of simultaneous users.

At a much more primitive level than the facilities of NLS, most of the interactive time-sharing computers of the late 1960's included software for "linking" terminals together, and for implementing "mailboxes" where short messages (usually to or from the computer operators) could be deposited or retrieved. These tools were constructed to help remote users obtain assistance when problems developed. Such tools were in use at least as early as 1965 on CTSS at MIT'S Project MAC.

The frustration of attempting to use NLS from remote ARPANET sites, combined with the general availability of primitive "mailbox" facilities on many of the network's Hosts, led relatively quickly to the development of simple programs which assisted a terminal user in creating outgoing messages and reading received messages. The origin of these programs is uncertain, but has been attributed to the BBN TENEX system [Henderson77].

Network message service was an immediate success. Message flow grew in volume to become the most visible traffic component on the network. Use of the service has had a substantial impact on the organizations involved, stimulating dramatic shifts of dependence away from the traditional media (postal service, telephone).

Computer supported message service predates the development of the ARPANET by a considerable period. One can find origins for the medium in network message services such as TWX or TELEX. There are many opinions why these services don't seem to have had as strong an influence on the organizations which have access to them as ARPANET mail has had on ARPANET sites, but one key may be the accessibility and convenience of terminals. An organization typically has one TWX or TELEX terminal per building; an ARPANET research site is likely to have one computer terminal per programmer. A second key to the growth may have been a "critical mass" of individuals who wanted and needed to communicate with one another; ARPA project managers who needed to communicate with principal investigators, protocol designers who needed to exchange documents, and a variety of other interest groups. This situation is probably different from, say, the population of users of a commercial time-sharing system or network.

Given the key differences between the ARPANET community and widely distributed sets of individuals with access to electronic transmission systems, as described above, there are several reasons why network mail came to dominate more traditional communications media. These include:

Transmissions speed: A message can be delivered from any input point to any mailbox location in only a few seconds. This is a result of the ARPANET communication system design.

Sender-receiver decoupling: Although transmission speed is almost "real time" in rate, there is no need to establish a real-time channel between the sender and the receiver as there is in telephonic communication (as well as the most common modes of TELEX and facsimile transmission). Requirements for establishment of a real-time channel are often time consuming for a caller and disruptive to the called party. The decoupling is a result of the buffering in the mailbox hosts.

Location-independent reception: Since access to a mailbox is via the network, a mail recipient is not isolated by mobility so long as network access is possible. This is a result of the widespread geographic coverage of the ARPANET and of the availability of truly portable terminal devices beginning in the early 1970's.

Location-independent transmission: Just as it is a major benefit for a mail recipient to avoid isolation during travel, so it is a major benefit for a sender to be relieved of the burden of tracking his correspondents. This is a result of the fixed location of a mailbox (as contrasted with the telephone, TWX, or TELEX).

Multiple copies: Much interpersonal communication is usefully shared by more than two people; for example, a small group of researchers working on a common project. This fact is perhaps best exemplified by the growth in office copier usage since its introduction. A piece of network mail can be sent to several mailboxes as easily as to one. This is perhaps the most important factor in the use of mail; unlike the telephone, it is easy to communicate among more than two people simultaneously, while unlike the postal service the interactions can be at electronic speeds. This is a result of the on-line (electronic) "storage" of the message at its point of generation, which makes it trivial to produce additional copies.

The conscious (or perhaps unconscious) recognition of these factors was instrumental in convincing large groups of network users to begin communicating through the mailbox facilities available on most of the time-sharing Hosts. However, as network mail began to supersede the use of postal and telephone services it became more and more obvious that one wished to deal with network mail in the same ways that have developed over the course of time for traditional office documents. That is, one wanted the ability to have lists of primary recipients, secondary (carbon copy) recipients, "blind" copies, file copies, and so on; dating and authentication (a signature surrogate) were seen to be desirable; it was useful to be able to "write notes in the margins", to forward messages, to keep specialized mailing lists, to engage in bulk mailings and so on. It also was seen to be useful to send messages to, and re- receive messages from, programs; for example, there are many "desk calender" programs which send daily reminders of scheduled events. In other words, the widespread use of simple network mail led inexorably to the desire for more automated office functions, as forecast by the developers of NLS. Thus, within a few years it seemed valuable to focus research on ways in which the network mail facilities could be extended to create a structured, multi-site "automated office".

Initially messages were thought of as simple text objects. As message manipulation programs became available, structure was imposed upon these objects to separate out a collection of header fields. The obvious next step was to view the message as a whole as a structured object. Conventions for interpreting the structure permits senders to generate, and recipients to distinguish, those components of the message which correspond to such classic message parts as the sender field. Thus the growth of more sophisticated mail processing programs led to the obvious necessity for mail formatting protocols (the mail itself is transferred from Host to Host via the File Transfer Protocol), and such protocols have been developed (on a relatively informal basis). Of course, the protocols not only specify the location, or method of identifying, the standard fields, but also the way to interpret the contents. For example, the "To" field should be thought of as a collection of addresses, the "Subject" field as a text string, the "Date" field as a date/time group, and so on.

More recently, some ARPANET message services have permitted extensions to the basic conventions by allowing message authors to define additional fields for their own purposes. Thus, communication on a fixed subject (for example, a company's contracts) could be facilitated by special message fields germane to that subject. Contract related messages might contain fields for contracting agency, start date, end date, and costs.

To enable a message program to carry out such operations as sorts and searches on these user fields, the data type of each such field must be made known to the program. Consequently, the agreed-upon structure of messages must be rich enough not only to support the existence of user fields but also to allow the inclusion in each such field of the data type of its contents. Thus the intention of the author can be conveyed to the recipient and to the program through which he manipulates his messages. Simply stated, messages must be somewhat self-describing, and to be able to interpret the self-description on a variety of Hosts a standard protocol will be required.

There are, of course, many open issues regarding network mail. First, although many investigators have asserted that some form of electronic mail will dominate the future of communications there are a number of stumbling blocks. Of these the least are economic and technical; far more important are the opposition of established communication groups and regulatory agencies [Panko77]. However, additional technical development is still clearly necessary to assure privacy and authentication, handle long documents (several printed pages) as conveniently as notes and letters, incorporate graphics (at least sketches and line drawings) in a simple way, and reduce the cost (dollars, weight, bulk) of convenient transmission/reception units (terminals) before it is reasonable to expect use of such systems by the average person for either business or personal communication.

3.5 Distributed Systems

The implementation of the ARPANET communication system created an environment in which it was possible to work on the development of multi-computer distributed systems. This section describes two such systems; the Resource Sharing Executive which was primarily implemented on TENEX systems, and the National Software Works which is a heterogeneous system. Also mentioned is the TIP Service Facility which uses the facilities of RSEXEC to provide TIP services. All of these systems are "network-based" systems, which not only use the ARPANET for communication among components, but wouldn't even have a reason for existence in the absence of a resource-sharing communication system.

3.5.1 The Resource Sharing Executive

The resource sharing executive (RSEXEC) is a distributed, executive-like system that runs on TENEX Host computers [Thomas73]. By sharing resources among themselves the Hosts can provide a level of service better than any one of them could provide individually. Within the environment provided by the RSEXEC a user need not concern himself directly with network details such as communication protocols nor even be aware that he is dealing with a network. The RSEXEC system was developed as an experimental vehicle to explore a wide variety of network operating system issues ranging from the types of features that would be useful to network users, to system structures and mechanisms for implementing those features, to strategies for ensuring reliable, fail-soft performance.

The goal was for RSEXEC to provide access to the combined resources of the TENEX Hosts much as the TENEX operating system and command language interpreter (called the EXEC) provide access to a single TENEX Host. The RSEXEC was designed to serve both as a command language interpreter for users and as a program execution monitor for user programs.

The RSEXEC system has three principal components: the RSEXEC program; the RSEXEC Server (RSSER) Program; and the RSEXEC/RSSER protocol (see Figure 3-2). There is an instance of the RSEXEC program for each active user of the system. An instance of the RSSER server program runs on each of the Hosts. Its task is to make the resources of its Host accessible to remote users on demand by accepting requests made upon it by remote RSEXEC programs. The RSEXEC/RSSER protocol is the set of conventions governing the interactions between the RSEXEC and

Figure 3-21 Components of RSEXEC

RSSER programs. It defines a set of commands and responses necessary to support the various RSEXEC system features

Although primarily a homogeneous system comprised of similar TENEX Hosts, dissimilar Hosts (Multics, IBM 360/TSO, PDP-10/ITS) have been partially integrated into the RSEXEC system [Forsdick77]. The method of integrating a new Host type into the system is to implement RSEXEC and RSSER programs for the new

Host. The functional requirements of the RSSER program are specified by the RSEXEC/RSSER protocol which was designed to be independent of Host type. The philosophy for the RSEXEC program is that for a given Host type the program should provide local users and programs access to the combined resources of the RSEXEC Hosts in the same style that the local operating system provides access to local resources. Thus, a user's interactions with a TENEX RSEXEC program would be similar to his interactions with a single Host TENEX system, while a user's interactions with a Multics RSEXEC program would be similar to interactions with a single Host Multics system. Although the style of interactions could be expected to vary from Host type to Host type, the functionality supported by RSEXEC programs for different Host types would be similar.

The usefulness of multi-Host systems such as the RSEXEC is, to a large extent, determined by the ease with which a user can manipulate his files. Because the Host used one day may be different from the one used the next, it is necessary that a user be able to reference any given file from all Hosts.

The file handling facilities of the RSEXEC were designated to:

1. Make it possible to reference any file on any Host by implementing a file name space which spans across Host boundaries.

2. Make it convenient to reference frequently used files by supporting "short hand" file naming conventions, such as the ability to specify certain files without site qualifications.

Uniform file access is achieved by providing a network-wide file naming syntax which extends each single Host file naming syntax for full file pathnames by adding a Host name field. Thus, any file in the system can be specified by its full RSEXEC pathname which includes its network location. This full pathname is interpreted in the same uniform way regardless of the Host at which it is generated. Convenient file access is achieved by allowing the use of partial pathnames to specify frequently referenced files. These partial pathnames are interpreted with respect to the user's own private working directory maintained by the RSEXEC program. A working directory, in general, may span Host boundaries in the sense that it may catalogue files from several Hosts.

The working directory contains an entry for each file in each of the component directories specified in the user's profile. At the start of each session the RSEXEC adds current detail to the user's directory by gathering information from the server programs at the Hosts specified in the user profile. Throughout the session the RSEXEC modifies the directory, adding and deleting entries, as necessary. The directory contains frequently accessed information (e.g., Host location, size, date of last access, etc.) about the user's files. It represents a source of information that can be accessed without incurring the overhead of going to the remote Host each time it is needed. The user's "profile" mentioned above is explicitly controlled by the user, who can add or remove components to control the set of Hosts over which, for example, partial pathnames are interpreted.

The user can take advantage of the distributed nature of the file system to increase the "accessibility" of certain files he considers important by instructing the RSEXEC to maintain images of them at several different Hosts. With the exception of certain special purposes files, the RSEXEC treats files with the same pathname relative to a user's directory as images of the same multi-image file. For example, the user profile is implemented as a multi-image file with an image maintained at every component directory.

File operations initiated by executing programs are handled by a technique called "encapsulation". The RSEXEC program intercepts certain file system operating system calls made by programs before they reach the local operating system. It interprets the file operations in the context of the network file system as described above. Operations referencing local files are passed directly to the local operating system. Remote operations are forwarded to the appropriate remote RSSER server program. An important benefit of this encapsulation technique is that programs written for a single Host environment need not be rewritten to operate within the network environment. Thus, the value of existing software, such as text editors and language processors, is significantly enhanced since the software execution environment includes remote as well as local data files.

Another important class of RSEXEC features are concerned with system status and interactions among users. These features are supported by commands which allow users to obtain information such as the current loads of the constituent Host systems (e.g., the number of active users, load factors, etc.) or the particular users currently logged into the various systems. In addition, a user can request that his terminal be directly "linked" to the terminal of another user (who may be logged into either the local system or a remote system) in order to engage in an on-line dialogue. At the user interface remote links are initiated and terminated in the same way as local links. However, remote linking is accomplished by cooperation between the RSEXEC program and a remote RSSER program, whereas local linking is directly supported by the local operating system.

Since the RSEXEC was originally created as an addition to existing autonomously administered TENEX systems, both its conception and implementation preserve the prerogatives of each of the Hosts. Each Host participating in the system totally maintains its own descriptive and state information about the resources that it supports. Thus, any RSEXEC Host can continue to operate autonomously, although with diminished scope. Such autonomous operation can, however, ultimately lead to conflicting interpretations of global file names. No attempt is made to maintain a long-term system-wide data base of network resources. Any replication of resources (e.g. multiple copies of a file) is evident only when an environment is dynamically configured for a user.

The total autonomy of the RSEXEC Hosts has the further administrative implication that a user must register individually with each Host whose resources he wishes to utilize. This involves establishing valid accounts, acquiring sufficient resource guarantees, setting up appropriate access controls, and so on for each constituent Host. Once this has been done, RSEXEC facilitates the use of the resources on these Hosts. Thus, although the RSEXEC eventually helps the user cope with the distributed environment, the user is still forced to first view it as the collection of independent, and perhaps even competing, entities which predated the ARPANET.

