Network Working Group
Request for Comments: 30

S. Crocker
UCLA
4 February 70

DOCUMENTATION CONVENTIONS

This note is a revision of NWG/RFC 10, 16, 24, and 27.

The Network Working Group consists of interested people from existing or potential ARPA network sites. Membership is not closed.

The Network Working Group (NWG) is concerned with the HOST software, the strategies for using the network, and initial experience with the network.

Documentation of the NWG's effort is through notes such as this. Notes may be produced at any site by anybody and included in this series.

Content

The content of a NWG note may be any thought, suggestion, etc. related to the HOST software or other aspect of the network. Notes are encouraged to be timely rather than polished. Philosophical positions without examples or other specifics, specific suggestions or implementation techniques without introductory or background explication, and explicit questions without any attempted answers are all acceptable. The minimum length for a NWG note is one sentence.

These standards (or lack of them) are stated explicitly for two reasons. First, there is a tendency to view a written statement as *ipso facto* authoritative, and we hope to promote the exchange and discussion of considerably less than authoritative ideas. Second, there is a natural hesitancy to publish something unpolished, and we hope to ease this inhibition.

Form

Every NWG note should bear the following information:

1. "Network Working Group"
   "Request for Comments: *X*"
   where *X* is a serial number. Serial numbers are assigned by Steve Crocker at UCLA.

2. Author and affiliation
3. Date
4. Title
   The title need not be unique.

Distribution

One copy only will be sent from the author's site to:

1. Abhai Bhushan, MIT
2. Steve Carr, Utah
3. Gerry Cole, SDC
4. Steve Crocker, UCLA
5. Bill English, SRI
6. Jim Fry, MITRE
7. Nico Haberman, Carnegie-Mellon
8. John Heafner, RAND
9. Bob Kahn, BB&N
10. Thomas O'Sullivan, Raytheon
11. Larry Roberts, ARPA
12. Paul Rovner, LL
13. Robert Sproull, Stanford
14. Ron Stoughton, UCSB

Reproduction, if desired, may be handled locally.

Addresses

Below are the most current addresses I have. Please correct as necessary:

Abhai Bhushan Room 807 - Project MAC 545 Technology Square Cambridge, Mass. 02139	MIT (617) 864-6900 x5857
Steve Carr Computer Science Dept. University of Utah Salt Lake City, Utah 84112	Utah (801) 322-8224
Gerry Cole 7842 Croyden Los Angeles, Calif. 90045	SDC 2500 Colorado Santa Monica, Calif. 90406 (213) 393-9411, x6135 x7057 (Sec'y)
Steve Crocker 3732 Boelter Hall UCLA Los Angeles, Calif. 90024	UCLA (213) 825-4864 825-2543 (Sec'y)
Bill English Stanford Research Institute 333 Ravenswood Menlo Park, Calif. 94025	SRI (415) 326-6200

Jim Fry The MITRE Corporation Westgate Research Park McLean, Va. 22101	MITRE (703) 893-3500, x355 x318
Nico Haberman Computer Science Dept. Carnegie-Mellon University Schenley Park Pittsburgh, Pa. 15213	Carnegie-Mellon (412) 683-7000, x226
John Heafner The Rand Corporation 1700 Main Street Santa Monica, Calif. 90406	RAND (213) 393-0411
Robert Kahn Bolt, Beranek and Newman 50 Moulton Street Cambridge, Mass. 02138	BB&N (617) 491-1850
Thomas O'Sullivan Equipment Division Headquarters Raytheon Company 40 Second Avenue Waltham, Mass. 02154	Raytheon (617) 899-8400
Larry Roberts ODS/ARPA 3D167 Pentagon Washington, D.C. 20301	ARPA (202) OX7-8663 OX7-8654
Paul D. Rovner Mass. Institute of Technology Lincoln Laboratory B-115 P.O. Box 73 Lexington, Mass. 02173	LL (617) 562-5500 x7211
Robert Sproull Artificial Intelligence Project Stanford University Stanford, Calif. 94305	Stanford (415) 321-2300 x4971
Ron Stoughton Computer Research Lab. UCSB Santa Barbara, Calif. 94025	UCSB (805) 961-3221

4733

Network Working Group
Request for Comments: 31

BINARY MESSAGE FORMS IN COMPUTER NETWORKS

Daniel Bobrow
Bolt, Beranek, and Newman
Cambridge, Massachusetts

William R. Sutherland
MIT Lincoln Laboratory*
Lexington, Massachusetts

February 1968

* Operated with support from the U. S. Advanced Research Projects Agency.

MESSAGE FORMS IN COMPUTER NETWORKS

INTRODUCTION

Network communication between computers is becoming increasingly important. However, the variety of installations working in the area probably precludes standardization of the content and form of inter-computer messages. There is some hope, however, that a standard way of defining and describing message forms can be developed and used to facilitate communication between computers. Just as ALGOL serves as a standard vehicle for describing numerous algorithms, and BNF serves as a standard for describing language syntax, a message description language would be useful as a standard vehicle for defining message formats.

Considerable progress has been made at the low level of message handling protocol and one can expect the ASCII protocols to be used. The discussion which follows assumes that the mechanics of exchanging messages, check sums, repeat requestes, etc., have been worked out. The topic of concern is how to describe the content and intent of a binary message body when the network header and trailer details have been stripped off.

Most attempts at describing the content of binary messages jump immediately into a consideration of the bit codings to be used. Long, thin rectangles are drawn to represent the binary bit stream; this stream is sliced up into boxes, and tables generally describe the bit options for each box. A better approach would be to provide a symbolic method for describing messages. The symbolism, by avoiding immediate references to specific bit details, should help one's understanding of the message content and the alternatives available in the message body. When the basic form of the binary message body is clear, the coding details of the actual bit fields can be shown.

Describing a binary message body is not much different from describing a text body or language. Text assumes fixed bit fields each containing one character. Standard language description methods (BNF) then show how the characters can be concatenated and what interpretation should be placed on character groups. Binary message descriptions require the additional capacity of defining various size fields in the message and the interpretation to be placed on the bits contained in the field.

A message description is initially intended as a reference standard to be written down on paper and made available to new users of a computer network. From this standard, the new user can discover the kind and form of the binary data being exchanged over the network. Once this is known, the programs necessary for using the network facilities can be created. Later on, in an established network, one can envision the promulgation of standards for newly developed binary formats via the exchange of ASCII text messages over the network itself instead of on paper through the mail. Still farther into the future, the text of a binary format standard could be used as input to compiler-like programs which automatically create data translation programs for converting one binary format to another. Right now, though, some kind of binary data description method, however trivial, is desperately needed.

A SUGGESTED BINARY FORMAT DESCRIPTION METHOD

The basic component of a binary message is a simple field consisting of a consecutive number of bits in the message. Binary messages consist of concatenated fields. A format description for a binary message will consist of a title and four declarative sections.

1) Symbolic names are declared for all the different kinds of fields found in the binary format being defined.
2) Symbolic names are declared for commonly used values of particular fields.
3) The legal ways of concatenating fields are indicated.
4) The number of bits in each field and any special considerations of bit codings are declared.

The following is a complete example of a binary message description for a trivial kind of pictorial data.

```
TITLE: ILLUSTRATIVE GRAPHIC DATA FORMAT
       FOR A HIERARCHALLY STRUCTURED PICTURE
       OF LINES AND POINTS.

SIMPLE FIELDS:
   OPT   - Option Control Field
   COORD - Numerical Coordinate Value
   ID    - Ident number for group of picture parts
   COUNT - Number of units in message

FIELD EQUIVALENTS:
   PHDR   ← '2' OPT
   LHDR   ← '4' OPT
   GRPHDR ← '1' OPT
   GRPEND ← '3' OPT
```

CHARACTERIZATIONS:

CPAIR	←	COORD = 2
POINT	←	PHDR + CPAIR
LINE	←	LHDR + CPAIR = 2
PARTS	←	POINT/LINE/PARTS + PARTS
PIXUNIT	←	GRPHDR + ID + PARTS + GRPEND
PIXMSG	←	'5'OPT + N: COUNT + PIXUNIT = N + '0' OPT

SIMPLE FIELD SIZES:

OPT	3
COORD	14
ID	9
COUNT	6

Declaration of Simple Fields

The declaration of a simple field includes a symbolic name, and for lack of a better way, an English description of what the contents of the field represent. For example:

SIMPLE FIELDS:

F1	— Geometric Options
EXP	— STD Number - Exponent
COORD	— STD Number - Geometric Coordinate

Representing Field Values

A field with a specific value can be represented by a number in single quotes followed by the field name. A number consists of standard digits construed as binary if zeros and ones. Other numbers must be followed by a base indicator unless no confusion is possible; Q is octal, D is decimal.

Example:

'1001' F1

'300D' COORD

'27Q' EXP

Field values are integer numbers assigned such that the least significant bit is sent first. Only that part of the number which fits the field is used. Appropriate sign extension is needed for negative numbers and for numbers whose bit representation is smaller than the field.

<u>Simple Field Equivalents</u>

The declaration of a Simple Field Equivalent provides a symbolic name which represents a particular field with a specific value. Example:

FIELD EQUIVALENTS:

C1 ← '1001' F1

C2 ← '1010' F1

<u>Characterization Statement</u>

A characterization statement defines a complex field (message or message part) by indicating how other fields can be combined and is similar to a definition statement in BNF. The left side is a complex field name separated (by ←) from the concatenation indications on the right. Field names or equivalent names are concatenated by plus (+), alternatives indicated by slash (/). Slash has precedence over plus so that A + B/C means A followed by either B or C. Alternatives must be distinguishable in their own right.