3.5.2 The National Software Works

The National Software Works (NSW) is an ARPANET-based system [Robinson76] designed to provide programmers access to a large range of software tools, e.g. text editors, compilers, assemblers, and debuggers, which can be used in the software development activity. From the standpoint of the programmer or program manager, the NSW environment consists of numerous software development tools, running on a variety of computer systems which are geographically and administratively distributed across the country, but accessible through a single access-granting, resource allocating monitor with a single, uniform file system.

Active work on the NSW development began in July 1974 under the joint sponsorship of the Air Force and ARPA. Initial operating capability using NSW components residing on two different computers (IBM 360/91 and PDP-10) was demonstrated in July 1976. The NSW project was conceived of as a possible way to attack the high cost and poor reliability of software within DOD by applying the results of ARPA research programs, especially the ARPANET.

It was noted that the types of tools mentioned above, which span the software development process from design through implementation and checkout, have been available on a variety of computers for many years, yet they are seldom applied together in an effective way to support software implementation projects [Forsdick77]. One reason is that existing tools have been implemented for different computer systems, and programmers typically do not have access to the range of machines that house them. Furthermore, even if they did, programmers would have to master a variety of different Host systems and command languages, and deal individually with basic interHost incompatibilities to use the tools. As noted, the NSW system addresses both of these problems.

The major components of the NSW are the front-end (FE) systems through which the users access the NSW, the access granting, resource controlling central component called the Works Manager (WM), Foreman (FM) modules that interface tools on the Tool Bearing Hosts (TBHs) to the WM, and communication protocols (MSG) that specify the communication links between the various NSW components.

A scenario of how NSW can be used illustrates the nature of the environment for software development it provides. The NSW system includes as TBHs PDP-10 Hosts operating under TENEX, Honeywell 6000 Hosts operating under Multics, and an IBM 360/91 operating under OS. Among the user community is a group developing software for the AN/UYK-20 computer, a small computer that does not itself support a wide range of software development aids, and is generally configured for production use rather than program development. The NSW will support the edit-compile-debug cycle for this group in the following way. Interactive text editors that run on TENEX and Multics are available for program preparation and modification. The source programs will be written either in assembly language or a higher level language for the AN/UYK-20. These source programs will be assembled or compiled by language processors that run on the 360/91 and loaded to form executable AN/UYK-20 object modules by loaders that can run either on the 360/91 or on TENEX.

Interactive debugging for the AN/UYK-20 programs will be done within NSW using a AN/UYK-20 debugger tool which runs on a TENEX TBH. This interactive debugger is, in fact, a multi-computer tool. Part of it runs on TENEX and part of it runs on a microprogrammable computer, the MLP-900, which is connected as a peripheral to one of the TENEX TBHs. The MLP-900 has been programmed to emulate the AN/UYK-20 and operates under the control of the TENEX interactive controller/debugger module. After the several edit-compile-debug cycles required to produce a debugged AN/UYK-20 program, the user can "export" his debugged AN/UYK-20 load modules from the NSW for final checkout and operation on an actual AN/UYK-20 machine.

To use any tool, a user simply specifies the tool by name (e.g., the TECO text editor, the AN/UYK-20 compiler, the AN/UYK-20 debugger) and the NSW system starts an instance of the tool and connects the user to it. The user need not know which

TBH supports the tool nor the command language of the particular TBH. He need only learn the NSW command language. To support tool execution the NSW implements a type of distributed file system. All NSW tools utilize this single NSW-wide file system for their file references. When a tool attempts to access a data file (e.g., a source program for a compiler, a load module for a debugger) the NSW system ensures that the access references the correct file independent of its actual location. If the referenced file is stored on another Host system, the file is automatically transported to the referencing TBH so that the access can be completed.

In addition, NSW may perform file transformations to deal with basic interHost file system incompatibilities. Thus, the user and the tools he employs need not concern themselves with the location of data files, the details of the file systems on the various constituent TBHs, the movement of files between Hosts, or the data translations that may be required to make a file created on one Host type usable on another Host type. Rather, they interact directly with a single, uniform NSW file system.

Each active user has a dedicated FE process which acts as his interface to the NSW system for the duration of his session. The FE acts principally as a command language interpreter making requests upon other components as necessary to satisfy user commands. The WM is the resource allocation and access control module for the NSW system. All attempts to access NSW resources, such as tools or files, must be authorized by the WM. To perform its task, the WM maintains system-wide data bases such as an NSW file system catalogue, tool descriptor information, user password and account information, and so on. Interactions between WM processes and other system components occur on a transaction oriented basis. That is, the system does not dedicate a single WM process to each active user for the duration of the user session. Rather, WM processes are dynamically allocated (and deallocated) as necessary to support a user session.

When a user requests the start of a tool, an FM process on the appropriate TBH is allocated to the user for the duration of the tool session. The FM process provides an execution environment for the tool and controls its operation. This execution environment differs somewhat from the standard environment provided to the tool by the local TBH operating system. For example, when a "file open" operation is initiated by a tool, the operation must be processed in the context of the entire NSW rather than that of the local Host operating system. The FM process responds to such an attempt by cooperating with a WM process to complete the file reference. The WM process consults the NSW file catalogue to verify the existence of the file specified by the FM, and the authorization of user and tool to access the file. Next, the WM acts to ensure that the file can be physically accessed by the FM/tool. In general, this may require movement of the file to the FM Host and possible translation of file data to a form usable by the tool.

Communication between the various NSW system processes well as the allocation and deallocation of those processes is the responsibility of a component called MSG. Two sorts of addressing are supported by MSG. Generic addressing is the means by which a process initiates a transaction with another previously unrelated process. Generically addressed messages are used when any process of a given type is acceptable. The second type of addressing is specific addressing. It is used when the sending process must communicate only with a particular process.

The FM component is designed to provide two different tool interfaces to NSW. An encapsulation interface is provided to permit software packages that were developed to run under a particular TBH operating system to be installed as NSW tools. This interface is similar in concept to the encapsulation mechanism used in the RSEXEC system as described in Section 3.5.1. The second tool interface to NSW provided by the FM is a direct interface which permits tools to explicitly call NSW functions. Generally the FM must interact with other NSW system processes to satisfy these tool calls. This interface is provided for tools developed specifically for operation within the NSW environment.

The NSW file system is built upon the file systems of the constituent Hosts. The file systems of the Hosts are used simply as a storage pool for NSW files. The WM maintains a file catalogue which contains an entry for each NSW file. A file entry includes the NSW name for the file, the location of the file in one (or more) of the constituent Host file systems, and various file attributes. NSW file operations are logically centralized in that the central WM file catalogue is (almost) always used to resolve file references.

When a tool opens a file, a copy of the file is moved from the NSW file storage space into a tool workspace maintained by the tool's FM. The file manipulated by the tool is truly a copy in that any modifications the tool makes to it will not be reflected in the actual NSW file maintained by the WM unless the FM/tool explicitly delivers the copy back into the NSW file system.

Because of its modular structure, the NSW system architecture is potentially resilient to individual Host failures. Indeed the inter-component protocols have been designed to make continued system operation possible in the presence of FE and FM/tool Host failures. In addition, mechanisms have been developed to recover files trapped in a tool workspace due to a TBH crash. When a crashed TBH is restarted, tool workspaces in use when the crash occurred are preserved in a way that allows a user to later retrieve selected files. At present, the centralization of the WM is the principal weakness of the system from a reliability point of view. Because the WM and its central data bases reside on a single Host, the NSW system is vulnerable to failure of this Host. A multi-Host implementation of the WM function would permit continued NSW operation in the presence of WM Host failures. In addition, it would also allow the system to gracefully expand as the number of users grows by making it possible to distribute the WM load among several Hosts. The principal technical problem here results from the fact that the WM data base is logically centralized. To distribute the WM, the data base must be distributed. This requires the development of synchronization mechanisms to insure that the distributed parts of the data base are maintained in a consistent manner.

An individual computer system can be an NSW TBH and at the same time directly service non-NSW users. However, the resources supporting the TBH implementation are dedicated exclusively for this purpose, and cannot ordinarily be manipulated from outside the NSW environment. This helps maintain the integrity of the globally maintained system resource data base.

The administrative centralization of network resources in NSW means that procedures for establishing access rights to these distributed resources are quite simple. One merely negotiates with a single NSW administrative organization to arrange an account which potentially allows access to all NSW resources. The logical centralization of authentication, resource allocation and access control functions within a single component simplifies the software implementation of these functions. It also means that there is a convenient framework for accounting for NSW resources utilized by an individual user.

The NSW attempts to shield the user from almost all details of network operation, and emphatically tends toward the "invisible distribution" end of the design spectrum. This position is based on the feeling that dealing with the network or the constituent Host systems at any level would impair a user's ability to easily utilize combinations of software development tools. However, there are a number of factors which suggest that a more "visible" approach to network systems is sometimes more appropriate. For example, there is frequently an appreciable decrease in performance when remote resources are accessed. Also, various similar system resources may have slightly different characteristics on different Host types. It is still too early in the development of network-based distributed systems to assert with confidence that one can identify the best operating point in this (visible/invisible) design spectrum.

3.5.3 The TIP Service Facility

As noted in the discussion of the TIP system in Section 3.2, the users (and administrators) of a small computer such as the TIP will almost always desire more services than the small computer can provide. In particular, because of memory limitations, the TIP is incapable of providing its users with a sophisticated command language. The TIP has no space to hold tables of passwords or statistics on its usage; thus, the TIP has no capability for access control or accounting. The TIP cannot distribute operational information to its users, such as announcements of system changes. What the TIP does provide is a relatively transparent, simple, flexible, and high performance interface between a terminal and the network. However, if access control, accounting, and operational capabilities are to be provided, it is necessary to devise a mechanism to obtain these capabilities elsewhere.

During the development of the RSEXEC system two relevant observations were made. First, since many of the features planned for the RSEXEC were well matched to the desires of TIP users, with some additional effort the RSEXEC system could provide TIP users with a sophisticated command language and other features they desired [Cosell75]. Second, because the RSEXEC was to be run on several systems, RSEXEC could potentially provide capabilities to the TIP very reliably. With a single Host providing a function, there would be times at which that Host would be down when some TIP user required the function. Thus, it would be possible through TIP use of the RSEXEC to obtain TIP capabilities superior to any the TIP could provide itself or that could be provided with the help of any single other Host.

Two mechanisms were developed to support the redundant implementation. The first is a "broadcast" Initial Connection Protocol. This enables a TIP to connect to an available and responsive RSEXEC rather than to a particular version at a specific site. Using this mechanism, a TIP broadcasts requests for service to the known RSEXEC sites and then selects the site that responds first as the one to provide the service.

The second is a mechanism to maintain multiple copies of the various information files (e.g., network news and Host schedules) at the RSEXEC sites. This mechanism allows additions to these distributed information files to be initiated from any RSEXEC site and guarantees that the additions are incorporated into each file image in a consistent manner.

Both of these mechanisms have been in operation for several years, supplying desired facilities to TIP users upon demand. Perhaps more interesting, however, was the extension of this system to several months of operational use in providing TIP access control and collecting TIP accounting information. These uses are currently discontinued because of a number of broader administrative issues, but while active the access control and accounting procedures function with a high degree of success.

During times when the mechanisms are active, the TIP code is changed so that whenever a user activates a TIP port, the TIP uses the broadcast ICP mechanism to connect to an RSEXEC which acts as a network login server. If the user successfully supplies a valid name and password, he is granted continued access to the TIP, the network, and to the standard RSEXEC functions. In addition, the RSEXEC transmits the user's network ID code (which serves to uniquely identify the user for authentication and subsequent accounting purposes) to the TIP and makes a "login" entry into a "TIP accounting" data file. If the user fails to supply a valid name and password then the RSEXEC does not forward an ID code to the TIP. If the TIP does not receive an ID code within the allowed time the TIP disconnects the terminal and the user is denied further access.

After the TIP receives the user's network ID code it activates "connect time" and (outgoing) message counters to accumulate usage data for the user's session. These counters remain active until the user terminates his TIP session. Periodically the TIP executes an "accounting checkpoint" procedure whereby it transmits usage data for its active users, accumulated since the last checkpoint, to a data collection server process. The data collection server stores the checkpoint data in an incremental TIP accounting file for later processing.

Like the login servers, the data collection servers are redundantly implemented to insure high availability and to achieve load sharing. The TIP uses a request mechanism similar the broadcast ICP to select one of the servers to accept its checkpoint data. The protocol is designed to allow considerable flexibility in the choice of a server. For example, a TIP can switch from one data collection server to another after initially choosing one in the event that the chosen server can not complete the transaction (for example, because of network or Host failure).

The collection of incremental accounting files created by the data collection servers is a large distributed and segmented data base. Some checkpoint data for a given TIP session may be collected by a data collection process at one site while other parts of the data for the same session are collected at other sites. The reduction of data in this distributed data base to produce periodic accounting summaries is accomplished by software which executes within the environment provided by the RSEXEC distributed file system.

The TIP service facility uses two fundamentally different types of distributed data bases. The first is one which is maintained "identically" at a number of sites. The second type consists of distributed, non-overlapping segments; that is, the data base is a collection of segments, each of which is singly maintained at a (possibly) different location. These two types represent extremes of distributed data base organization; other applications may call for "intermediate" types such as a data base consisting of a collection of segments of which some, but not all, are redundantly maintained.

Use of the first type of distributed data base is exemplified by two aspects of the TIP service facility:

1. The RSEXEC maintains a copy of the TIP news file at each of the RSEXEC sites. Updates to the news file are limited to addition of news items. The system allows additions to the data base to be initiated at any RSEXEC site and insures that all such updates are transmitted to and incorporated into all copies of the data base.

2. The TIP login system requires that the network user ID data base be maintained in a consistent manner at all RSEXEC sites. Each copy of this data base is a collection of

mutually independent user entries. Allowable updates to this data base include the addition, modification, and removal of individual user entries. The data base management techniques allow updates to be initiated at any site and guarantee that they are consistently incorporated into all copies of the data base (i.e., if all updating activity were to cease, all copies of the data base would eventually be identical).

The techniques used to maintain the data bases consist of two independent parts:

1. A reliable, data-independent, update transmission and distribution mechanism which uses persistent processes at the update entry sites to guarantee that all updates are eventually delivered (once, and only once) to all data base sites.

2. A data-dependent update action procedure which is activated at data base sites whenever update commands arrive.

The operation of the TIP accounting system results in the creation and manipulation of the second type (i.e., segmented) of data base. The primary concern in the accounting application was data base organization and convenient data access. The specific data base issues that required attention were:

1. Cataloging: It is obviously important to know where the various data segments (incremental accounting files) reside so that they can be accessed. This cataloging function is provided by the RSEXEC distributed file system.

2. Insuring that no duplicate entries occur in the data base. Because the entries contain accounting information, it is critical that any redundancy does not cause duplicate charging. The data collection protocol was carefully designed to prevent the occurrence of duplicate data entries in spite of the fact that data is broadcast to all of the servers.

3. Insuring that each data base entry is processed exactly once when accounting summaries are produced. Time stamping plays a fundamental role in guaranteeing "once only" processing.

The implementation of the TIP service facility is interesting primarily because it provides a working demonstration that it is possible to provide easy access to large and complex functions in a transparent way from much simpler machines.

Further, the presence of multiple components in a distributed system, together with the potential for their redundancy, makes it possible to achieve reliability by constructing systems from modules most of which are kept relatively simple. By using simple modules, component failure due to malfunction of non-essential features can be reduced. Complex components are redundantly supported in an effort to enhance their reliability. Such an approach takes full advantage of both the heterogeneity and homogeneity of various network components. The important issues in designing a system of this type are the assignment of functions among the various machines, the degree of redundancy required, and the protocols used to bind the system modules together.

Once such a virtual executive is conveniently available to TIP users, it becomes possible to think of additional features that can be added. For instance, the RSEXEC makes available to Host users a file system which spans machine boundaries. It is a simple technical step to provide the TIP users (who, unlike the Host users, have never had a file system) with a virtual file system. Another example: while the RSEXEC has the capability to permit users to leave messages for other users, it does not provide the capability for TIP users to receive such messages.

Yet, through the concept of resource sharing, the potential capability to provide virtual mailboxes through which users can receive messages exists. Furthermore, through the redundancy inherent in the system, these mailboxes could be provided in a way which would insure that a user's mail was accessible no matter which individual computers were down. A final example: once the TIP user is connected to the RSEXEC and is ready to use the services of some Host, and once it is possible for the user to call for service independent of Host, there is no need to retain in the user's view the concept of the Host(s) from which service is obtained; rather, the virtual executive could be expanded to provide the virtual operating system from which all service is obtained. Such a concept combines the physical "front-end" properties of the TIP with the logical FE/WM partnership of the NSW.

3.6 Network-based Experiments and Research

The ARPANET subnet of IMPs and communications links was created primarily to provide a communication service to existing Host organizations and user populations. These groups have, over the period of network operation, been involved in many areas of research involving programs implemented on the Hosts; these programs may be categorized as "Host-based" research. However, another class of research projects and experiments have come about as a result of the existence of the ARPANET; this class may be characterized as "network-based" in that it seeks to evaluate or extend the utility of the packet switching concepts embodied in the ARPANET.

This section discusses several of the network-based experiments and research programs which were instituted prior to the time the management of the network was assumed by DCA; many of these research programs are continuing to the present. It is also worth noting that an extensive program of research in extending packet techniques to radio networks is underway [Kahn75], but is not discussed here because its interaction with the ARPANET is minimal.

3.6.1 AFCS Experiment

Although ARPA itself provided funding of the Network Control Center, which measured network performance from an operational point of view, and the Network Measurement Center, which ran a number of performance experiments involving very large measurements, the first (and perhaps only) outside evaluative measurements of the ARPANET were carried out by the Air Force Communications Service in 1972 [AFCS72]. The experiment was designed to evaluate the suitability of the ARPANET for transporting large volumes of traffic between pairs of sites; specifically, the performance and cost were to be compared to the Circuit Switching Unit (CSU) of AUTODIN.

The experiment was defined, in early 1971, to be based on the interconnection of existing Air Force Univac-418 computers at Tinker Air Force Base (Oklahoma) and McClellan Air Force Base (California). ARPA agreed to install IMPs at these two locations, and assisted the AFCS in the definition of an ad hoc Host-Host protocol for use in the experiment. The protocol eventually defined used multiple link numbers for a single communication in order to achieve high bandwidth, and used a minimal Host-Host header in order to achieve high efficiency.

The actual measurements took place from mid-April to the end of June in 1972. The shortest path between the Tinker and McClellan IMPs during this period traversed five hops (i.e., five IMP-to-IMP circuits.

The test findings and conclusions which resulted from this experiment were:

1. Throughput between the two Hosts for all traffic averaged 25 Kbs.

2. For test traffic of known contents, 31,761 18-bit words out of 247,365,000 words transmitted were observed to be in error; this amounts to about .01%. Analysis led to the conclusion that most (or all) of these errors were introduced in the IMP/Host interface. Therefore, AFCS recommended that a hardware error detection scheme be implemented for this interface. In addition, it was recommended that a software error detection scheme be incorporated in the Host-Host protocol.

3. The cost per megabit transmitted was found to be approximately equal to that of the AUTODIN CSU at daily traffic levels below 168 megabits. At higher daily

traffic levels the ARPANET was less costly than AUTODIN.

4. It was recommended that further studies be conducted "to determine the feasibility of implementing a secure network, patterned after the ARPANET, for DoD users with large digital transmission requirements" [AFCS72].

3.6.2 Speech Transmission

There are several reasons for interest in transmitting speech signals in digital form through a packet-switched network. Perhaps the most important stems from an intrinsic difference between traditional circuit switched transmission and packet switched transmission as exemplified by the ARPANET; during periods of excessive demand circuit switched systems bar access to new calls, whereas packet switched systems may merely increase per message delay. The advantage of a guaranteed connection is so important in some government and commercial situations that dedicated lines are leased in order to provide it [Forgie75]. Packet networks should be able to provide the same guarantee at lower cost.

for "voice message mode" only, if available

+ integrated network
+ ability to use silence for other traffic

In addition, digitized speech offers other advantages. Digital transmission is less sensitive to noise, encoding a digital signal for privacy or security is easier, and buffering, processing, and archiving spoken messages on large Host computers is possible.

A major obstacle to transmission of speech through packet networks is the number of bits per second required to encode the speech signal. High fidelity analog-to-digital waveform conversion would result in about 250 kbs; provision of telephone-quality speech signal would require about 50 Kbs. For this reason there is currently a great deal of research in the area of speech bandwidth compression. There are two general approaches under investigation; these can be denoted the waveform coding approach and the vocoder approach. The waveform coding approach is based on the fact that successive samples of the speech waveform are not independent, allowing a restriction of the range of possible successors to a given state which the coding must represent, and thus a restriction in the code size. The vocoder approach is based on methods of modeling the speech process. If the modeling is successful, a speech signal can be represented by a set of parameters which require lower bandwidth for transmission than the signal itself. Demonstration vocoders have required as few as 1.2 Kbs for the signal characteristics of selected speakers, and adequate coding of most voices can be done in the 2.4-9.6 Kbs range.

ARPANET packet speech experiments have been based on the use of relatively low bandwidth, speech compression devices, mostly in the vocoder class. However, the majority of available devices have been constructed assuming connection to switched (or dedicated) communication channels of fixed bandwidth and delay; these characteristics, of course, do not pertain to the ARPANET. Thus the packet speech research and development has been designed to try to find solutions to two problems:

1. Finding ways to reduce the average transmission rate by taking advantage of the variability of the actual speech data rate, in order to avoid unnecessary network load.

2. Finding ways to mask the variability of ARPANET transmission delay when regenerating the speech signal, in order to avoid listener annoyance.

By increasing the complexity of the vocoder's interface to the communication channel, in particular by interfacing it to the ARPANET through a Host of modest size, it has been relatively easy to reduce the average data rate. For example, no bits at all need to be transmitted when the talker is silent, either pausing to think or to wait for the other party in a conversation to finish talking. Further reductions in average data rate are possible by taking advantage of the fact that during certain speech sounds the character of the sound changes much more slowly than its maximum rate.

It has been less easy to mask the variability of ARPANET transmission delay. Obviously, some buffering is required at each vocoder; at the source buffering is required to construct packets and hold them until the local IMP accepts them, and at the destination buffering is required to read the packets and feed the bits serially to the speech synthesizer. At first it appeared that by expanding the destination buffering to handle a few seconds of signal (a few thousand bytes of storage), and starting the synthesizer only when the buffer was partially filled (e.g., half) with input from the network, the variability could mostly be masked from the listener. However, the source IMP to destination IMP algorithms were designed for very reliable transmission; to achieve the desired level of reliability packets may be retransmitted many times in order to be sure they get through. In addition, the ARPANET promises to deliver data in the same order in which it was transmitted. Thus, if a packet must be retransmitted, the delivery of all subsequent packets will be delayed. Since source IMP to destination IMP inquiry and retransmission timeouts may be measured in tens of seconds, a

this is still a good techn. With 8 in the single packet pipe, Forgie says it works single lost packet may have a major disruptive effect on the speech receiver.

A human listener, of course, is well adapted to interpolating speech signal across bursts of noise. Thus, from the point of view of packet speech transmission, it would be preferable if the ARPANET did not try so hard to deliver the data reliably. An algorithm which gave up after a second or two and delivered subsequent packets without worrying about the gap would be much better. The protocol used for the transmission of speech packets could include timing information, and the synthesis device could fill signal gaps with a continuation of the immediately-preceding signal, white noise, or complete silence.

To summarize, the packet speech transmission requirements are neither for high bandwidth, low delay, nor almost perfect reliability; rather the requirements are for medium bandwidth, modest reliability, but very low variability of delay. Special Host-Host protocols can be used to reorder packets, detect missing packets, smooth the delay, and adjust the output signal appropriately. In fact, the ARPANET was modified to allow a restricted set of Hosts to send packet traffic which bypasses the normal source IMP to destination IMP protocols, precisely for packet speech experiments. Unfortunately, however, the IMP congestion control algorithms are intimately entwined with the missing packet detection and retransmission algorithms; thus unrestricted growth in the speech traffic might lead to congestion problems or other interference with the rest of the Hosts. Nevertheless, several demonstration conversations have been carried out over the ARPANET using these techniques.

3.6.3 Packet Broadcast by Satellite

The ALOHA System [Abramson70], developed at the University of Hawaii, introduced the use of a "multi-access" channel for packet communication. In this system, a number of independent transmitters accumulate packets of information; all packets are to be sent to a single central site. Whenever a transmitter has accumulated a full packet it appends its own address and a checksum, and transmits this packet on a fixed radio frequency. The same frequency is used by all transmitters, so two transmissions may overlap in time. If this happens, the checksum received at the central site will indicate an error and the (composite) transmission will be ignored.

A second frequency is used for all packet transmissions from the central site to the remote stations. Since this frequency is used by only one transmitter, conflicts do not occur. Address information in these packets allows the remote receivers to identify the packets intended for them. Packets correctly received at the central site are answered with positive acknowledgments sent along with central site data. Thus, when a remote station sends its packet it starts a timer; if the timer expires before the acknowledgment arrives the packet must have been destroyed by channel noise or conflict and is retransmitted.

The capacity of the multi-access channel is obviously less than the rated channel bandwidth, due to conflicts and retransmissions. In fact, under certain reasonable assumptions about frequency distribution of packet arrivals, it has been shown that a channel operated in this way has a capacity of about 18% of rated bandwidth. However, it has been noted that if such an ALOHA channel were divided into slots (each able to hold a packet) and if the nodes modified their behavior to transmit packets so that the leading edge of the packet always coincides with the leading edge of a slot, even though the nodes remain free to transmit into a slot without regard for the transmission of other nodes, the effective channel capacity is doubled to 36%. A channel operated in this manner is called a "slotted ALOHA" channel.