Characterization statement parts can be grouped in the normal manner by parentheses. (A + B) /C means either A followed by B or C.

<u>Repetition Indicators</u>

Repeated occurrences of a field may be indicated by following the field name with an equal sign (=) and a number. For example:

CPAIR ←(COORD = 2) i.e. excatly two COORD fields

PPAIRS ←(C1 + CPAIR = 10D) / (C2 + CPAIR = 40D)

<u>Assignments Within a Characterization Statement</u>

Simple fields interpretable as integers can be assigned to a variable within the right side of a characterization statement. This variable can then be used as a repetition indicator. Example:

MS ←N1 : EXP + CPAIR = N1

indicates that MS consists of field EXP interpreted as an integer and then exactly that number of CPAIRS. All variables are global in scope.

<u>Conditional Fields</u>

Within a characterization statement a field may or may not occur depending on the contents of some other previous field. This situation is indicated by assigning a label to the determining field. The conditional occurrence is then by enclosing a condition expression and the optional field description in brackets ([and]). For example:

SS ← V:F1 + CPAIR + [V = C1 ⊃ PPAIRS]

which defines a format of 2 and perhaps 3 fields.

a) Field F1 labeled V followed by

b) Field CPAIR followed by

c) Field PPAIRS if the first field (V) was C1; otherwise, this third field is not present in the message.

Conditional Alternatives

Alternatives selected by the contents of some previous field rather than by the contents of the alternative field itself are indicated by an extension of the conditional field notation. For example:

SM : = W : F1 + CPAIR + [W = C 1 ⊃ CPAIR / C2 ⊃ PPAIRS /
'1110' ⊃ PPAIRS = '14' D]

The determining field occurs at the beginning of the conditional alternative and each alternative then includes its value for the determining field and the alternative field then present.

Size of Simple Fields

A separate field size declaration is provided.

SIMPLE FIELD SIZES:

F1	4
EXP	7
COORD	12

This size declaration should appear at the end of the message description; thus, forcing the reader to postpone an early consideration of bit details.

NGW/RFC 32

ELECTRONIC SYSTEMS LABORATORY
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Memorandum MAC-M-394

CONNECTING M.I.T. COMPUTERS TO THE ARPA COMPUTER-TO-COMPUTER COMMUNICATION NETWORK

Dietrich Vedder
January 31, 1969

ABSTRACT

The Advanced Research Projects Agency of the Department of Defense is planning to connect computing facilities of different academic and industrial institutions by a nationwide network. MIT may connect either the GE 645/Multics System or the IBM 7094/CTSS System to this network. This report explores some of the network and routing problems of a computer-to-computer communication network, and outlines the necessary hardware connection for the GE 645 and for the IBM 7094. For the GE 645, it seems best to connect the IMP (Interface Message Processor) of the ARPA Network to the Direct Common Peripheral Adapter (High-Performance Channel) of the GIOC. For the IBM 7094, it is necessary to connect to Channel D, presently occupied by the PDP-7/Kludge. A small message-distributing computer is proposed to provide ports for an IMP, the PDP-7/Kludge, the PDP-9/Kludge, and up to 15 ARDS terminals.

Submitted as Project Report for 6.842

Supervisor: John E. Ward

Work reported herein was supported (in part) by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number NOnr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

I. INTRODUCTION

The Advanced Research Projects Agency (ARPA) of the Department of Defense is planning to connect computing facilities of different academic and industrial institutions by a nationwide network. This network is to be used as a research tool to study computer communication problems. The network should also make unique computing facilities present in only one center available to many computation centers.

The network is of the store-and-forward type, in that a message is sent from one nodal point of the network to the next and is stored there before it is sent on towards its destination. Each nodal point consists of a so-called Interface Message Processor (IMP). The IMP's in the network are connected by 50-kilobit data links that make up the branches of the network. In addition to the nodal IMP's, each computing facility (called a host) connected to the network will have an IMP associated with it, and this IMP will perform most of the reformatting and message handling for the host. Each host will connect to the 50-kilobit network only through its own IMP.

M.I.T. is to be part of the network just described. It is not clear at this time, however, whether the computer to be connected to the network will be the GE 645 (Multics), or the IBM 7094 (CTSS), or possibly both. Also, little is known about the ultimate configuration of the IMP and of the network outside of the general specifications given in the initial proposals. Despite these questions, it was felt desirable to investigate the hardware implications of connecting M.I.T. computers to the ARPA network.

Section II discusses some general network considerations, including the problem of system start-up, and also presents some ideas on network structuring to avoid the classic "shuttle" problem in message routing. Section III discusses possible IMP/645 interfaces, and concludes that the Direct Common Peripheral Adapter (High-Performance Channel) of the 645 GIOC should be used. Although no specific information is available on the IMP I/O system, a cost of $2,100 in parts is estimated for construction of an IMP/HPC interface.

Considerations in connecting an IMP to the 7094 are presented in Section IV. It is concluded that a small message distributing computer added to Channel D could handle an IMP, the two buffered displays (PDP-7/Kludge, and PDP-9/Kludge), and a number of additional ARDS display terminal parts.

II. THE NETWORK

A. General Network Considerations

Messages from one host to another are to be handled in the following manner. A host will send a message to its own IMP via a host-IMP interface. This message should not be longer than $\overset{2^{13}}{8192}$ bits.* The message is then reformatted in the host's IMP to be compatible with the communication network, and it is split into small packets of uniform size. Each packet is a small message that has a header including such information as the address of the destination host, the address of the sending host, and an identification tag designating it as, say, the fifth packet belonging to a certain message of a total of nine packets. All packets belonging to the message are then sent forward according to some priority scheme and will arrive at the IMP of the destination host (not necessarily in order, since different packets may have traveled different routes). The IMP at the destination host will re-assemble the different packets belonging together into a message, reformat the message if necessary to be compatible with the destination host, and send the message to the destination host.

It is clear that an IMP in the network has only one major function, since it is only connected to other IMP's, while an IMP connected to a host has two major functions:

1. It must handle the processing associated with sending messages between it and other IMP's. This function involves:
   a. Address of decoding and routing
   b. Storage of message packets in transit

*This is one constraint put on the system to prevent overloading.

c. Handling of priorities among the different messages in transit

d. Generation and processing of messages controlling message traffic between IMP's.

2. It must handle the processing associated with sending messages between it and its host. This function involves:

   a. Reformatting (if necessary) of all messages leaving and entering the host

   b. Assembling messages to the host from packets received from the network, and splitting messages from the host into packets to be sent out on the network

   c. Control of communications between IMP and host.

How the individual functions are handled in each IMP is determined a lot by general network philosophy. The part of the report directly following therefore deals with some of the issues such as starting-up IMP's in the network, structuring the network, and routing algorithms applicable to the network.

B. The Start-Up Problem

Starting up a computer involves a definite number of steps. First, execution must be stopped at the beginning of the start-up sequence, and the computer must be initialized. Then a program is forced from some external source into a known part of memory; and the computer is then started at a specific point of that program. From then on the computer is under program control and in the run state. It can now bring other portions of a control program into memory, or it can execute programs; in other words, it can do what it was programmed to do.

The important point in this process is that the computer must be forced into a known state by external means. The simplest way to accomplish this if, of course, by having an operator do it manually. Alternatively, a second computer can perform the start up if it is directly connected to the first computer. For example, it is relatively easy to let a host start its own IMP, since we can have a special interface with special signals to implement start up.

It is a little harder to load a network IMP (not connected directly to any host) from another IMP, since we now require that:

1. all IMP's have their power turned on (this certainly must be done manually),
2. the communications adapter of the distant IMP to be started must be functioning, i.e., it must be able to synchronize itself to a received string of data,
3. the communications adapter must have hardware to recognize a special character, which would cause the IMP to be initialized and stopped and which would set up the hardware in the communication adapter such that the subsequent characters in the start-up transmission are deposited in some known area of memory,
4. at the end of the transmission the IMP must be started at some know location in memory.

The special character for start up could be some control character preceded by DLE, similar to an STX (start of text) or an ETX (end of text) preceded by a DLE.

It should be apparent than at IMP not associated with a host can only be started directly by an adjacent IMP (unless it is done manually), since the start-up sequence cannot travel through another IMP without starting that IMP. If we want to make one host responsible for starting the entire network of IMP's, then that host can only do the job indirectly (unless each IMP has an individual hardware-recognizable start-up character) by telling some other IMP to start IMP's adjacent to it.

The program for an IMP not associated with a host should come from an adjacent IMP. Since the programs of adjacent IMP's are identical (or at least can be made identical except for some routing tables), there is no problem in starting one IMP by an adjacent IMP. For host-associated IMP's, however, it does not make sense to get the program from an adjacent IMP. This is because at least the host-associated part of the program is probably unique and should be stored in the storage system of the host to which it belongs. For maintenance reasons alone, this program should reside in the host's storage system.

Now, should each host start its own IMP, or should one designated host start all IMP's, and then have each IMP associated with a host call on its host for the rest of its program. This author feels strongly that each host should start its own IMP. Some of the reasons why it should be this way are given below.

C. Some Thoughts on a Growing Computer Communications System

The research objectives for the ARPA network are stated in quite general terms. It is therefore not clear to the author whether these objectives are at all directed toward finding some generally applicable solutions for a large nationwide computer-to-computer communication network.

A large network in this case means a network of such a size that each IMP cannot easily keep in its memory all the information about how all other IMP's are interconnected.

Certainly the problems in creating such a network are numerous. Standards for message format and communications protocol* must be strictly adhered to.

The network configuration and the routing algorithms must be structured in such a manner that the system can grow easily; and it is this area of problems, which is expanded a little bit below.

Furthermore, it is not clear whether the store and forward system chosen is ultimately the best for the job. A switched channel system, which would provide a direct route for the duration of a transmission, using either analog carrier facilities or pulse code modulation carrier facilities are certainly feasible, but not as readily available at this time, as is a store and forward system.

*A.K. Bhushan, R.H. Stotz, "Message Format and Protocol for Inter-Computer Communication."

In the initial proposals for the network not much is said about network structure and routing algorithms outside of the statement that an interconnection table of all IMP's exists in each IMP, and that each IMP is to generate its own routing table. Some of the questions that come to mind are: as the system grows, does each IMP still store a complete interconnection table? How do we prevent a system failure due to two IMP's trying to deliver messages to each other, even though both IMP's decided they are too full to receive any more messages? How do we prevent two or more IMP's sending the same message back and forth, because other channels leading towards the addressed destination are busy?

The last question leads to the famous "shuttle" problem that arose in the toll network of the telephone system. This problem can be described by a simple example. Let us assume a party in Boston starts a telephone call to a party in Cleveland. The call is routed to New York, since Boston does not have any direct circuits to Cleveland. New York does have circuits to Cleveland, but they are all busy, so the decision is made to go to Philadelphia, since circuits exist from Philadelphia to Cleveland. It could turn out that Philadelphia in testing its circuits finds them all busy and returns the call to New York, since it knows also that circuits exist from New York to Cleveland. If this is allowed to happen, all the circuits between Philadelphia and New York could be made busy by one call that does not have a chance to be completed, and the traffic between Philadelphia and New York greatly disturbed. As will be seen, the telephone system has been structured so this does not happen.

The analogous situation may occur between two IMP's that are trying to send a message to a third IMP yet find the channels to the third IMP <u>temporarily</u> busy (the IMP's could be told of a permanent outage) and as a result shuttle the message back and forth, thereby possibly denying the channel between them to other messages that have a chance of getting through.

One way to solve this problem is to order all the nodal points in the network (i.e., central offices in the telephone network, IMP's in the computer communication network) in a tree structure having several levels of hierarchy. Figure 1 shows such a tree structure which is completely analogous to the tree structure now existing in the telephone system in the USA and also in European countries. The endpoints in the tree structure are denoted by capital letters A through H. Each endpoint homes on so-called primary centers, which in turn home on sectional centers, which in turn home on regional centers. Every one of the regional centers (there may be more than two) is as a rule directly connected to all other regional centers. Also, it is possible to skip levels in the hierarchy, such as endpoint E and primary center e directly homing on regional center f.

The tree structure is used in routing algorithms by first defining a backbone route, which has the property that it goes up and down in the hierarchy only once. Short cuts are not allowed in defining a backbone route. For example, the backbone route from A to D is A-a-d-b-D; the backbone route from C to F is C-b-d-f-g-h-c-F and not C-b-h-c-F or C-F.

In selecting an actual route, it is allowed (and in fact preferred) to skip as many nodes or centers in the backbone route as possible; however, it is not allowed to add any new nodes not in the backbone route to the route selected, nor select any nodes in the backbone route out of sequence. Thus, the only allowed route from B to G is B-a-d-f-g-h-c-G. The route B-a-d-b-h-c-G is not allowed, even though it has one fewer node than the backbone route, since b is a newly added node. The shuttle problem discussed above is the reason why that rule is imposed on the network. Since center b is allowed to go to d on its way to G, center d must not be allowed to go to b on its way to G.

If C wants to send a message to F, it can do so via three routes. C-F is the first choice, C-b-h-c-F is the second choice, and C-b-d-f-g-h-c-F is the third choice and identical to the backbone route.

Note that as traffic warrants, nodes in the lower levels of the tree structure may be connected directly and thereby alleviate traffic bottlenecks in the upper levels of the structure. In fact any node in the

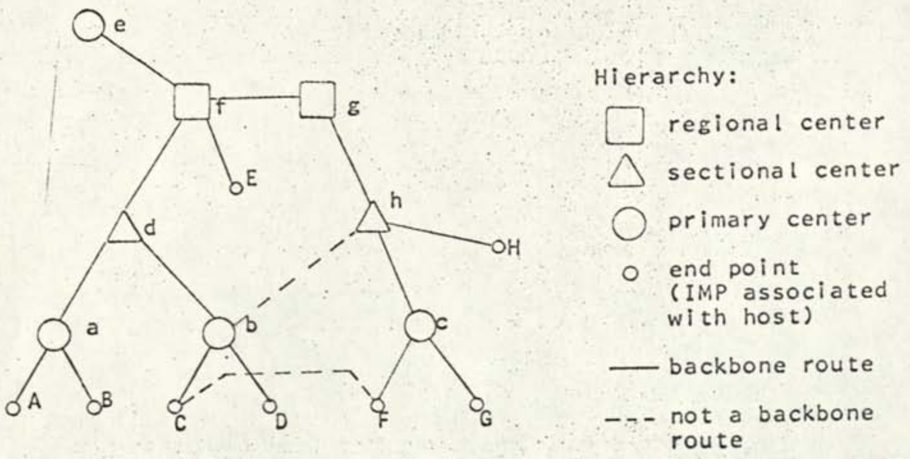

Figure 1 Hiearchical Tree Structure for Message Switching

network may still be connected to any other node in the network; the tree structure is able to accommodate all these links.

The tree structure can solve very nicely the problem of the nodal IMP's filling up with undelivered (or undeliverable) messages by use of the following rule. Given two IMP's that are connected to each other, the one having a higher status in the hierarchy can prevent the lower one from sending any messages to it, but the lower one cannot prevent the higher one from sending any messages to it. If two IMP's have equal status, such as centers f and g in Fig. 1, then one is picked arbitrarily as having a slightly higher status. This rule ensures that the higher levels of the hierarchy, where traffic tends to build up first, can get rid of their messages and thus stay operational.

It can happen, however, that specific bottlenecks develop because certain hosts or IMP's are out of service and are unable to receive messages. In this case, the hosts or IMP's that are unable to receive must be recognized, and any messages going to these computers must be turned back or rerouted. All the operational IMP's on the route of the unsuccessful message should then be informed that all messages with the same address as the unsuccessful message should either be rerouted or not accepted at all. In other words, some simple processing should be done on why a message cannot be delivered, and appropriate action should be taken all along the route of the trouble-encountering message. There are now two rules that govern message transmission:

1. Any IMP in the hierarchy must under normal condition accept messages from an IMP of higher status directly connected to it, but not vice versa.
2. Any message that encounters blocking due to an out-of-service condition of some IMP or host must cause all IMP's on its route (up to the node where the blocking occurred) to be set so that they will either reroute future messages that have the same address or not accept them.

It is of course the second rule that makes the first rule workable. The first rule implies that a lower IMP must be able to accept messages at a certain maximum rate given by the number and capacity of the connecting channels to higher IMP's. It is the second rule that guarantees that the lower IMP's will be able to get rid of messages at the same or a higher rate than they are asked to receive at.

Once the IMP network has the described tree structure superimposed on it, it is not necessary for each individual IMP to know how all other IMP's are interconnected. Each IMP would have a translation table, which maps all addresses into specific actions, these actions being the transmission of a message via a certain route or the rejection of the message due to a wrong address. The translation tables can still get large if the addresses are not properly chosen, but splitting the addresses into groups helps to alleviate the problem. We only have to look at a telephone number being split into area code + office code + number to see how the problem could be solved. footnote!

D. Autonomy of IMP's

If a communication system such as the IMP network grows to a relatively large size, it is difficult to control the network from one point. Furthermore, it is not wise for reliability reasons to have just one point control the network. One should instead make an individual IMP virtually autonomous. Unless one wants to shut down the system at night and start it in the morning, an IMP will only be started up after some failure. But a failure will usually require some maintenance work, performed by a knowledgeable person. It makes much more sense to let him start up the IMP after correcting the problem than to have a distant host direct the start up of the machine. Furthermore, it should not be too hard to construct the programs in such a way that the basic program is identical (except for some data areas) for each IMP that is not connected to a host. It should therefore be easy to get the program for each individual IMP from an adjacent IMP.

Since each IMP directly connected to a host has some special programs to interface it to the host, it should be the host that should store the backup copy of the IMP's program. Also, since the host's IMP appears as an I/O device to the host and is physically near the host, it does make sense to start the IMP from the host, although it is not absolutely necessary to do it that way.

III. THE CONNECTION BETWEEN THE GE-645 AND THE IMP

A. Comparison of the Word Synchronous Adapter and the High-Performance Channel

The IMP will appear to the GE-645 as just another I/O device, and must therefore be connected to the General Input Output Controller (GIOC) of the GE-645. Two different adapters were considered for connecting the IMP to the GIOC: the Word Synchronous Adapter, and the Direct Common Peripheral Adapter (also called High Performance Channel = HPC).*

The Word Synchronous Adapter (WSA-600) is an indirect adapter, which means that it does not store its own Data Control Word (DCW). The GIOC therefore has to access memory three times for each 36-bit word it transfers for the WSA-600. The first memory access is used to obtain the DCW from the proper mailbox, the second memory access is used for data transfer, and the third memory access is used to store the updated DCW in its mailbox.