Packet broadcast by satellite extends these ALOHA concepts to a geosynchronous satellite transponder and an ARPANET-like environment. The principal differences are:

1. There is no central site. All stations are equal in their access to the channel; each accepts only the packets addressed to it.

2. The long round-trip time makes it worthwhile to attempt to schedule portions of the channel (in a highly dynamic way) for any portions of the traffic with known characteristics.

3. Since there is no conflict-free frequency portion of the channel on which to send positive acknowledgments, it may be worthwhile to schedule a conflict-free time portion of the channel for each transmitter to send acknowledgments. This may conserve bandwidth by helping to prevent acknowledgments from being lost, with needless retransmissions resulting from such loss.

In line with these ideas, ARPA has commissioned an experimental packet satellite network to carry out tests of a variety of channel protocols and operational nodes. This network required the development of a new Satellite IMP which controls the use of the channel. The Satellite IMP is a conventional IMP with several additions and modifications both in hardware and software [Butterfield74].

The first hardware addition is to increase the memory to a size sufficient to buffer all the transmitted packets which can be awaiting acknowledgment simultaneously. Buffer space is necessary for some 32 packets assuming a 50 Kbs channel and a one quarter-second propagation up to the synchronous satellite and back down. That is, 32 packets can be sent out before the acknowledgment returns for the first.

The next hardware addition is a mechanism for signaling the satellite radio transmitter when to turn the radio carrier on and off as packets are transmitted; this is necessary because if two satellite ground stations have their radio transmitter carriers on simultaneously, they jam each other.

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

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

The Satellite IMP software effectively makes the Satellite IMP an extension of an ARPANET IMP, using the satellite channel to provide an additional ARPANET link between the Satellite IMPs. The satellite channel protocol, however, is designed to allow many Satellite IMPs to share the channel. Because of its special nature, the satellite channel and its associated Satellite IMPs are most conveniently treated as an independent network temporarily interconnected with the ARPANET.

For experimental purposes, one of four different multiple access broadcast channel protocols can be selected by a software switch in the Satellite IMPs. These protocols consist of fixed TDMA, slotted Aloha, a rudimentary version of Reservation-Aloha, and Reservation-TDMA. Each of these is a demand assignment (DA) system, although the fixed TDMA protocol allows the DA only through its variable destination (packet address) capability. The other three also have this capability, but in addition allow channel transmission time to be shared among the Satellite IMPs in a completely dynamic manner.

Based on some experience with these protocols, as well as continuing analytic work, a contention-based priority-oriented demand assignment, or CPODA, channel protocol has been under development. The goal of the CPODA protocol is to provide a very flexible multiple-access demand-assignment system for use in a broadcast satellite channel, while also allowing users to specify a number of different priority levels and delay constraints. Further, the protocol is designed to allow the integration of both block and stream traffic, where the latter refers to packet voice and similar types of traffic having a relatively short delay constraint on each packet along with a relatively constant bandwidth requirement over the time span of a conversation.

A major feature of the CPODA protocol, which distinguishes it from the Reservation-TDMA protocol currently implemented in the Satellite IMP, is its use of a random access technique for making reservations. Channel time is divided into subframes, with one subframe used in a random access manner by all stations to make reservations, while the other subframe is used for the resulting scheduled transmissions of both block and stream data packets.

To allow experiments to be performed with these protocols unencumbered by ARPANET source-destination protocols and uncontrolled traffic sources, the Satellite IMP software also includes the following additions:

Satellite IMP Message Generator: This code provides a controlled source of experiment traffic which is not subject to IMP source-destination protocol, and provides a wide range of message lengths and rates.

One-way Delay Measurement Code: This code makes use of the synchronized clocks provided by the Satellite IMP's slotting algorithm to measure the delay encountered by each message between its time of origin in one Satellite IMP and its correct receipt in the destination Satellite IMP.

Access Control Code: This code overrides normal IMP routing decisions, allowing only experimental traffic to use the satellite channel while forcing all other traffic over normal ARPANET circuits. To allow use of the satellite channel as a backup link for ARPANET when the normal trans-Atlantic path is disrupted, the code automatically switches, when necessary, into a "normal" routing mode.

The Satellite network consists of two Satellite IMPs located at large-antenna (30 meter) earth stations in Etam, West Virginia and Goonhilly, England which have been in place for some time. The system uses a 50 Kbs satellite channel supplied through the SPADE subsystem. Two additional nodes were installed in the Satellite Network later in the project; these are located at the Scandinavian large-antenna site in Tanum, Sweden and at a COMSAT laboratory small-antenna site in Clarksburg, Maryland. The Etam, Goonhilly, and Tanum sites are each directly connected to the ARPANET via IMP-to-IMP circuits, so for a considerable period of time these Satellite IMPs have been operated as an extension (or subnet) of the ARPANET proper. However, the long-term intent is to operate the Satellite Network as an independent, although interconnected, system.

3.6.4 Internetting and Gateways

The work done to date on the interconnection of computer networks has usually assumed a configuration such as that shown in Figure 3-3. On each network there are Hosts (denoted by H in the figure) which desire to communicate with Hosts on other networks. The networks are connected together by units (denoted by G) called "gateways". The gateways must in some way convert traffic in the format of one network into traffic in the format of another network.

Because Host-Host protocols differ from one network to the next, and because these protocols are generally complicated and incompatible, it seems clear that Hosts on different networks wishing to communicate must do so in a common protocol. Therefore, much of the work to date in network interconnection has been the specifications within such bodies as IFIP Working Group 6.1 [Cerf76] and more recently ISO of common end-to-end protocols. However, since standardization efforts tend to take a long time, ARPANET experimenters have "standardized" on an end-to-end protocol described in 1974 [Cerf74]. In this protocol the logical entity in the Host which performs the protocol functions is called the Transmission Control Program or TCP.

Figure 3-3: Networks Connected by Gateways

Given a Host protocol (TCP) and a pair of networks of somewhat different characteristics (ARPANET and the Satellite

Network described in the preceding section), ARPA has been involved in the definition of experiments to investigate the functions that gateways should perform in a network interconnection environment. In fact, by the time that the ARPANET to Satellite Network gateways (three of them) achieved initial operation in mid-1977, plans had been formalized for pairwise gateway between:

- ARPANET and ARPA's experimental packet radio network
- ARPANET and an internal network at BBN's Research Computer Center
- ARPA's experimental packet radio network and ETHERNET at the Palo Alto Research Center of Xerox
- ARPANET and an internal network at MIT's Laboratory for Computer Science

as well as probably expansion of the ARPANET to Satellite Network gateway in England by addition of a connection to the Experimental Packet Switched Service of the British Post Office.

One of the outstanding questions of network interconnection is whether the gateways should connect networks at the packet or Host level. At packet level, a portion of the gateway would actually become a node on each of the networks being connected, while at Host level, a portion of the gateway would actually be a Host on each of the networks being connected. In the current experiments, the gateways connect at the Host level primarily to maintain a "sovereignty" of the networks involved. This choice allows connection to the networks at a point where the interface is both well defined and well controlled by each constituent network. Each network can protect itself against activities of the gateway to the same extent as it may protect itself against the activities of any other Host.

Since the gateways connect to the networks as Hosts, then the format of the messages passed to a network are specified by the Host/network protocol. This protocol is then used to permit transparent transmission of segments of an internetwork message by embedding the internetwork segment in the text of a local network message. Such a composite unit has two leaders and potentially two trailers. The outermost leader and trailer provide information for the local network and will be different in each network through which the message passes. The leader will specify the address of the gateway Host to which the message should be delivered, any allocation or sequencing information which is used by the Host/network protocol, and any further information demanded by that protocol. An example of a trailer that might be required by the Host/network protocol would be padding and a checksum. Within this outermost leader and trailer is the internetwork data segment with its leader and trailer. The internetwork leader specifies such information as the ultimate destination, sequencing, and reassembly information. The actual data which is being transferred is the text of this segment.

In addition to the ability to accept messages from one network and resend them into another, it is likely to be desirable for the gateways to have the following additional characteristics:

Routing Capability: Inter-gateway routing is desirable for all of the standard reasons routing is desirable in a network. For example, for reliability there should be alternate paths over which traffic may be routed; for achieving higher bandwidth than is available over any single path, gateways should be able to route traffic over parallel paths; different classes of traffic need to follow different routes (e.g., traffic requiring low delay should be routed around networks which insert large delays).

Access Control and Accounting Mechanisms: A given constituent network may wish to limit some classes of traffic or all traffic at some times (e.g., because of regulatory considerations, country A might not want traffic from country B to country C to pass through country A's network). Also, a given user might not wish his traffic to pass through some particular constituent networks, e.g., for political reasons. Alternatively, the constituent networks are very likely to want to monitor the use of their network by internetwork traffic.

Ability to Perform Message Fragmentation: Because of the differences in message size of the constituent networks connected by a gateway, the gateway must have the ability to fragment a larger message arriving from one network into smaller messages which are acceptable by the next network. When such fragmentation occurs, the message stream must eventually be reassembled into its original structure. The TCP protocol provides this function at the destination Host.

Congestion Control: Congestion will inevitably occur at the gateway unless specific measures are taken to prevent it. This congestion can occur as a result of speed mismatches between the networks connected by a gateway, because several gateways on a network may simultaneously transmit traffic to a single destination gateway, because traffic may have to be held during a period of recovery from a failure, and so on. One specific kind of congestion results from deadlocks, such as when gateway A is full of traffic for gateway B which is full of traffic for gateway A.

Retransmission Capability: When a message is lost in the network between two gateways, either the last gateway can retransmit the message or the message can be retransmitted from the source Host to the destination Host. It has been shown that hop-to-hop retransmission is more efficient than source-to-destination retransmission if the possibility of message loss is appreciable; and even when there is little possibility of message loss, the variance of retransmission delays is less with hop-to-hop retransmission than with source-to-destination retransmission. While some networks deliver messages very reliably, other networks rely on source to destination retransmission and in some cases are quite cavalier about throwing away messages. Thus, it seems that at least when the Hosts on a network are normally responsible for retransmission across that network, the gateways ought to provide retransmission.

A gateway with the characteristics described above is very similar to a node on a packet switching network. This leads to the notion of a gateway virtual network wherein the gateways act as nodes and the network spanned by the gateways acts as virtual lines fully connecting all the gateways on that network. Further, there is a gateway logically associated with each Host attempting internetwork communication, providing at least some of the functions (e.g., routing, but perhaps not retransmission) which must be provided between networks. However, there need not necessarily be a one-to-one correspondence between Hosts and gateway machines. For instance, the logical entity that is the gateway may have the form of a program running in a Host computer. Alternately, the gateway could be in a stand-alone machine serving one or more Hosts. On the other hand the gateway connecting two networks could take the form of a program running in a Host which is connected to both networks. In general, a gateway should be able to connect any combination of Hosts and networks.

It is entirely possible for a Host not to have its own gateway, preferring to use a gateway elsewhere in its network to perform the gateway functions it needs. In this case, the Host would simply know the addresses of a few gateways in its network, and would send its internetwork traffic arbitrarily to one of these gateways for routing and forwarding. Of course, the gateway arbitrarily chosen might not be on the best path to the destination, and this gateway might even forward the traffic to another (better) gateway in the same network for forwarding outside the network. Thus, this approach may have some inefficiencies but it reduces the number of gateways that have to be constructed.

The gateway network has many of the maintenance characteristics of a stand-alone packet switching network. It requires centralized development and maintenance responsibility, including a gateway network monitoring and control center.

The Gateway Virtual Network solution is a general solution to the problem of interconnecting networks which is not highly dependent on the nature of the networks being connected. Because of this it can be expected that additional networks which are connected in the future will not require modification. A principal goal of the Gateway and Internetting experiments currently underway is to discover whether this expectation can be met.

3.6.5 End-to-End Security

As described in Section 3.3, it has been possible to build equipment (the PLIs) which allows use of the ARPANET for the transmission of classified data in a secure way between pairs of Hosts, and it appears likely that use of a multi-address capability for this equipment will be obtained.

Nevertheless, the PLI (even the multi-address version) is far from ideally matched to the possibilities offered by the network. For example, the PLI is too large and expensive to be useful for terminal connections. The KG units at all PLIs must use a single key to allow general communication; however, it might be desirable for A and B to engage in communication secure from C and for A and C to engage in communication secure from B.

For reasons such as these ARPA has engaged in a program of research on end-to-end security, in an unsecured packet network environment, with the following goals:

- Provide secure communications over a packet-oriented network
- Provide end-to-end encryption for both Host computers and terminal users

- Avoid the use of classified equipment in the experimental system
- Provide a system as close to militarily secure as possible, with an eye to certification of a similar system
- Achieve efficient use of the packet-oriented network
- Design a system which could be deployed cost effectively
- Operate in an internetworking environment, illustrated with at least the ARPANET, the Packet Radio Network, and the ARPA Satellite Network
- Make use of proven techniques where possible
- Minimize impact on Host operating systems; and, in fact, if possible make the security system transparent from the Hosts' point of view.