The WSA-600 is full duplex, therefore, data transfer can occur simultaneously in both directions. Seventy-two bits of storage is provided for each direction of transfer, which splits into one word of buffering and one word of shift register. Character assembly and disassembly is done on a plug-selectable basis for six + parity, seven + parity, and four-out-of-eight code. The WSA-600 is capable of transmitting and receiving at a maximum rate of 240 kilobits/second. If seven + parity code is used

*Actually a third adapter, the Custom Direct Adapter (CDA), was also considered. This adapter performs even better than the HPC (it transfers 12-bit, 18-bit, or 36-bit words at a time) but it would probably be more expensive to rent, since it has more equipment than the HPC (7 rows of modules for the CDA versus 5 rows for the HPC). The real problem with this adapter is the fact that no hardware exists yet; and it is less likely to be put into production than the WSA-600.

between the GIOC and the IMP, then a rate of 30,000 characters per second can be sustained. The major disadvantage of the WSA-600 is that it presently exists only in specification form; the adapter itself has not been completely designed.

The High Performance Channel (HPC) is a direct adapter. It therefore stores the 72-bit DCW and needs only one memory cycle to do a data transfer. A 36-bit single word or a 72-bit double word may be transferred at a time. The HPC has a buffer register of 72 bits plus another 72-bit character assembly area. Six bits plus parity are transferred at a time to and from the connected I/O device. Data is transferred in only one direction at a time. The maximum data transfer rate is 400,000 characters per second, which is equal to 2.4 million bits per second.

footnote

B. Some of the Reasons Why the HPC Should be Selected

The HPC is, of course, an existing and working circuit. On the other hand, the decision to complete development of the WSA-600 rests with the General Electric Company and will depend probably on their market predictions for the WSA-600. While a 50-kilobit data-set interface is the only I/O device that promises to be a standard feature of the IMP, favoring connection to a WSA-600 that has the same interface, it can be argued that a nonstandard HPC interface for a small computer such as the IMP can be designed and built relatively easily here at MIT.

Beside the problems of procuring the actual interfaces, questions of speed and efficiency greatly favor the HPC. We surely do not want to run the interface between the IMP and the GIOC at less than 50-kilobits per second. This is because the entire IMP communication system runs at that bit rate, and providing less than that between host and IMP would surely create a bottleneck. At the rate of 50 kilobits per second, efficient data transfer through the GIOC is important in order that the other I/O operations are not slowed down. The HPC is six times more efficient than the WSA-600 (this efficiency of the HPC is due to the storage of the DCW in the adapter and the buffering and transfer to the GIOC of a double word instead of a single word). The actual maximum data transfer rate of the HPC is 2.4 million bits per second and is ten times as large as the maximum transfer rate of the WSA-600. This relatively high transfer rate is nice to have, especially if the IMP connected to the GIOC terminates several 50-kilobit lines to nearby universities such as Harvard and Dartmouth that ultimately may end up having a high traffic rate with MIT.

In short, the IMP should be connected to the GIOC via the HPC for three main reasons:

1. An interface circuit between HPC and IMP can be procured relatively easily.
2. The GIOC operates much more efficiently if the IMP is connected via an HPC.
3. The maximum data transfer rate is high enough to avoid bottlenecks, even if traffic into and out of the GE-645 computer system is heavy.

It should also be said that estimates of monthly rental charges for the HPC and the WSA-600 are about equal; so there is no cost disadvantage in picking the HPC.

C. <u>Interface Requirements for the HPC</u>

The interface requirements are specified in detail in the General Electric Product Performance Specification for the "Common Peripheral Interface" (43A130524). Answers to any detailed questions should be found in this document. Only a general outline of the interface and of the hardware required to connect a small computer and the HPC will be given here.

The following leads interconnect to the HPC:

Lines from the HPC:

1. There are seven information lines from the HPC (6 data lines + parity).

2. Three lines from the HPC are used to control data transfer operations: the Read Clock Line, the Write Clock Line, and the End Data Transfer Line.
3. The I/O Line from the HPC is used to send commands from the GE-645 to the peripheral (in this case the IMP).
4. A Program Load Line from the HPC is used to request the peripheral to send one record for bootstrap loading and similar purposes. This line is probably not necessary in the IMP interface.
5. A peripheral Reset Line from the HPC tells the IMP that the GE-645 is operating or not operating.

Lines to the HPC:

1. There are seven information lines to the HPC (6 data lines + parity).
2. Four Major Status Lines transmit status information to the HPC.
3. Three lines to the HPC are used to control data transfer operations: the Read Clock Line, the Write Clock Line, and the Terminate Line.
4. A Special Interrupt Line to the HPC allows the IMP to demand action from the GE-645.
5. An External Reset Line to the HPC tells the HPC whether the IMP is operational or not.

All lines between the HPC and the IMP (except the Major Status Lines to the HPC and the External Reset Lines in both directions) carry pulses and must conform to the specifications set by General Electric. On the IMP side, all of these lines must therefore have the proper pulse drivers or pulse receivers as appropriate. The Major Status Lines carry levels instead of pulses and must be equipped accordingly.

The External Reset Leads in both directions have a relay signaling system. An Enabled condition is signaled if the center conductor of the External Reset Lead is connected to the shield, and a Disabled condition is signaled if the center conductor is not connected to the shield. The connecting function is done by mercury wetted contacts ensuring clean switching.

The External Reset Lead keeps its respective side in the reset state as long as the other side has not signaled the Enable condition. The External Reset Lead thus accomplishes the suppression of line transients while the HPC and the IMP are disconnected or while one side has its power off or is disabled for some other reason. Note that the signaling convention very nicely takes care of the disconnect or power off condition, since the External Reset Lead will be open and thus will keep its respective side reset.

The Communications between IMP and HPC can be implemented by using commands from HPC to the IMP and status information and interrupts from IMP to HPC.

The HPC (and the GE-645 behind it) is, of course, in control of the communications between HPC and IMP; it can send commands to the IMP interface initiating appropriate operations. The IMP interface can send status information back to the HPC, thereby causing action indirectly. Similar to teletype consoles, the IMP would also be able to send an interrupt to the HPC and thus initiate new action.

Commands are sent from the HPC to the IMP on the seven information leads. Status is sent back to the HPC via the four Major Status Lines and also via the seven information leads. The detailed procedure for passing commands and status information is given in the previously mentioned General Electric document. Also, some restrictions and conventions in assigning meaning to the command and status words are given in that document.

D. An Estimate of the Cost of an HPC/IMP Interface

One cannot design a specific interface without knowing the detailed specifications of the IMP; but an estimate of the cost of an IMP interface is possible, since most small computers have similar I/O features. It is assumed in this estimate that the IMP has a feature similar to the

Direct Memory Access feature of the PDP-9. (The 8-million bit maximum transfer rate specified for the IMP-host interface virtually dictates such a feature.)

The following is a list of essential hardware for the interface:

7-bit register	(Receive and transmit buffer, pulse receivers included)
12-bit register	(Two-character buffer)
4-bit register	(Major Status)
7 pulse drivers	(7 information lines to HPC)
4 cable drivers	(to transmit Major Status)
2 mercury relays	(for External Reset Lines)
4 pulse receivers	(for pulse receiving on control leads)
4 pulse drivers	(for pulse sending on control leads)
3 flip-flops and 18 gates	(to distribute character over entire word)
13-bit register	(address register for PDP-9 DMA)
control logic	

From the above list we can estimate the cost assuming DEC modules.

Items	Cost for Each	Total Cost
39 flip-flops	$15	$ 585
11 pulse drivers	11	121
2 mercury contact relays	10	20
18 gates	5	90
4 cable drivers	12	48
4 pulse receivers	15	60
control logic	--	500
2 mounting trays	142	284
I/O bus	--	380
		$2,088

Thus the cost of an IMP interface is about $2,100 not counting any wiring, installation, and engineering cost.

IV. THE CONNECTION BETWEEN THE IBM-7094 AND THE IMP

A. General Discussion

The IBM-7094 with its time-sharing system CTSS is an established system with a number of capabilities that may prove useful to the community of users to be connected by the IMP network. It is therefore possible that the IMP network may at first be connected to the IBM 7094.

A hardware connection to the IBM-7094 is not as straightforward as to the GE-645. The only available high-data-rate connection to the IBM-7094 is channel D, and that channel is already used to handle the two existing ESL refreshed display consoles (Kludges). Figure 2 shows the present configuration, which at this time is completed except for the installation of the data link. In this configuration the PDP-7 is used to handle the Kludge 1 and is also used to transfer messages and display lists from the 7094 to the PDP-9 and vice versa. If we want to connect the IMP, we have to use channel D, not only for the IMP, but also for Kludge 1 and Kludge 2. Furthermore, we possibly would like to connect a number of ARDS ports to channel D, since the number of ARDS ports provided by the IBM-7750 is limited to at most eight ports (four at the moment).

There are at least two possible ways to handle this channel multiplexing problem. One can use the PDP-7 as the multiplexing computer in addition to its task of refreshing the Kludge 1, or one can use a separate message distributing computer. Technical considerations favor the separate message distributing computer slightly, since it represents a "cleaner" solution in that the different functions, such as driving a Kludge and distributing messages, are separated and since it is easier to troubleshoot and maintain the system.

It turns out that the economics also favor the separate message distributing computer. A look at the memory requirements makes this clear. If one wants to run up to 15 ARDS ports, about 3.3K words of storage are needed for the ARDS ports alone (See Appendix A). An additional 2K words of storage are needed for temporarily storing display lists or messages

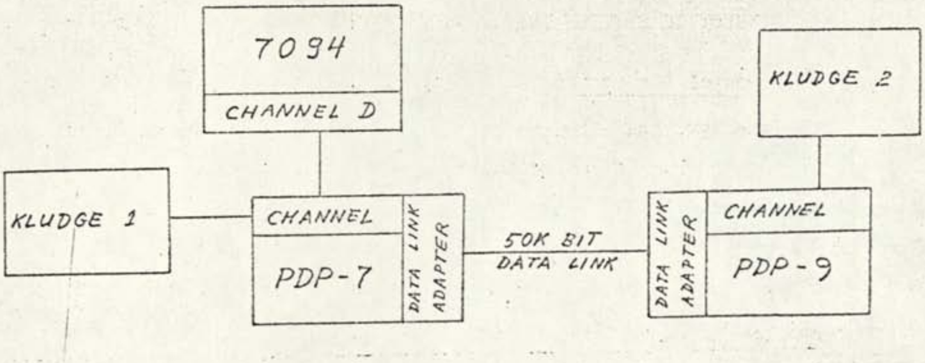

Figure 2 Present Equipment on 7094 Channel D

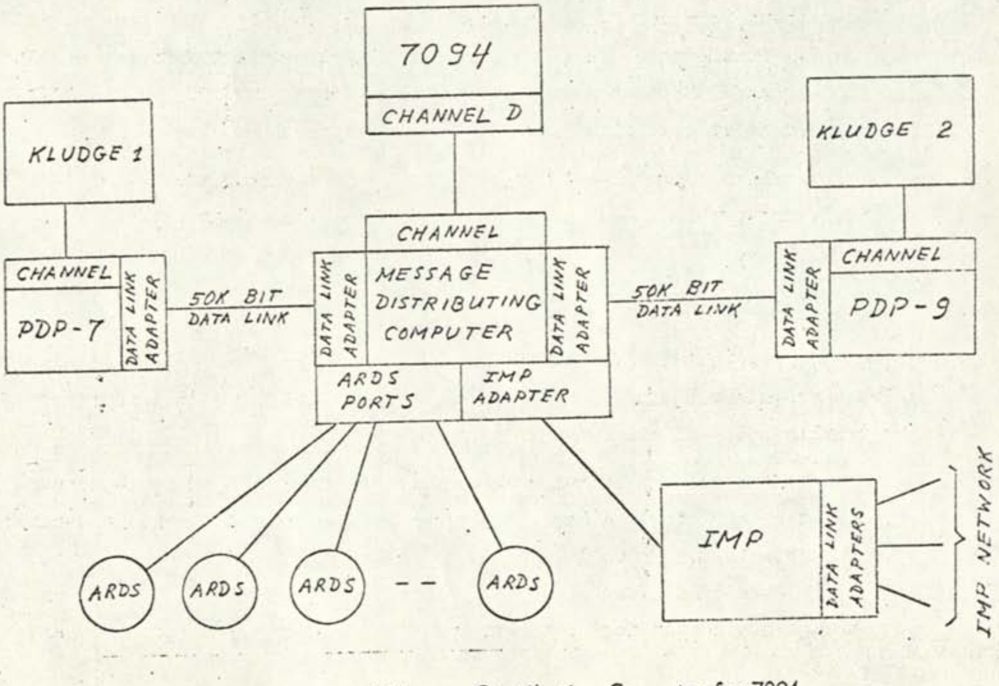

Figure 3 Proposed Message Distributing Computer for 7094

to and from the IMP. If the PDP-7 is to be the multiplexing computer, another 2K words is needed for the display list of Kludge 1. Since the present storage size of the ESL PDP-7 is 8K words, this would leave only 700 words for all programs in the PDP-7. An increase of the PDP-7 memory is therefore needed for this solution. Furthermore, it is not advisable to run the multiplexing program in the PDP-7 without a memory protect feature, since we want to allow Kludge users to modify programs in the PDP-7. As a result, if the PDP-7 is used as a multiplexing computer, we need at least another 4K of memory, a memory extension control, and a memory protect option. The cost of these features are as follows:

4K memory bank (installed)	$21,020
Memory extension control (installed)	8,340
Memory protect option (installed)	900
	$30,280

But for thirty thousand dollars we should be able to get a message distributing computer with the necessary I/O equipment to do the job. Figure 3 gives the configuration for a separate message distributing computer.

B. I/O Configuration of the Message Distributing Computer

Since both the PDP-7 and the PDP-9 already have 50-kilobit data link adapters, it is easy to connect them to the message distributing computer via 50-kilobit data links and data link adapters. Thus both Kludges can be moved around on the campus and do not have to be adjacent to the 7094.

Since the 50K bit data link arrangements are character oriented (7 bits + parity), there is a 160 microsecond time interval for responding to a data transfer request from the data link adapters, provided double buffering is used for the characters in the data link adapters.

The connection to the 7094 should probably be done via a channel very similar in structure to the channel built for the PDP-7. A direct memory access feature in the small computer is very desirable for this purpose.

The ARDS ports should consist of half-duplex circuitry capable of receiving at 150 bits/second and transmitting at 1200 bits/second. Double buffering of characters in each ARDS port is not necessary if it can normally be guaranteed that the message distributing computer can respond to a data transfer request by an ARDS port within 800 microseconds. If this is not possible, then double buffering of characters will lengthen the required response time to about 7 milliseconds.

It is hard to say what the adapter to the IMP should look like, since a number of policy decisions are involved. If it is not considered essential that the IMP can be started remotely from the 7094, then a 50K bit adapter, even if used without a data link, is probably best, since both the IMP and the message distributing computer can easily be provided with an additional data link adapter. If remote starting capability is required, then the IMP interface must either be a channel interface that is capable of sending a special start command or a 50K bit data link adapter capable of responding to a special start character. It should be clear that the message distributing computer must also have the capability of being started remotely, if the IMP is to have that capability.

C. Features of the Message Distributing Computer

The message distributing computer should have a memory size of 8K to accommodate up to about 15 ARDS ports and one display list or IMP message simultaneously. The protocol for message distribution and memory allocation should be arranged to give messages from the IMP to the 7094 top priority, since we must ensure that the IMP network can get rid of its message (see discussion on IMP network and congestion).

An index register would be helpful to do the large amount of list processing. Efficient shifting operations are also necessary to do the conversion from characters to words and vice versa.

APPENDIX A

Required Memory for Operating a Number of ARDS Ports

If we want to connect a number of ARDS ports to the IBM 7094/CTSS through a message distributing computer, then the required memory in the message distributing computer depends a lot on how often we are willing to allow user programs in the 7094 to be swapped in and out of core. Transmission to ARDS is at a rate of 1200 bits per second. Since each character is 10 bits, this is 120 characters per second. If we can guarantee that normally 600 characters can be stored in the message distributing computer for any particular ARDS port (but not all ARDS ports simultaneously), then the time interval between output imposed swaps is normally longer than five seconds; if we allow ten seconds worth or storage, then this interval is longer than ten seconds, and so on.

The following assumptions were made in order to calculate the memory requirements for ARDS output:

1. The output bit rate is 1200 bits/second.
2. It is assumed that on the average only 20 percent of the ARDS terminals are outputting at one time.
3. It is assumed that the number of ARDS ports is 5, 10, or 15.
4. It is assumed for the purpose of calculation that the maximum number of characters stored per ARDS port is either 600 or 1200 (5 or 10 seconds worth respectively).
5. The behavior at an individual ARDS port is statistically independent from all other ARDS ports.

If we are willing to store a maximum number of 600 characters (6000 bits) for an individual port, then the average number of bits per port is: $6000 \times 0.2 = 1200$ bits/port.

The standard deviation can be computed as follows:

$$\sigma^2 = E(x^2) - [E(x)]^2 = 6000^2 \times 0.2 - 1200^2$$

$$\sigma^2 = 5.76 \times 10^6$$

$$\sigma = 2400 \text{ bits (for one port)}$$

$$\sigma = \sqrt{n} \times 2400 \text{ bits (for n ports)}$$

If we are willing to store a maximum number of 1200 characters for an individual port, then the average number of bits is 2400 bits/port and $\sigma = \sqrt{n} \times 4800$ bits for n ports.

Below are two tables for a maximum storage of 600 and 1200 characters per ARDS port respectively, giving the required storage for 5, 10, and 15 ports.

	5 ports	10 ports	15 ports
max. number of bits	30,000	60,000	90,000
3σ limit (bits)	22,000	34,800	45,900
3σ limit (words) (14 bits/word)	1,570	2,490	3,290

Table 1. Maximum Allowed Storage per ARDS Port is 600 Characters

	5 ports	10 ports	15 ports
max. number of bits	60,000	120,000	180,000
3σ limit (bits)	44,000	73,600	91,800
3σ limit (words) (14 bits/word)	3,140	4,980	6,580

Table 2. Maximum Allowed Storage per ARDS Port is 1200 Characters

The tables show the absolute maximum amount of storage ever needed, and also the amount of required storage that we get if we add 3 standard deviations to the average amount of storage needed. The actual amount of storage needed at any moment is very unlikely to be above the amount given by the 3 σ limit, provided the initial assumptions hold.