The encryption unit developed in this program is referred to as a BCR. The letter B indicates the black side of the unit, the letter C an interior crypto or KG portion of the unit, and the letter R the red side of the unit. This three component unit taken as a whole provides a secure interface between the Host and the network. The B portion of the unit is involved both in interfacing between the BCR and the IMP and in an end-to-end (B to B) protocol. The R portion of the unit is involved both in interfacing between the BCR and the Host (or terminal) and in an end-to-end (R to R) protocol.

Each BCR contains two LSI-11s, one as the Black side processor and one as the Red side Processor. The Crypto unit is a fairly complicated unit with storage for several different Keys, control paths which allow Keys to be loaded and selected, control paths which couple the Red and Black processors with low-bandwidth unencrypted paths, and a KG unit. The actual encryption algorithm chosen for the KG unit is the algorithm now adopted as a federal standard. The BCR data and control flow paths are shown in Figure 3-4.

Figure 3-5 is useful in describing the general approach of the experiment. There are four different site configurations illustrated from top to bottom. At the top is shown a typical secure Host. This Host supports general processes. The processes interface to the network through a user level protocol such as TELNET, and the TCP (see the previous section) Host-Host protocol. The Host itself is interfaced to the packet network via the BCR. The second configuration from the top of the figure is very similar to the general Host configuration except that the Host, instead of supporting general processes, supports a number

Figure 3-4: BCR Data and Control Paths

Figure 3-5: Possible Site Configurations of local terminals. Two Hosts configured as in the top of the figure, or two terminals concentrators configured as in the second line of the figure, or a terminal concentrator and a general Host, can communicate in a secure manner if their respective KG units are using matching keys. Furthermore, a given general Host or terminal concentrator can simultaneously communicate securely with more than one other such site, since the BCR units are able to select a different key for each message by a method described below.

The third configuration illustrates a situation where a terminal is not so near other terminals that it is cost effective to install a terminal concentrator Host; this situation is indicated by the terminal connection to the packet network being via modems. In this case, the BCR unit and the TCP and TELNET functions must all be effectively at the terminal. This currently requires three LSI-11s; the TCP and TELNET are incorporated into one and the others serve as B and R processors.

At the bottom of the figure is illustrated a special Host called a Key Distribution Center (KDC). This Host takes part in setting up the correct keys in the various BCR units. To be able to properly set up the keys, the KDC must be able to simultaneously communicate with the red side of the BCR units and the black side of the BCR units. Therefore, the KDC must simultaneously be connected to the network both with a BCR (for communication with R's) and without a BCR (for communication with B's). This dual connection has the further implication that the KDC itself must be certified to not leak secure data. If a communicating Host is sufficiently cooperative, the KDC can provide very specific security control, down to the level of security compartment, need to know, etc. The KDC for the current experiments is implemented on a TENEX system.

A scenario of the operation of the system is as follows. When a data packet is presented to the red minicomputer for transmission over the network, it must include (or imply) its source, destination, and length. Furthermore, if the packet is the first such packet, information for determining the classification of the data must be included. If the red minicomputer discovers that there is no entry in its tables relating the specified virtual source and destination to a link (described below) it either discards the packet (if no classification information is supplied) or sends a message to a KDC, using a preset link through the KG unit.

In response, the KDC sets up a duplex communications channel from end to end through the network. This is done by instructing the black minicomputer to set up the key storage for an unused link in the KG units at each end of the secure channel and specifying the actual source and destination addresses to be associated with that link in the black minicomputers. It also informs the red minicomputers of the link to use for that virtual connection. Now packets can be sent over the link until either the red minicomputer (from information supplied in the packets) or the KDC closes the link.

Figure 3-6 shows the relationship of the addressing tables loaded into the minicomputers and the keys loaded into the KG units to the link number. The common piece of information is the link. The link can be used to select the encryption key, the decryption key (they need not be the same), the virtual address information, and the actual address information. The address tables in the minicomputers permit the inverse translation to be done as well. One or more links are predefined to permit messages to be sent to the KDC.

The operation of the red and black minicomputers is nearly symmetric for outgoing and incoming messages. When the red

Figure 3-61 Relationship Between Keys, Addresses, and Links (black) minicomputer receives a packet from the Host (network) it strips off the virtual (actual) address information and looks up the link to use for that connection. It supplies that link number to the KG unit and then passes the remainder of the packet into the KG for encryption (decryption). The KG passes the link number on to the black (red) minicomputer and then the encrypted (decrypted) data. The black (red) minicomputer uses the link supplied to insert the actual (virtual) address information into the packet and sends the packet to the network (Host).

The differences in operation of the red and black minicomputers are in the handling of packets for which no corresponding link can be found. The red minicomputer attempts to establish the link as described earlier but the black minicomputer discards the packet.

The modification to the Host-Host protocols necessary to use this secured network are minimal. Principally, the source TCP must supply classification information to the red minicomputer to permit it to establish the necessary link. This information is encrypted and passed along to the destination where it may be used as an additional check on the validity of the connection. This information may be supplied at any time but must be supplied in the initial packet. In addition, the red minicomputer at one end of the connection (at least) must be informed by its Host computer when the link is no longer needed. This information is not the same as the information which the two Host computers exchange to close the connection, since packets containing that information may need to be retransmitted.

3.7 ARPANET Operation and Maintenance

The operation and maintenance of the ARPANET itself makes extensive use of the communication subnet and a number of network Hosts. The highly centralized approaches taken to these functions are quite novel, and a number of sophisticated, even if ad hoc, techniques have been developed during the course of operational experience. However, unlike most of the other significant aspects of network construction, experimentation, and utilization, the network operating mechanisms and techniques were not designed from scratch, but rather grew in response to various internal and external pressures. Thus a discussion of network operation and maintenance would be incomplete without mention of some of these historical factors.

3.7.1 Control Functions in the IMP

The recognition of the necessity for network control functions began with the design of the IMP hardware and software in 1969 [McKenzie72] [McKenzie75]. In the hardware design it was recognized that with several parties responsible for various pieces of equipment in the network, the problem of fault isolation was critical. Accordingly, both the Host interfaces and the modem interfaces included the capability for "loopback" test mode under program control. The loopback mode of operation basically connects the IMP's outputs to the IMP's inputs so that the IMP may generate test traffic, send it through the interface, and check the returning traffic against the traffic which was generated. Thus, for example, if a carrier-supplied communications circuit connecting two IMPs appears to be giving trouble, the IMPs can be directed to loop both of the modem interfaces connected to this circuit. If test traffic passes through these looped interfaces successfully, then it is reasonably certain that the carrier's equipment is at fault. On the other hand, if one of the two looped interfaces continues to show a pattern of trouble, it is reasonably certain that the IMP hardware or software is at fault. In either case, the appropriate repair and maintenence organization can be contacted, with little risk that the trouble will eventually be traced to equipment which is the responsibility of another party.

The IMP also includes hardware for the automatic detection of power failures and for automatically restarting the program when power is eventually returned. The IMP hardware (in the initial 516 systems) also contained a "watchdog timer" which counted down a clock at a rather slow rate and, if the clock was ever counted to zero, generated a very high priority interrupt.

The IMP software periodically reset this clock to its maximum value when the program was operating correctly. The interrupt routine triggered by the watchdog timer automatically reloaded (from a neighbor) and restarted the IMP program. In this way it was possible to recover from many erroneous states, whether caused by hardware failure or software bugs, without manual intervention.

The original IMP software, in addition to the watchdog timer routine and the software necessary to loop and unloop interfaces and to send test traffic and check the results, contained several other diagnostic features. These fall under two main headings, IMP statistics and DDT. The IMP statistics package contains a variety of routines, most of which are usable by any Host; these include the ability to trace the progress of a packet through the network, the accumulation of message and packet length histograms, the ability to take a "snapshot" of the internal state of an IMP at a given instant, and the ability to generate artificial traffic. One statistics routine, however, called the "trouble reports" routine, has its use restricted to network control and monitoring functions and, unlike other statistics routines, cannot be activated and deactivated by any Host, but remains permanently enabled and reports only to a single location. DDT is a small debugging package which contains a simple command interpreter, capable of such functions as examining and modifying a memory word, clearing a block of memory, searching memory for a particular stored value, etc. DDT is structured so that it can be driven remotely through the network, returning any responses back through the network. Controls (which were not present in the initial implementation) insure that DDT can be operated only from appropriate locations.

3.7.2 Early Operation

During the first several months of the ARPANET, when all the installed IMPs were located in California and Utah, operation of the network relied heavily on the assistance of personnel employed at these sites, as well as on the more or less continual on-site presence of the development staff members from BBN. Recovery from an IMP failure generally involved local use of DDT through the IMP console Teletype by site personnel, working in conjunction with a BBN programmer (either on-site or by telephone). Each IMP was equipped with a high-speed paper tape reader and each site kept a paper tape of the most recent IMP software version. In case recovery was not possible using DDT, site personnel reloaded the IMP through the tape reader. If after several attempted loads the IMP continued to fail, the problem was presumed to be a hardware problem and Honeywell Field Service, which had maintenance contracts on all the machines, was called in to perform appropriate diagnosis and repair.

Each IMP monitored the status of the common carrier circuits connected to it and displayed their status in a set of lights on the IMP front panel. Site personnel would periodically examine these lights and, if the circuit was seen to be in a down condition, would use DDT via the IMP console Teletype to loop the interfaces, generate test traffic, and monitor the results, which again were displayed as an up or down condition in the front panel lights. Site personnel would then contact BBN and, depending on the results of the testing, either Honeywell or the common carrier would be notified of an equipment failure.

However, with the installation of the network's fifth IMP at Bolt Beranek and Newman in early 1970, testing and diagnosis became considerably more centralized. The "trouble reports" statistics program in each IMP was set to send periodic reports to the console Teletype connected to the BBN IMP. Data on IMP status and circuit status were sent as ASCII text to be typed out on the Teletype. Since there is limited buffering within the

IMP, and the output was produced on a low-bandwidth device, space within the message was at a premium. However, a person was required to read and interpret the data; the message format thus had to be at least somewhat intelligible. Balancing these two factors resulted in concentrating on only that data which was vital to the continued functioning of the network: circuits usable or unusable, the state of the console switches at each IMP, or (by inference from the absence of any report at all, for a long period of time) indication that an IMP had stopped functioning completely.

Members of the IMP hardware and software development team were responsible for scanning this typescript occasionally and initiating appropriate actions when necessary. Since no individual was exclusively assigned to monitor the Teletype, and since personnel were normally available only during regular working hours, it was possible to have a long period of outage for a circuit or an IMP without anyone noticing it or beginning corrective action. However, since the network was very small, and since its operation was not critical to almost any Host activity, this did not constitute a severe problem.

As the network became larger and more reliable, the proportion of status messages typed on the Teletype which said anything other than "everything still OK here" decreased, thus making the location of messages which required action more difficult. In order to make location of these critical messages easier, the format of the trouble report routine was changed so that it used the "synchronized clock" of the statistics package. Each IMP was programmed to send a status message every 15 minutes; each IMP also examined its own status every minute and sent an additional message if it detected a status change. Thus every 15 minutes a block of "check-in" reports was typed, one from each IMP. Interspersed between these blocks on the typescript there would be an occasional status change report. In general, only the change reports required action.

3.7.3 The Network Control Center

As the network continued to expand in size, scanning the typescript on the IMP Teletype for network events which required human intervention became increasingly difficult. In addition, there was considerable interest in periodic reports on IMP and line performance, statistics on circuit usage in order to obtain advance warning of approaching saturation, and Host traffic in order to investigate accounting algorithms for network usage. These factors led to the attachment of a small Host to the BBN IMP dedicated to monitoring network performance and doing much of the bookkeeping required for the reports, as well as watching for events which might require human intervention. The computer chosen for this Host, known as the Network Control Center (NCC) Host, was identical to a network IMP, namely a Honeywell 316 computer with Host and modem interfaces, 12K of 16-bit memory, and a real-time clock. Choosing this particular set of hardware meant that, in the event of a failure in the NCC Host, the prototype IMP normally used for software development could be temporarily substituted. A set of peripheral interfaces, which were packaged separately from the NCC Host itself, were provided to drive output Teletypes, a set of display lights, and operator switches; this equipment could easily be connected to the I/O bus of an alternate CPU if the need arose.

By mid-1971 the NCC Host had become operational and it became possible to make several changes to the format of the "trouble reports" generated by each IMP. First, the ASCII text format was abandoned in favor of binary encoding. Next, the reports were expanded to include more internal status information, as well as statistics on Host traffic and line usage. It was also possible to increase the frequency of reporting to once a minute for the "check-in" and to send change notices as soon as changes were detected. The NCC Host computer was given the job of examining each of these reports, noticing changes in network status, and alerting the operations staff which by this time had grown to dedicated operator coverage for about 12 hours a day, five days a week. Operators were informed by the NCC Host that network status had changed by printouts on one of the Teletypes connected to the Host and, in the case of IMP or circuit failure, by flashing a light (a separate light for each IMP or circuit) and sounding an audible alarm. The NCC Host also accumulated the line usage and Host traffic statistics and summarized these on a second Teletype.