Note that the assumption on maximum number of characters per ARDS terminal was made only for purpose of calculating some figures on required memory. Now having the figures, we can turn around and say, given the amount of storage and number of ports in the table, it is very unlikely that we will be running into output storage limitations if a user program happens to generate up to but not more than 600 or 1200 characters of output at one time.

In the main report, 3.3K words was taken as a bogey figure for the amount of required memory for ARDS ports. This figure is based on 15 ARDS ports and maximum allowed storage of 600 characters per port (see Table 1). Note however that another 2K was reserved for display list storage. It so happens that display lists of this size are sent very infrequently; furthermore, we do not expect a large amount of IMP traffic initially. Therefore, an extra 2K of memory should usually be available for ARDS traffic, raising the maximum storage allowance per port to about 1000 characters per port. The initial 8K memory size requirement for the message distributing computer should therefore yield good response. In the future, when the system is running and IMP traffic increases to significant levels, actual traffic measurements can supersede the estimates above, and possible memory extensions can be planned more accurately.

A word about buffering inputs into ARDS ports is in order. Presently the IBM 7750 sends every full character received from a port on to the 7094 as soon as it is assembled. The storage requirements for ARDS input are therefore very small, unless we significantly change the present way or operating.

4735

Network Working Group
Request for Comments: 33

S. Crocker
12 February 70

New HOST-HOST Protocol

Attached is a copy of a paper to be presented at the SJCC on the HOST-HOST Protocol. It indicates many changes form the old protocol in NWG/RFC 11; these changes resulted from the network meeting on December 8, 1969. The attached document does not contain enough information to write a NCP, and I will send out another memo or so shortly. Responses to this memo are solicited, either as NWG/RFC's or personal notes to me.

HOST-HOST Communication Protocol in the ARPA Network*

by C. Stephen Carr
University of Utah
Salt Lake City, Utah

and

by Stephen D. Crocker
University of California
Los Angeles, California

and

by Vinton G. Cerf
University of California
Los Angeles, California

* This research was sponsored by the Advanced Research Projects Agency, Department of Defense, under contracts AF30(602)-4277 and DAHC15-69-C-0285.

INTRODUCTION

The Advanced Research Projects Agency (ARPA) Computer Network (hereafter referred to as the "ARPA network") is one of the most ambitious computer networks attempted to date.[1] The types of machines and operating systems involved in the network vary widely. For example, the computers at the first four sites are an XDS 940 (Stanford Research Institute), an IBM 360/75 (University of California, Santa Barbara), an XDS SIGMA-7 (University of California, Los Angeles), and a DEC PDP-10 (University of Utah). The only commonality among the network membership is the use of highly interactive time-sharing systems; but, of course, these are all different in external appearance and implementation. Furthermore, no one node is in control of the network. This has insured generality and reliability but complicates the software.

Of the networks which have reached the operational phase and been reported in the literature, none have involved the variety of computers and operating systems found in the ARPA network. For example, the Carnegie-Mellon, Princeton, IBM network consists of 360/67's with identical software.[2] Load sharing among identical batch machines was commonplace at North American Rockwell Corporation in the early 1960's. Therefore, the implementers of the present network have been only slightly influenced by earlier network attempts.

However, early time-sharing studies at the University of California at Berkeley, MIT, Lincoln Laboratory, and System Development Corporation (all ARPA sponsored) have had considerable influence on the design of the network. In some sense, the ARPA network of time-shared computers is a natural extension of earlier time-sharing concepts.

The network is seen as a set of data entry and exit points into which individual computers insert messages destined for another (or the same) computer, and from which such messages emerge. The format of such messages and the operation of the network was specified by the network contractor (BB&N) and it became the responsibility of representatives of the various computer sites to impose such additional constraints and provide such protocol as necessary for users at one site to use resources at foreign sites. This paper details the decisions that have been made and the considerations behind these decisions.

Several people deserve acknowledgment in this effort. J. Rulifson and W. Duvall of SRI participated in the early design effort of the protocol and in the discussions of NIL. G. Deloche of Thomson-CSF participated in the design effort while he was at UCLA and provided considerable documentation. J. Curry of Utah and P. Rovner of Lincoln Laboratory reviewed the early design and NIL. W. Crowther of Bolt, Beranek and Newman contributed the idea of a virtual net. The BB&N staff provided substantial assistance and guidance while delivering the network.

We have found that, in the process of connecting machines and operating systems together, a great deal of rapport has been established between personnel at the various network node sites. The resulting mixture of ideas, discussions, disagreements, and resolutions has been highly refreshing and beneficial to all involved, and we regard the human interaction as a valuable by-product of the main effort.

THE NETWORK AS SEEN BY THE HOSTS

Before going on to discuss operating system communication protocol, some definitions are needed.

A HOST is a computer system which is part of the network.

An IMP (Interface Message Processor) is a Honeywell DDP-516 computer which interfaces with up to four HOSTs at a particular site, and allows HOSTs access into the network. The configuration of the initial four-HOST network is given in Figure 1. The IMPs form a store-and-forward communications network. A companion paper in these proceedings covers the IMPs in some detail.[3]

A message is a bit stream less than 8096 bits long which is given to an IMP by a HOST for transmission to another HOST. The first 32 bits of the message are the leader. The leader contains the following information:

(a) HOST

(b) Message type

(c) Flags

(d) Link number

When a message is transmitted from a HOST to its IMP, the HOST field of the leader names the receiving HOST. When the message arrives at the receiving HOST, the HOST field names the sending HOST.

Only two message types are of concern in this paper. Regular messages are generated by a HOST and sent to its IMP for transmission to a foreign

HOST. The other message type of interest is a RFNM (Request-for-Next-Message). RFNM's are explained in conjunction with links.

The flag field of the leader controls special cases not of concern here.

The link number identifies over which of 256 logical paths (links) between the sending HOST and the receiving HOST the message will be sent. Each link is unidirectional and is controlled by the network so that no more than one message at a time may be sent over it. This control is implemented using RFNM messages. After a sending HOST has sent a message to a receiving HOST over a particular link, the sending HOST is prohibited from sending another message over that same link until the sending HOST receives a RFNM. The RFNM is generated by the IMP connected to the receiving HOST, and the RFNM is sent back to the sending HOST after the message has entered the receiving HOST. It is important to remember that there are 256 links in each direction and that no relationship among these is imposed by the network.

The purpose of the link and RFNM mechanism is to prohibit individual users from overloading an IMP or a HOST. Implicit in this purpose is the assumption that a user does not use multiple links to achieve a wide band, and to a large extent the HOST-HOST protocol cooperates with this assumption. An even more basic assumption, of course, is that the network's load comes from some users transmitting sequences of messages rather than many users transmitting single messages coincidently.

In order to delimit the length of the message, and to make it easier for HOSTs of differing word lengths to communicate, the following formatting procedure is used. When a HOST prepares a message for output, it creates a 32-bit leader. Following the leader is a binary string, called *marking*, consisting of an arbitrary number of zeroes, followed by a one. Marking makes it possible for the sending HOST to synchronize the beginning of the text of a message with its word boundaries. When the last bit of a message has entered an IMP, the hardware interface between the IMP and HOST appends a one followed by enough zeroes to make the message length a multiple of 16 bits. These appended bits are called padding. Except for the marking and padding, no limitations are placed on the text of a message. Figure 2 shows a typical message sent by a 24-bit machine

DESIGN CONCEPTS

The computers participating in the network are alike in two important respects: each supports research independent of the network, and each is under the discipline of a time-sharing system. These facts contributed to the following design philosophy.

First, because the computers in the network have independent purposes it is necessary to preserve decentralized administrative control of the various computers. Since all of the time-sharing supervisors possess elaborate and definite accounting and resource allocation mechanisms, we arranged matters so that these mechanisms would control the load due to the network in the same way they control locally generated load.

Second, because the computers are all operated under time-sharing disciplines, it seemed desirable to facilitate basic interactive mechanisms.

Third, because this network is used by experienced programmers it was imperative to provide the widest latitude in using the network. Restrictions concerning character sets, programming languages, etc. would not be tolerat and we avoided such restrictions.

Fourth, again because the network is used by experienced programmers, it was felt necessary to leave the design open-ended. We expect that conventions will arise from time to time as experience is gained, but we felt constrained not to impose them arbitrarily.

Fifth, in order to make network participation comfortable, or in some cases, feasible, the software interface to the network should require minimal surgery on the HOST operating system.

Finally, we accepted the assumption stated above that network use consists of prolonged conversations instead of one-shot requests.

These considerations led to the notions of connections, a Network Control Program, a control link, control commands, sockets, and virtual nets.

A connection is an extension of a link. A connection connects two processes so that output from one process is input to the other. Connections are simplex, so two connections are needed if two processes are to converse in both directions.

Processes within a HOST communicate with the network through a Network Control Program (NCP). In most HOSTs, the NCP will be part of the executive, so that processes will use system calls to communicate with it. The primary function of the NCP is to establish connections, break connections, switch connections, and control flow.

In order to accomplish its tasks, a NCP in one HOST must communicate with a NCP in another HOST. To this end, a particular link between each pair of HOSTs has been designated as the control link. Messages received over the control link are always interpreted by the NCP as a sequence of one or more control commands. As an example, one of the kinds of control commands is used to assign a link and initiate a connection, while another kind carries notification that a connection has been terminated. A partial sketch of the syntax and semantics of control commands is given in the next section.

A major issue is how to refer to processes in a foreign HOST. Each HOST has some internal naming scheme, but these various schemes often are incompatible. Since it is not practical to impose a common internal process naming scheme, an intermediate name space was created with a separate portion of the name space given to each HOST. It is left to each HOST to map internal process identifiers into its name space.

The elements of the name space are called sockets. A socket forms one end of a connection, and a connection is fully specified by a pair of sockets. A socket is specified by the concatenation of three numbers:

(a) a user number (24 bits)

(b) a HOST number (8 bits)

(c) AEN (8 bits)

A typical socket is illustrated in Figure 3.

Each HOST is assigned all sockets in the name space which have field (b) equal to the HOST's own identification.

A socket is either a receive socket or a send socket, and is so marked by the low-order bit of the AEN (0 = receive, 1 = send). The other seven bits of the AEN simply provide a sizable population of sockets for each user number at each HOST. (AEN stands for "another eight-bit number".)

Each user is assigned a 24-bit user number which uniquely identifies him throughout the network. Generally this will be the 8-bit HOST number of his home HOST, followed by 16 bits which uniquely identify him at that HOST. Provision can also be made for a user to have a user number not keyed to a particular HOST, an arrangement desirable for mobile users who might have no home HOST or more than one home HOST. This 24-bit user number is then used in the following manner. When a user signs onto a HOST, his user number is looked up. Thereafter, each process the user creates is tagged with his user number. When the user signs onto a foreign HOST via the network, his same user number is used to tag processes he creates in that HOST.* The foreign HOST obtains the user number either by consulting a table at login time, as the home HOST does, or by noticing the identification of the caller. The effect of propagating the user's number is that each user creates his own virtual net consisting of processes he has created. This virtual net may span an arbitrary number of HOSTs. It will thus be easy for a user to connect his processes in arbitrary ways, while still permitting him to connect his processes with those in other virtual nets.

WHAT ABOUT PROCESSES ALREADY RUNNING IN THAT HOST

The relationship between sockets and processes is now describable (see Figure 4). For each user number at each HOST, there are 128 send sockets and 128 receive sockets. A process may request from the local NCP the use of any one of the sockets with the same user number; the request is granted if the socket is not otherwise in use. The key observation here is that a socket requested by a process cannot already be in use unless it is by some other process within the same virtual net, and such a process is controlled by the same user.

An unusual aspect of the HOST-HOST protocol is that a process may switch its end of a connection from one socket to another. The new socket may be in any virtual net and at any HOST, and the process may initiate a switch either at the time the connection is being established, or later. The most general forms of switching entail quite complex implementation, and are not germane to the rest of this paper, so only a limited form will be explained. This limited form of switching provides only that a process may substitute one socket for another while establishing a connection. The new socket must have the same user number and HOST number, and the connection is still established to the same process. This form of switching is thus only a way of relabelling a socket, for no change in the routing of messages takes place. In the next section we document the system calls and control commands; in the section after next, we consider how login might be implemented.

SYSTEM CALLS AND CONTROL COMMANDS

Here we sketch the mechanics of establishing, switching and breaking a connection. As noted above, the NCP interacts with user processes via system calls and with other NCPs via control commands. We therefore begin with a partial description of system calls and control commands.

System calls will vary from one operating system to another, so the following description is only suggestive. We assume here that a process has several input-output paths which we will call ports. Each port may be connected to a sequential I/O device, and while connected, transmits information in only one direction. We further assume that the process is blocked (dismissed, slept) while transmission proceeds. The following is the list of system calls:

Init <port>, <AEN 1>, <AEN 2>, <foreign socket>

where <port> is part of the process issuing the Init

and <AEN 1> } <AEN 2> } are 8-bit AEN's (see Figure 3)

The first AEN is used to initiate the connection; the second is used while the connection exists.

<foreign socket> is the 40-bit socket name of the distant end of the connection.

The low-order bits of <AEN 1 > and <AEN 2> must agree, and these must be the complement of the low-order bit of <foreign socket>.

The NCP concatenates <AEN 1> and <AEN 2> each with the user number of the process and the HOST number to form 40-bit sockets. It then sends a Request for Connection (RFC) control command to the distant NCP. When the distant NCP responds positively, the connection is established and the process is unblocked. If the distant NCP responds negatively, the local NCP unblocks the requesting process, but informs it that the system call has failed.

Listen <port>, <AEN 1>

where <port> and <AEN 1> are as above. The NCP retains the ports and <AEN 1> and blocks the process. When an RFC control command arrives naming the local socket, the process is unblocked and notified that a foreign process is calling.

Accept <AEN 2>

After a Listen has been satisfied, the process may either refuse the call or accept it and switch it to another socket. To accept the call, the process issues the Accept system call. The NCP then sends back an RFC control command.

RESUME ?

Close <port>

After establishing a connection, a process issues a Close to break the connection. The Close is also issued after a Listen to refuse a call.

Transmit <port>, <addr>

If <port> is attached to a send socket, <addr> points to a message to be sent. This message is preceded by its length in bits.

If <port> is attached to a receive socket, a message is stored at <addr>. The length of the message is stored first.

Control commands

A vocabulary of control commands has been defined for communication between Network Control Programs. Each control command consists of an 8-bit operation code to indicate its function, followed by some parameters. The number and format of parameters is fixed for each operation code. A sequence of control commands destined for a particular HOST can be packed into a single control message.

RFC <my socket 1>, <my socket 2>,
<your socket>, (<link>)

This command is sent because a process has executed either an Init system call or an Accept system call. A link is assigned by the prospective receiver, so it is omitted if <my socket 1> is a send socket.

There is distinct advantage in using the same commands both to initiate a connection (Init) and to accept a call (Accept). If the responding command were different from the initiating command, then two processes could call each other and become blocked waiting for each other to respond. With this scheme no deadlock occurs and it provides a more compact way to connect a set of processes.

CLS <my socket>, <your socket>

The specified connection is terminated

CEASE <link>

When the receiving process does not consume its input as fast as it arrives, the buffer space in the receiving HOST is used to queue the waiting messages. Since only limited space is generally available, the receiving HOST may need to inhibit the sending HOST from sending any more messages over the offending connection. When the sending HOST receives this command, it may block the process generating the messages.

RESUME <link>

This command is also sent from the receiving HOST to the sending HOST and negates a previous CEASE.

LOGGING IN

NOT SO IN ALL CASES We assume that within each HOST there is always a process in execution which listens to login requests. We call this process the logger, and it is part of a special virtual net whose user number is zero. The logger is programmed to listen to calls on socket number 0. Upon receiving a call, the logger switches it to a higher (even) numbered sockets, and returns a call to the socket numbered one less than the send socket originally calling. In this fashion, the logger can initiate 127 conversations.

To illustrate, assume a user whose identification is X'010005' (user number 5 at UCLA) signs into UCLA, starts up one of his programs, and this program wants to start a process at SRI. No process at SRI except the logger is currently willing to listen to our user, so he executes

```
        Init, <port> = 1, <AEN 1> = 7, <AEN 2> = 7,
              <foreign socket> = 0.
```

His process is blocked, and the NCP at UCLA sends

```
        RFC   <my socket 1> = X'0100050107',   ← HEX
              <my socket 2> = X'0100050107',
              <your socket> = X'0000000200'
```

The logger at SRI is notified when this message is received, because it has previously executed

```
        Listen      <port> = 9, <AEN 1> = 0.
```

The logger then executes

```
        Accept      <AEN 2> = 88.
```

In response to the Accept, the SRI NCP sends

```
        RFC         <my socket 1> = X'0000000200'
                    <my socket 2> = X'0000000258'
                    <your socket> = X'0100050107'
                    <link> = 37
```

where the link has been chosen from the set of available links. The SRI logger then executes

```
        Init        <port> = 10
                    <AEN 1> = 89, <AEN 2> = 89,
                    <foreign socket> = X'0100050106'
```

which causes the NCP to send

```
        RFC         <my socket 1> = X'0000000259'
                    <my socket 2> = x'0000000259'
                    <your socket> = X'0100050106'
```

The process at UCLA is unblocked and notified of the successful Init. Because the SRI logger always initiates a connection to the AEN one less than it has just been connected to, the UCLA process then executes

```
        Listen      <port> = 11
                    <AEN 1> = 6
```

and when unblocked,

Accept <AEN 2> = 6.

When these transactions are complete, the UCLA process is doubly connected to the logger at SRI. The logger will then interrogate the UCLA process, and if satisfied, create a new process at SRI. This new process will be tagged with the user number X'010005', and both connections will be switched to the new process. In this case, switching the connections to the new process corresponds to "passing the console down" in many time-sharing systems.

USER LEVEL SOFTWARE

At the user level, subroutines which manage data buffers and format input destined for other HOSTs are provided. It is not mandatory that the user use such subroutines, since the user has access to the network system calls in his monitor.

In addition to user programming access, it is desirable to have a subsystem program at each HOST which makes the network immediately accessible from a teletype-like device without special programming. Subsystems are commonly used system components such as text editors, compilers and interpreters. An example of a network-related subsystem is TELNET, which will allow users at the University of Utah to connect to Stanford Research Institute and appear as regular terminal users. It is expected that more sophisticated subsystems will be developed in time, but this basic one will render the early network immediately useful.