It is worth noting that the term "Network Control Center" is a misnomer; "Network Coordination Center" or "Network Monitoring Center" would be a more appropriate name. In fact, a very important result of the ARPANET project is the demonstration that it is possible for a network to control and operate itself without explicit control from a control center. Many network designs include at their heart the assumption that there must be a manned control center with the capability of adjusting the network's routing, instructing nodes which lines to use, adjusting nodal buffer storage, adjusting nodal priorities, and so forth. The ARPANET depends on no such central control. All such adaptations are made by the nodes themselves, alone or in concert; further, these decisions are generally made more rapidly and accurately than could be done from a manned control center. The ARPANET NCC is responsible for coordinating maintenance and growth, but is not involved in real-time control.

The functions of the Network Control Center Host have expanded rather dramatically, although in small steps, from the time of its initial implementation. First, the hardware was expanded from that of the standard IMP to that of the standard Terminal IMP, providing additional core memory as well as the possibility of operating several terminal output devices. In late 1974 the Teletype which was used to print the log of network events was replaced by a 1200-baud line printer. This was necessitated not only by the greatly increased number of network nodes, but also by the large amount of additional information which the IMPs were reporting to the NCC Host. Reported events now include the status and error frequency of communications circuits in the network, the up/down transitions of each Host in the network, the lengths of various buffer queues in each IMP, the settings of the various console switches on the IMPs, and a variety of "trap" conditions which may occur.

The trap mechanism allows the programmers or maintainers to have the IMP notify the Network Control Center whenever a particular block of code is executed. A single trap subroutine reports the program counter, the contents of the accumulator, and the contents of the IMP's index register for each trap condition. This mechanism is useful for determining the frequency of execution of various blocks of code within the IMP, and the set of traps changes as the attention of the programmers moves from one area of the IMP algorithm to another. The trap mechanism is also used to report failures of the software checksums and other difficulties related to hardware failures.

As the amount of information presented to the NCC operators on the log printer increased, and because an increasing amount of that information did not call for direct operator action at all, but was destined instead for system programmers and the maintenance staff, it became desirable to find a way to separate the log into several pieces. One possible solution would have been to connect a multiplicity of output devices to the NCC Host, each output device maintaining a separate log. However, the separation of the log output into several pieces need not be done in real time. Thus a single log is maintained, with the lights display and audible alarm as the primary method of notifying the operators that immediate action may be required. Programs have been written for network TENEX systems which periodically probe the NCC Host and request copies of all available log information, which is then placed on TENEX bulk storage. Further, the data from the TENEXes is sent to the Datacomputer roughly once per day for long-term storage. Programs have been written to examine this data and separate it into as many categories as desired. The list of occurrences in each category can then be distributed to the appropriate NCC staff member and examined for significance; a record of the global context for each such event can be obtained from the single printed log.

3.7.4 IMP Software Maintenance

The IMP software included the capability of obtaining a reload of its core memory, except for a small region containing constants specific to each machine and the loader itself, from a neighbor IMP under command from DDT or from console switches; the watchdog timer also used this mechanism. It seemed reasonable to use this mechanism for all new software releases once an IMP was installed at BBN. The procedure of releasing new software could then be to mail a tape of only the constants and loader area of the IMP to each site and at a pre-scheduled time have each site load this tape into the IMP. The entire new version of the IMP program would be read into the BBN IMP through the high-speed paper tape reader. The sites adjacent to BBN would then direct their IMPs to reload from the BBN IMP. Once this was completed, sites adjacent to the reloaded sites would, in turn, reload from these sites until the new software was propagated through the entire network.

Using this software release procedure, site personnel were involved in the release only at those times when the constants and loader area, or the IMP-to-IMP communications protocols, were changed; in many cases new software could be released using only the BBN paper-tape reader and DDT. Thus the software development staff began to place emphasis on keeping the software "compatible" across IMP releases; that is, insuring that the protocol which the IMPs use to communicate with each other remain unchanged.

Finally in mid-1972, when the network had grown to about 30 nodes, it became necessary to release an incompatible version of the IMP software. It proved impossible to coordinate a release based on the paper tape procedure, and therefore it was necessary to devise a new plan which would not require any site intervention, in spite of the incompatible nature of the new software. The plan which evolved required several steps: first, DDT was used to overwrite the loader area of each machine with a new loader which could communicate either with the old version of the software or the intended new version (a bit in the IMP-to-IMP header was usurped to differentiate the systems). Once this step had been completed at every IMP, one of the circuits was disconnected from the BBN IMP and connected instead to a development machine, as illustrated in Figure 3-7. This second "BBN IMP" was running the new IMP software, but this was not confusing to the machines running the old system; the software incompatibility meant they could communicate with only one "BBN IMP" rather than two. A message was then sent from the console Teletype of the "old" BBN IMP to the DDT of site "A" (refer to Figure 3-7) instructing site A to reload itself from the "new" BBN IMP. When this reloading process was completed, there was a "network" of two machines running the new system, and a network of the remaining machines running the old system. This procedure was repeated for each machine until the entire network except for the "old" BBN IMP was running the new system. This IMP was then disconnected from the network and its circuit(s) connected instead to the "new" BBN IMP.

Figure 3-7: Incompatible Software Release - Topology Example

An additional complication remained to be faced, the case where the topology did not allow two paths from BBN to an IMP, one through the old system and one through the new system. Again referring to Figure 3-7, the "arm" containing two machines (C and D) has only one connection to the rest of the network (through

8). The procedure for this case was as follows. First, the NCC insures that the new software is operating at IMP E. Next, using DDT, a software "patch" is installed in IMPs B, C, and D such that each patched machine will ignore reload requests from its neighbors. Then IMP D is instructed to reload itself from IMP C; similarly, IMP C is instructed to reload itself from IMP B. Finally, IMP B is instructed to reload itself from IMP E. Since IMP E is not patched to ignore reload requests, it immediately transmits a copy of the new software to IMP B. This reload overwrites the patch which told IMP B to ignore reload requests, so IMP B will then honor the next of IMP C's (repeated) reload requests. Similarly, this reload overwrites the patch in IMP C, and thus the new software is propagated to IMP D.

Of course, with the advent of the TIP, even supposing all the TIP programs remained identical, it was not possible to guarantee that a Terminal IMP would have any neighbor which was a Terminal IMP (of course, the "IMP" portion of a TIP remained identical to other IMPs). Thus, in the event of any software changes or failures in a Terminal IMP, a mechanism to reload the Terminal IMP software, without reliance on a neighbor, was required.

The first approach to the problem of reloading TIPs was to load the TIP software, which might be specialized for each individual TIP, into the "development TIP" (which was used for stand-alone software checkout of TIP software), connect this TIP to the network, and install in it a small routine which could interact with the DDT at the target TIP site. This routine read a word of TIP code and sent this word to the DDT at the remote site with instructions to load it into memory at the same location from which it had been taken. The process was repeated until the entire TIP program had been loaded (through the network) into the remote TIP.

Although this process worked, it was cumbersome and required a great deal of operator activity with the resultant possibility of error, as well as interrupting software development work which might be taking place. Thus, as the number of TIPs in the network increased, pressure mounted for some more automatic method of reloading. It seemed obvious that a reliable Host computer, equipped with a bulk storage unit, could store a copy of the program for each TIP. Even if the TIPs became very divergent, it would not require a great deal of bulk storage to store a separate copy for each TIP. By late 1972 facilities which supported this activity had been implemented on a PDP-1 computer at BBN which was connected to the ARPANET as a Host. Eventually these support facilities, as well as others which had been implemented on the PDP-1, were moved to TENEX systems because of their greater power and the reliability inherent in their redundancy. These facilities followed the earlier procedure, as described above, of using DDT to load a copy of the TIP program, or a patch to the IMP or TIP software, one word at a time in what amounted to a simulation of an operator's typing. However, this method could not be extended to the loading of IMP software either for failed machines or for new releases, since use of DDT through the network requires the entire set of IMP software to be running.

As time passed, and the network became increasingly large and varied, the mechanisms of complete memory reload from a neighbor and of selective reload via "automated" DDT became increasingly awkward and inadequate. Several examples of needs not well satisfied by those mechanisms are:

- The NCC wanted to be able to propagate new releases by sending simple commands to each target IMP telling it to accept core transfers from some other designated IMP.

- To examine an IMP that crashed and get it back on line quickly the NCC needs to be able to ship the core image of the failed IMP back to the NCC to be examined later by a programmer.

- A general reloading scheme was desirable to load "add-on" pieces such as the TIP, VDH, or Satellite IMP code, which are not part of the basic IMP system, and Hosts for which the NCC has full responsibility such as the PLI.

- With the addition of Pluribus IMPs to the Network, the existing method of loading an IMP from its neighbor was insufficient. Once a Pluribus IMP has been installed, most network configurations require a general way to load a remote IMP across the network through dissimilar machines.

- Loading or dumping a complete core image of a Satellite IMP using a broadcast satellite link is not convenient since it prevents use of the link by other nodes for several seconds.

- A logical adjunct to sending core images through dissimilar machines is sending core images through (perhaps identical) machines which may be running different software. Thus, a new compatible software release could be accomplished in a more modular fashion by loading individual IMPs at the most appropriate time for them, without being constrained to first load at least one neighbor IMP.

The mechanism constructed for accomplishing all of the above is a packet core transfer mechanism which consists of two endpoints -- the font (source of the core image) and the sink (destination of the core image) -- and a protocol governing the reliable transmission of the packetized core image. The font puts pieces of the core image into packets and sends them over the network; the sink receives packets from the network and builds the appropriate core image.

Either font or sink can take one of three basic forms. The first is that of a real network Host, such as a TENEX used by the NCC. Thus the TENEX could be the font for sending "add-on" (i.e. TIP, VDH, etc.) pieces to running IMPs, or the sink for receiving core images of running IMPs for periodic (preventive maintenance) verification of correct software. The second form is that of the "block" fake Host in a running IMP. This fake

Host is the font for any transmission of core from the running IMP (i.e. for verification at the NCC or for loading another dead IMP of a similar machine type), and the sink for any pieces for the core image loaded into the live IMP (i.e. "add-on" pieces). The third form is a stand-alone program running in a non-functioning IMP, in conjunction with a live neighbor IMP's block fake Host, which acts as the port into the network for the failed IMP, and is explained more fully below. The block fake Host is able to distinguish between packets addressed to itself (form two) which require it to load or dump its own IMP's core, and packets addressed to a failed neighbor IMP (form three) which it reformats and sends over a direct circuit from itself to that neighbor.

To bring up a non-functioning IMP, the required stand-alone program (that may have to be loaded on-site) should be as short and simple as possible. Therefore, it should know nothing about network procedures such as acknowledging, routing, RFNMs, etc., and talks over its modems in "blocks", a data format which is different from any other type of packet, and which carries much less header information. The live neighbor IMP at the other end of the modem also understands blocks. The block fake Host converts packets arriving in standard forms for its non-functioning neighbor into blocks and sends them over the modem to that neighbor. It also converts blocks arriving from its non-functioning neighbor into packets which are then transmitted over the network. When operating in this third form, the block fake Host is completely unaware of the nature and extent of the core transfer; thus, the internal structure of the failed IMP and its neighbor can be completely dissimilar.

The protocol calls for an initiation handshake, followed by a one-way flow of pieces of core image, with possible resynchronization messages in the reverse direction, followed by a termination handshake. The initial request for a core transfer may originate at the NCC, or in a healthy IMP which has a dead neighbor, or from a dead or dying IMP, and goes to the font. The font is now open and ready to send core, and sends a request towards the sink, which then becomes open and ready to receive core. The font follows the initial request with sequence-numbered pieces of the core image, and then a "last message" message. If piece-numbers ever get out of sequence, or a piece is excessively tardy, the sink sends back a piece-number reset request, and waits for the correct piece to arrive. When the sink gets the "last message" message, it echos it back to the font, causing the procedure to be terminated.

3.7.5 Hardware Maintenance

By mid-1972 the Network Control Center staff had been expanded to provide operator coverage 24 hours a day, seven days a week. At about this time, first order responsibility for network operations was assigned to the operators, with back-up provided by the hardware and software development staff members who were instantly available at the NCC during the working day and available at home after hours. The operating staff also assumed responsibility for a number of peripheral issues such as the scheduling of installation of new sites and new circuits, the coordination of routine maintenance activities so as not to disrupt network connectivity, and so forth. Field maintenance personnel from both the common carriers and the Honeywell repair offices (who had primary responsibility for the repair of IMP hardware problems) were instructed to accept maintenance calls only from the Network Control Center, rather than from the individual sites, since only the NCC had a global picture of the network and was able to see how problems in one area might be related to problems in another area.

As a sidelight, even prior to this time the carrier personnel had been instructed to accept trouble calls only from

BBN. As the network grew into new areas, far removed from Cambridge, this frequently led to difficult interactions with carrier personnel, who could not easily believe that a group in Cambridge could be accurately reporting the state of a circuit running between, for example, Urbana, Illinois and Salt Lake City, Utah. Fortunately, all circuits were ordered by the government through a central office in Washington, and this office was able to observe the NCC track record in reporting failures in other areas of the country. When disputes arose involving a new circuit, a call from the central facility was usually sufficient to convince the repair office that the Network Control Center's repair request was believable. It is important to note that this believability was the direct result of the great care that had been taken to build adequate diagnostic tools into the IMPs, so that the carrier personnel were not called because of problems which were actually caused by the IMP hardware or software.

In early 1974, a hardware engineer with a considerable amount of programming experience was added to the Network Control Center staff and given responsibility for the maintenance of operational IMPs. The maintenance engineer initiated a quality assurance program which is carried out on each new IMP and each new interface before it is permitted to be shipped from BBN to the field. The testing procedure is sufficiently routine so that the majority of it can be carried out by the NCC operators during times when they are not occupied with more urgent tasks.

However, a key result of assigning a maintenance engineer to the NCC was the discovery that, with the aid of powerful Host computers, and a rapid communications system, and the use of IMP tools such as DDT which allowed routines to be added to a running IMP to test aspects of its hardware performance, it was possible to maintain IMP hardware by debugging, rather than by trouble shooting. Further, maintenance and debugging is actually of a higher quality, because it is controlled by experts who can be concentrated at the center rather than by the less highly trained personnel normally available on a field maintenance staff. Of course, the field maintenance staff is still necessary when physical presence is needed at a field site; e.g., when the central personnel wish a hardware component changed.

One example of the changed approach to maintenance is the study of memory errors. The trouble-shooting approach to this problem is to swap memory driver cards from one memory stack to another to see which, if any, causes the problem to move. This typically requires a lot of on-site time and many service interruptions. The debugging approach uses the packet core transfer mechanism plus a verification program running on TENEX which retrieves a copy of the core memory of a running IMP and compares this, word by word, to an image of correct memory contents on the TENEX mass storage. Using this method it is possible to identify any pattern of dropped or picked bits and from this pattern the particular driver in error. Another example of a diagnostic procedure is the use of DDT to accurately measure an IMP's real-time clock. Failure of the real-time clock can lead to several problems with an IMP, particularly to too frequent or too infrequent probes of the circuits to which the IMP is connected, which is likely to result in one of the IMPs connected to the circuit declaring it unusable. The test of the clock can identify it as the cause of this trouble much more easily than can trouble-shooting in the field.

Use of these centralized techniques leads to a different economic balance between the central staff and the field staff. Thus, by 1975 it became desirable to discontinue Honeywell hardware maintenance responsibility and add this task to the NCC in order to "trade" field staffing support for central staffing support. The result of these changes in maintenance philosophy was a reduction in the average node downtime due to hardware and software failures from about 1.5% to about 0.25%, a reduction of about a factor of six.

REFERENCES

Abramson70 - N. Abramson, "The ALOHA System - Another Alternative for Computer Communication", AFIPS 1970 National Computer Conference Proceedings, Vol. 37, November 1970, pp. 281-285.

AFCS72 - Air Force Communications Service, "AFCS Test of the ARPA Network", Richards-Gebaur AFB, Missouri, January 1972.

Bouknight73 - W.J. Bouknight, G.R. Grossman, and D.M. Grothe, "The ARPA Network Terminal System - A New Approach to Network Access", Proceedings of the Third Data Communications Symposium, November 1973, pp. 73-79.

Braden73 - R.T. Braden, "Update on NETRJS", Network Working Group Request for Comments #599, December 1973.

Butterfield74 - S.C. Butterfield, R.D. Rettberg, and D.C. Walden, "The Satellite IMP for the ARPA Network," Proceedings of the Seventh Annual Hawaii International Conference on System Sciences, Honolulu, January 1974, pp. 70-73.

Carr70 - S. Carr, S. Crocker, and V. Cerf, "Host-Host Communication Protocol in the ARPA Network", AFIPS 1970 National Computer Conference Proceedings, Vol. 36, May 1970, pp. 589-597.

Cerf74 - V. Cerf, and R. Kahn, "A Protocol for Packet Network Interconnection," IEEE Transactions on Communications, Vol. COM-22, No. 5, May 1974.

Cerf76 - V. Cerf, A. McKenzie, R. Scantlebury, and H. Zimmermann, "Proposal for an International End to End Protocol," ACM Computer Communication Review, Vol. 6, No. 1, January 1976, pp. 63-89.

Cosell75 - B. Cosell, P.R. Johnson, J.H. Malman, R.E. Schantz, J. Sussman, R.H. Thomas, and D.C. Walden, "An Operational System for Computer Resource Sharing", Proceedings of the Fifth Symposium on Operating System Principles, Austin, Texas, November 1975, pp. 75-81.

Dorin77 - R.H. Dorin and D.E. Eastlake, "Use of the Datacomputer in the Vela Seismological Network", Proceedings of the IEEE International Symposium on Computer Aided Seismic Analysis and Discrimination, Falmouth, Massachusetts, June 1977, pp. 35-39.

Englebart73 - D.C. Englebart, R.W. Watson, J.C. Norton, "The Augmented Knowledge Workshop", AFIPS 1973 National Computer Conference Proceedings, Vol. 42, June 1973, pp. 9-21.

Falk76 - H. Falk, "Reaching for a Gigaflop", IEEE Spectrum, Vol. 13, No. 10, October 1976, pp. 65-69.

Forgie75 - J.W. Forgie, "Speech Transmission in Packet-switched Store-and-Forward Networks", AFIPS 1975 National Computer Conference Proceedings, Vol. 44, pp. 137-142.

Forsdick77 - H.C. Forsdick, R.E. Schantz, R.H. Thomas, "Operational Systems for Computer Networks", BBN Report No. 3614, July 1977.

Henderson77 - D.A. Henderson and T.H. Myer, "Issues in Message Technology", Proceedings of the Fifth Data Communications Symposium, Snowbird, Utah, September, 1977.

Kahn75 - R.E. Kahn, "The Organization of Computer Resources into a Packet Radio Network", AFIPS 1975 National Computer Conference Proceedings, Vol. 44, pp. 177-186.

Kehl73 - W. Kehl, "The UCLA Campus Computing Network: An ARPANET Resource", Educom Bulletin, Vol. 8, No. 4, Winter 1973, pp. 10-17.

Marill75 - T. Marill and D. Stern, "The Datacomputer - A Network Data Utility", AFIPS 1975 National Computer Conference Proceedings, Vol. 44, pp. 389-395.

McKenzie72 - A.A. McKenzie, B.P. Cosell, J.M. McQuillan and M.J. Thrope, "The Network Control Center for the ARPA Network", Proceedings of the 1972 International Conference on Computer Communication, Washington, D.C., May 1972, pp. 189-191.

McKenzie75 - A.A. McKenzie, "The ARPA Network Control Center", Proceedings of the Fourth Data Communications Symposium, October 1975.

Metcalfe71 - R. Metcalfe, "Some Historic Moments in Networking", Network Working Group Request for Comments #89, January 1971.

Mobley77 - R.L. Mobley, "Comments on the ARPANET", personal correspondence with Bolt Beranek and Newman, August 1977.

Ornstein72 - S.M. Ornstein, F.E. Heart, W.R. Crowther, H. K. Rising, S.B. Russell, and A. Michel, "The Terminal IMP for the ARPA Computer Network", AFIPS 1972 National Computer Conference Proceedings, Vol. 40, pp. 243-254.

Panko77 - R.R. Panko, "The Outlook for Computer Mail", Telecommunications Policy, June 1977, pp. 242-253.

Retz75 - D.L. Retz, "ELF - A System for Network Access", Proceedings of the IEEE Intercon Conference, New York City, April 1975.

Retz76 - D.L. Retz and B. W. Schafer, "Structure of the ELF Operating System", AFIPS 1976 National Computer Conference Proceedings, Vol. 45, pp. 1007-1015.

Robinson76 - R.A. Robinson, "The National Software Works: Its purpose, Status and Future Plans", Rome Air Development Center, briefing to ARPANET Sponsors' Group Meeting, November 1976.

Thomas73 - R.H. Thomas, "A Resource Sharing Executive for the ARPANET, AFIPS 1973 National Computer Conference, Vol. 42, pp. 155-163.

4. OBSERVATIONS

For many of the people in government, at the major contractors, and in the participating universities and research centers the development of the ARPANET has been an exciting time which will rank as a high point in their professional careers. In 1969 the ARPANET project represented a high risk, potentially high impact research effort. The existence of the net in practical useful form has not only provided communications technology to meet many short term needs, but it represents a formidable communications technology and experience base on which the Defense Department as well as the entire public and private sectors will depend upon for advanced communications needs. The strong and diverse experience base generated by the ARPANET project has placed this country ahead of all others in advanced digital communications science and technology.

4.1 Social Issues

Somewhat expectedly, the network has facilitated a social change in the United States computer research community. It has become more convenient for geographically separated groups to perform collaborative research and development. The ability to easily send files of text between geographically remote groups, the ability to communicate via messages quickly and easily, and the ability to collaboratively use powerful editing and document production facilities has changed significantly the "feel" of collaborative research with remote groups. Just as other major improvements in human communication in the past have resulted in a change in the rate of progress, this social effect of the ARPANET may finally be the largest single impact of the ARPANET development.

A non-trivial question of considerable importance to the country is why has the ARPANET project been so successful. It would certainly be nice if the same formula could be applied to other pressing national needs. While timing, accident, and luck must not be discounted, it is possible to identify several possible contributing factors:

o Within the Defense Department, and even within the government as a whole, ARPA has had an unusually high degree of freedom in identifying the right people and then funding those people in a relatively rapid and efficient manner. Further, even within ARPA, Larry Roberts was an unusually gifted individual with a special blend of personal technical competence, political persuasiveness, and drive. Although there are always some constraints, Roberts was able to manage the research and development at the many organizations involved with the network in a very unfettered fashion. It was clearly benevolent dictatorship rather than committee management, although Roberts "listened well" before deciding on a course of action. Strong management also existed within the organization of the prime contractor, Bolt Beranek and Newman Inc., where project control was vested in a single strong individual. It is probably the case that only ARPA could have built the ARPANET with the same degree of success.

o Despite the fact that the ARPANET was a government project and further despite the fact that it was run within the Defense Department, it was possible, at least initially, to free the research and development from some of the constraints which often seriously hamper other activities. In particular, the project was entirely unclassified and even the interactions with some highly classified agencies managed to stay on an unclassified basis. Second, the provision of network service was for a very long time provided as a "free good"; this allowed people to experiment with the use of the network without being forced to make early and probably ill advised cost/benefit decisions. Third, despite a deeply ingrained government and Defense Department worry about unauthorized use of government facilities, it was possible to build the ARPANET without complex administrative control over access or complex login procedures or complex accounting of exactly who was using the net for what. Finally, the ARPANET did not have to interconnect with other existing and/or decrepit communication systems; it was possible to establish line protocols and interface standards *de novo*, in the best ways that could be devised. Thus, the ARPANET program was incredibly free of "artificial" requirements and was able to concentrate intensively on the primary required research and development. Much later in the project, after success had been assured, it was relatively easy to reopen consideration of some of these issues; and at the current time, for example, the Defense Communications Agency is charging user groups for network access, and there have been experiments (using private line interfaces) in the transmission of classified data over the ARPANET, and research was done on how to implement network login procedures, etc., etc. The key was to avoid such extra confusions at the outset of the project.

- o A very convenient fact was the common ARPA support of both the "network authority" and the initial early group of network users. It was possible for ARPA to strongly encourage a cooperative attitude and cooperative engineering at the time in the project when such cooperation was most critically necessary.

In sum, the project was an illustration of what can be accomplished with strong technically sophisticated central management, adequate resources, and a clear headed undeviating concentration on the central research and development issues.

The largest single surprise of the ARPANET program has been the incredible popularity and success of network mail. There is little doubt but what the techniques of network mail developed in connection with the ARPANET program are going to sweep the country and drastically change the techniques used for intercommunication in the public and private sectors. By hindsight, one can easily see the reasons for this success. The primary prior existing communications techniques (the U.S. postal service and the telephone) have extremely serious deficiencies! The postal service has become more expensive and its performance has dropped; one must expect delays measured in days for letters, and some times the delay is weeks. In the case of the telephone, our increasingly mobile society makes it difficult to reach the desired person, and reaching 10 distributed people within a short period of time is essentially impossible. Leaving telephone messages works if a careful system is established and a travelling individual checks back with his answering service or secretary frequently, but it is often inconvenient and is usually limited to the prime shift working day of the secretarial world. To find a busy person able to accept a phone call at the time you make it is truely unusual, and some officials have desks covered with little pink slips reporting on telephone messages with requests for return calls. Into this milieu was dropped a technique of network mail where at any time of the day or night one can send a message to any number of other people and expect that message to semi-instantaneously be available in the computer mailbox of all the recipients. Then one only has to assume the habit on the part of all individuals using the system to occasionally check their mailboxes when they are free and not at a meeting, and the performance of the communications is immeasurably improved over the postal service or the telephone. With the addition of sophisticated tools for answering messages, filing messages, forwarding messages, and categorizing messages, the system takes on yet another step function of performance over the alternate possibilities. In the space of just a couple of years this "computer center curiosity" became a smashing success on the ARPANET; there was a sufficiently large community of individuals who wished to communicate, and they all were relatively mobile, and the system overnight became a way of life. The implications of this kind of success are enormous. Perhaps such advances would have eventually come anyway as the U.S.

Postal Service cost-performance continues to drop, but there is no question that the ARPANET program provided a truely convincing demonstration of the power of this approach. The change will not be overnight, because it does depend upon the availability of terminals accessible to wider and wider populations, but commercial systems are already available and substantial DoD experiments are already under way.

Before leaving the class of rather broad observations, it is perhaps appropriate to note one disappointing political phenomenon that was visible in the ARPANET program. In particular, the ARPANET program provided another example of the difficulty of strong research cross-fertilization between various major branches of the federal government. As the success of the ARPANET became clear, many individuals realized that there were other communities of research in the United States who could substantially gain from some experience with the ARPANET. For example, the research communities supported by the National Institutes of Health and the research communities supported by the National Science Foundation could have gained substantially by a serious interaction with the ARPANET program. Based on this obvious fact, several attempts were made to create such cross ties (and, in fact, one important NIH-supported facility was tied to the ARPANET and did provide some benefit in this direction). However, it was extremely difficult to make progress down this path. A clear "not invented here" phenomenon existed in some quarters, a feeling that DoD programs should not be expanded to include other agencies existed in some parts of the DoD, and in some other agencies a feeling existed that "military" programs should not be mixed with "clean" scientific programs. Further, the uncertainty over the direction in which technology transfer might take place from ARPA did not make it easy for other agencies to commit funding for integration attempts without any clear expectation about the duration of the ARPANET life. A philosophical view of this difficulty might hold that only about a half decade delay in the use of the technology by other research communities was introduced, and why should that matter? However, at the time it seems hard to condone such difficulty of technical cross-fertilization. Perhaps there might be some way by which the federal government could encourage greater research cross-fertilization between agencies in selected areas.

4.2 Technical Lessons

Leaving the broad social plane, the ARPANET program provided several technical lessons which are worthy of general comment:

4.2.1 Terminal Handling

Rather early in the ARPANET program, it became clear that terminal access to the net via the main Host computers was an inadequate approach; many classes of users required direct terminal access to the net in order to use major Hosts at other locations. The first response to this pressure was the design of the Terminal Interface Message Processor (TIP) by the ARPANET prime contractor, Bolt Beranek and Newman Inc. The TIP was designed to address a limited problem; to handle character-oriented asynchronous terminals only and to be an integral part of the network authority and not available for user programming or special user features. This limited goal and hard-nosed attitude permitted the rather rapid completion of the TIP design, the fielding of many TIPs, and the rapid availability of wide spread terminal access to the network. Thus, the TIP effort was extremely successful in reaching its limited goal.

Unfortunately, but perhaps not surprisingly, the limited goal and absolute restriction on user programming created considerable unhappiness in portions of the potential user community, and created considerable pressure for other "better" terminal access techniques. Some of the complaining was fully justified; this approach to servicing interactive character terminals had "leapfrogged" a whole segment of the industry that was concerned with batch processing and the use of synchronous line disciplines from such batch processing units. Other complaints were less rational and more self serving, where some groups really wanted "a computer of their own" under the guise of obtaining terminal access to the ARPANET. Still other criticism was based on an honest difference of opinion as to the relative ease of designing and deploying a terminal access device that would permit user programming and would handle a broader class of terminals. Finally, some criticism was provided by highly sophisticated users who were used to the terminal support "services" provided by the most advanced large Hosts, and did not like the limited services provided by the tiny mini-Host resident in the TIP.

In response to this pressure, ARPA for a time supported the development of a device called the "ARPA Network Terminal System" (ANTS). This was a system based on a PDP-11 which was intended to provide user programming ability, the handling of synchronous terminals as well as asynchronous terminals, and a more powerful set of services to the terminal user. Unfortunately, the goals were somewhat ambitious and, although a few ANTS were put into the field, the configuration management of the program, the difficulty in debugging fielded versions of the system, and delays in implementation eventually led to a cessation of ARPA support for the effort. Since the pressures did not really diminish, a second similar attempt was made with a system called "ELF". Here ARPA support was provided to try to standardize, improve, and deploy a PDP-11 based terminal support software system which had been independently developed for a particular project. Again, the hope was to permit user programming and the handling of a wider class of terminals. From the viewpoint of deployment, the efforts to use ELF were much more successful; goals were far more limited, and more attention was paid to orderly development and maintenance. However, ELF has probably been more useful as a base for individualized mini-Hosts serving local specialized Host functions than as a means of widely distributed network access for large numbers of terminal users. At the time the ARPANET was transferred to DCA, primary terminal access was still either through the main network Hosts or via the many TIPs in the network. There are several morals to this history. First, it is extremely difficult to build a system which can handle all possible terminals; it is a bit like the "everything" machine and leads to an unlimited expansion of the problem. Second, very great attention must be paid to software configuration management, central program design and release, central maintenance and software support, and close control of program changes if an evolving computer-based device is to be deployed in considerable numbers around the network. There is still an open technical question whether such a device could be sensibly and cost-effectively fielded in a way which _would_ permit both some level of user programming to tailor the functions of the device and a standard set of controlled basic services; at this point in the development of the technology a conservative approach precludes such user programming if a device is to be widely deployed.

Another related aspect of the terminal handling problem has to do with the management extent of the network authority. It was discovered that when an unsophisticated user typed at a terminal in a remote location and something went wrong (that is, the proper response was not received), the user typically "blamed the network" despite the fact that any number of possible things: the terminal itself, the local modem, the local line, the modem at the other end of the local line, the terminal handling device (e.g., TIP), the network itself, the eventual Host computer, or any similar point on the return path, could have been at fault. Thus, it is extremely important that the network authority have adequate administrative control over the entire collection of equipment from the terminal right through to the Host computers if it is to be in a position to respond to such unstructured outcries of rage from the end terminal user. During the ARPANET program, numerous "crisis" incidents were precipitated by the failure of equipment which was not in any way under the control of the network authority. [Perhaps the best single example was difficulties in the local on-base telephone system at Ames where it took a massive effort to eventually uncover the offending equipment and where the network authority was forced in its own defense to participate and lead in the massive debugging activity even though the offending device eventually found was clearly outside the network authority's control.] The generalizable lesson from this kind of history is that devices that attach to networks must be extremely clearly in <u>somebody's</u> camp of responsibility. Either the device must clearly belong to the network authority and must have built-in techniques for debugging and failure location, or the device must clearly belong to the Host organization or some other well identified group who understands its responsibilities for maintenance and trouble location; devices cannot sit in cracks between different authorities. Although this idea is really quite simple, it is frequently overlooked.

A final point on the general terminal handling problem has to do with the location of the "intelligence" for dealing with terminal related matters. In general, the data processing power can either be in the terminal itself, in the terminal handling network port (e.g., the TIP), in the eventual final Host computer or, of course, distributed among these three locations in some way. As the cost of data processing is dropping, more and more intelligence is appearing in the terminal itself and this entire matter must be periodically reviewed to ensure that the system performance can take advantage of technological change.

4.2.2 Reliability and Fault Isolation

At the outset of the ARPANET design considerable effort was expended toward built-in techniques of fault isolation and recovery. For example, IMP-to-IMP circuits could be cross patched for fault isolation, IMPs themselves had power fail recovery mechanisms, and mechanisms for reloading one IMP from a neighbor. As the network grew, it was therefore somewhat of a surprise that the initial efforts down this path were not nearly enough. Over the life of the ARPANET program, there was a nearly continuous effort to add techniques and improve existing techniques for fault isolation, rapid recovery, and containment of failures. The lesson probably is that it is difficult to have too much attention to fault isolation and recovery mechanisms. Many interesting techniques were slowly added to improve the network's ability to cope with trouble:

o For critical places of code such as the routing computation, the code itself was checksummed before it was operated. Similarly, key data structures were checksummed before they were accessed. This kind of protection was added when it was discovered that the trouble in these particular classes of computations could cause network-wide failures rather than simply local difficulties.

- o It was discovered that dial-in modems for remote terminals could break or "hang" and there was no simple way to discover this had happened. A technique was designed wherein a centrally located autodialer controlled by the network authority used an out WATT's line to periodically dial and test all dial-in ports all over the network (at least in continental United States).

- o It was discovered that when a difficulty occurred in an IMP which caused an automatic program reload, the necessary debugging information was lost and the trouble would likely recur. A technique was added to automatically dump offending code before a reload was attempted which then permitted comparison with a master copy and improved cability for debugging.

- o The ARPANET program was forced to cope with the simultaneous need for a twenty-four hour a day continuously operating reliable system and at the same time relatively constant levels of growth, modification, and change. After some early false starts when large changes introduced periods of substandard performance, a technique was evolved whereby greater attention was paid to dividing changes into small incremental changes that were compatible with the previous system. Then, when trouble occurred, debugging could be concentrated on the small incremental change or retreat taken to the previous release.

o As network users began to depend upon a few particular "service" Hosts for a wide variety of services, including message services, it became more important that such Hosts maintain availability to the network. In some cases, a Host might be operating adequately as perceived by its own local operators and yet in some way not be properly servicing the network connection. A technique was evolved whereby the network authority added software tools to "watch" these particular critical Hosts and was then in a position to urge corrective action by the local Host authorities if and when necessary.

4.2.3 Maintenance Management

In the early years of the ARPANET program, the IMPs and TIPs of the network were maintained by subcontract to the manufacturer of the basic mini computer (Honeywell). This represented a cost-effective approach because Honeywell had maintenance facilities in many cities. However, this rather standard form of computer maintenance was simply inadequate for the high reliability requirements of the ARPANET. After a time, maintenance of the network nodes was undertaken directly by Bolt Beranek and Newman Inc., the network prime contractor, and special new techniques were developed which greatly improved the average nodal reliability. In particular, techniques were developed of central maintenance management wherein a very strong team of hardware and software experts at the central Network Control Center acted in close concert with a small number of field personnel who became highly expert in the IMP and TIP machines. The central staff could use the network itself to observe the behavior of an offending node and it could talk through difficulties with the local maintenance engineer. Further, the people in the field became much more dedicated and responsive to ARPANET difficulties as compared to time-shared Honeywell personnel who had to take responsibility for many different kinds of equipment in their geographical territory. This approach to maintenance probably has important benefits for other distributed systems, especially as those systems will increasingly be interconnected in networks and thereby accessible to central maintenance.

4.3 Impact on Other Research Areas

A final class of observation should appropriately deal with the extent to which the ARPANET program made it possible for other lines of research to be effectively pursued. There have been at least three different important ways in which the net has had this kind of impact: (1) The encouragement of research on large specialized resources which would not have been cost-effective without a network as a distribution mechanism; (2) the encouragement of computer research on the subject of interprocess communication which would be generally useful even without a network, but becomes critical in the network environment; and (3) the development of interpersonal communication techniques which again are generally useful without a network, but which become critical in a network environment.

4.3.1 Specialized Resources

Several important research projects with considerable potential for the Defense Department would simply not have been undertaken without the network's existence as a distribution tool. One example is the "National Software Works" where attempts are being made to develop integrated sets of computer tools for program design and construction which then can be used by remote groups which do not have such construction tools locally available. A second example is the Datacomputer project where a large store and efficient techniques for accessing this store were developed with the notion that this store would be used by diverse groups remotely distant from the store itself. In each case, the development of such techniques may be generally useful, but the development would not have been undertaken if the initial utilization was forced to be on a local basis.

4.3.2 Interprocess Communication

Very early in the ARPANET project, one researcher made a comment which can be paraphrased as "why bother trying to get programs to intercommunicate over a network when we don't even know how to make programs intercommunicate within a single Host". The answer is that there was not sufficient incentive to work on interprocess communication in a deep way until the existence of the network made such interprocess communication more critically necessary. Here is a case where the network has forced research in areas which will be useful even aside from network applications. Both at a formal and a practical level considerable research has now been undertaken on interprocess communication and some useful results have been obtained.

4.3.3 Person-to-Person Interactions

In the case of the spectacular success with network mail but also in some other cases of somewhat lesser impact (such as document production facilities, editing facilities, etc.), the existence of the network created a much larger community of individuals who had need to intercommunicate and thus considerable effort has been expended on such interpersonal communication tools. Such work will be of benefit in situations even without networks and will certainly be of benefit as networks become more a way of life.

4.3.4 CONCLUSIONS

These kinds of benefits have represented a very important contribution of the ARPANET program and it is somewhat fitting to end on the note that the ARPANET program has had a strong and direct feedback into the support and strength of computer science, from which the network itself sprung.