A user at the University of Utah (UTAH) is sitting at a teletype dialed into the University's PDP-10/50 time-sharing system. He wishes to operate the Conversational Algebraic Language (CAL) subsystem on the XDS-940 at Stanford Research Institute (SRI) in Menlo Park, California. A typical TELNET dialog is illustrated in Figure 5. The meaning of each line of dialog is discussed here.

(i) The user signs in at UTAH

(ii) The PDP-10 run command starts up the TELNET subsystem at the user's HOST.

(iii) The user identifies a break character which causes any message following the break to be interpreted locally rather than being sent on to the foreign HOST.

(iv) The TELNET subsystem will make the appropriate system calls to establish a pair of connections to the SRI logger. The connections will be established only if SRI accepts another foreign user.

The UTAH user is now in the pre-logged-in state at SRI. This is analogous to the standard teletype user's state after dialing into a computer and making a connection but before typing anything.

(v) The user signs in to SRI with a standard login command. Characters typed on the user's teletype are transmitted unaltered through the PDP-10 (user HOST) and on to the 940 (serving HOST). The PDP-10 TELNET subsystem will have automatically switched to full-duplex, character-by-character transmission, since this is required by SRI's 940. Full duplex operation is allowed for by the PDP-10, though not used by most Digital Equipment Corporation subsystems.

(vi) and (vii) The 940 subsystem, CAL, is started.

At this point, the user wishes to load a CAL file into the 940 CAL subsystem from the file system on his local PDP-10.

(viii) CAL is instructed to establish a connection to UTAH in order to receive the file. "NETWRK" is a predefined 940 name similar in nature to "PAPER TAPE" or "TELETYPE".

(ix) Finally, the user types the break character (#) followed by a command to his PDP-10 TELNET program, which sends the desired file to SRI from Utah on the connection just established for this purpose. The user's next statement is in CAL again.

The TELNET subsystem coding should be minimal for it is essentially a shell program built over the network system calls. It effectively established a shunt in the user HOST between the remote user and a distant serving HOST.

Given the basic system primitives, the TELNET subsystem at the user HOST and a manual for the serving HOST, the network can be profitably employed by remote users today.

HIGHER LEVEL PROTOCOL

The network poses special problems where a high degree of interaction is required between the user and a particular subsystem on a foreign HOST. These problems arise due to heterogeneous consoles, local operating system overhead, and network transmission delays. Unless we use special strategies it may be difficult or even impossible for a distant user to make use of the more sophisticated subsystems offered. While these difficulties are especially severe in the area of graphics, problems may arise even for teletype interaction. For example, suppose that a foreign subsystem is designed for teletype consoles connected by telephone, and then this subsystem becomes available to network users. This subsystem might have the following characteristics.

1. Except for echoing and correction of mistyping, no action is taken until a carriage return in typed.
2. All characters except "↑", "←" and carriage return are echoed as the character typed.
3. ← causes deletion of the immediately preceding accepted character, and is echosed as that character.
4. ↑ causes all previously typed characters to be ignored. A carriage return and line feed are echoed.
5. A carriage return is echoed as a carriage return followed by a line feed.

If each character typed is sent in its own message, then the characters

H E L L O ← ← P c.r.

cause nine messages in each direction. Furthermore, each character is handled by a user level program in the local HOST before being sent to the foreign HOST.

Now it is clear that if this particular example were important, we would quickly implement rules 1 to 5 in a local HOST program and send only complete lines to the foreign HOST. If the foreign HOST program could not be modified so as to not generate echoes, then the local program could not only echo properly, it could also throw away the later echoes from the foreign HOST. However, the problem is not any particular interaction scheme; the problem is that we expect many of these kinds of schemes to occur. We have not found any general solutions to these problems, but some observations and conjectures may lead the way.

With respect to heterogeneous consoles, we note that although consoles are rarely compatible, many are equivalent. It is probably reasonable to treat a model 37 teletype as the equivalent of an IBM 2741. Similarly, most storage scopes will form an equivalence class, and most refresh display scopes will form another. Furthermore, a hierarchy might emerge with members of one class usable in place of those in another, but not vice versa. We can imagine that any scope might be an adequate substitute for a teletype, but hardly the reverse. This observation leads us to wonder if a network-wide language for consoles might be possible. Such a language would provide for distinct treatment of different classes of consoles, with semantics appropriate to each class. Each site could then write interface programs for its consoles to make them look like network standard devices.

Another observation is that a user evaluates an interactive system by comparing the speed of the system's responses with his own expectations. Sometimes a user feels that he has made only a minor request, so the response should be immediate; at other times he feels he has made a substantial request, and is therefore willing to wait for the response. Some interactive subsystems are especially pleasant to use because a great deal of work has gone into tailoring the responses to the user's expectations. In the network, however, a local user level process intervenes between a local console and a foreign subsystem, and we may expect the response time for minor requests to degrade. Now it may happen that all of this tailoring of the interaction is fairly independent of the portion of the subsystem which does the heavy computing or I/O. In such a case, it may be possible to separate a subsystem into two sections. One section would be the "substantive" portion; the other would be a "front end" which formats output to the user, accepts his inputs, and controls computationally simple responses such as echoes. In the example above, the program to accumulate a line and generate echoes would be the front end of some subsystem. We now take notice of the fact that the local HOSTs have substantial computational power, but our current designs make use of the local HOST only as a data concentrator. This is somewhat ironic, for the local HOST is not only poorly utilized as a data concentrator, it also degrades performance because of the delays it introduces.

These arguments have led us to consider the possibility of a Network Interface Language (NIL) which would be a network-wide language for writing the front end of interactive subsystems. This language would have the feature that subprograms communicate through network-like connections. The strategy is then to transport the source code for the front end of a subsystem to the local HOST, where it would be compiled and executed.

During preliminary discussions we have agreed that NIL should have at least the following semantic properties not generally found in languages.

1. Concurrency. Because messages arrive asynchronously on different connections, and because user input is not synchronized with subsystem output, NIL must include semantics to accurately model the possible concurrencies.
2. Program Concatenation. It is very useful to be able to insert a program in between two other programs. To achieve this, the interconnection of programs would be specified at run time and would not be implicit in the source code.
3. Device substitutability. It is usual to define languages so that one device may be substituted for another. The requirement here is that any device can be modelled by a NIL program. For example, if a network standard display controller manipulates tree-structures according to messages sent to it then these structures must be easily implementable in NIL.

NIL has not been fully specified, and reservations have been expressed about its usefulness. These reservations hinge upon our conjecture that it is possible to divide an interactive subsystem into a transportable front end which satisfies a user's expectations at low cost and a more substantial stay-at-home section. If our conjecture is false, then NIL will not be useful; otherwise it seems worth pursuing. Testing of this conjecture and further development of NIL will take priority after low level HOST-HOST protocol has stabilized.

HOST/IMP INTERFACING

The hardware and software interfaces between HOST and IMP is an area of particular concern to the HOST organizations. Considering the diversity of HOST computers to which a standard IMP must connect, the hardware interface was made bit serial and full-duplex. Each HOST organization implements its half of this very simple interface.

The software interface is equally simple and consists of messages passed back and forth between the IMP and HOST programs. Special error and signal messages are defined as well as messages containing normal data. Messages waiting in queues in either machine are sent at the pleasure of the machine in which they reside with no concern for the needs of the other computer.

The effect of the present software interface is the needless rebuffering of all messages in the HOST in addition to the buffering in the IMP. The messages have no particular order other than arrival times at the IMP. The Network Control Program at one HOST (e.g., Utah) needs waiting RFNM's before all other messages. At another site (e.g., SRI), the NCP could benefit by receiving messages for the user who is next to be run.

What is needed is coding representing the specific needs of the HOST on both sides of the interface to make intelligent decisions about what to transmit next over the channel. With the present software interface, the channel in one direction once committed to a particular message is then locked up for up to 80 milliseconds! This approaches one teletype character time and needlessly limits full-duplex, character by character, interaction over the net. At the very least, the IMP/HOST protocol should be expended to permit each side to assist the other in scheduling messages over the channels.

CONCLUSIONS

At this time (February 1970) the initial network of four sites is just beginning ~~to beginning~~ to be utilized. The communications system of four IMPs and wide band telephone lines have been operational for two months. Programmers at UCLA have signed in as users of the SRI 940. More significantly, one of the authors (S. Carr) living in Palo Alto uses the Salt Lake PDP-10 on a daily basis by first connecting to SRI. We thus have first hand experience that remote interaction is possible and is highly effective.

Work on the ARPA network has generated new areas of interest. NIL is one example, and interprocess communication is another. Interprocess communication over the network is a subcase of general interprocess communication in a multiprogrammed environment. The mechanism of connections seems to be new, and we wonder whether this mechanism is useful even when the processes are within the same computer.

REFERENCES

1 L ROBERTS

<u>The ARPA network</u>

Invitational Workshop on Networks of Computers Proceedings

National Security Agency 1968 p 115 ff

2 R M RUTLEDGE et al

<u>An interactive network of time-sharing computers</u>

Proceedings of the 24th National Conference

Association for Computing Machinery 1969 p 431 ff

3 F E HEART R E KAHN S M ORNSTEIN W R CROWTHER D C WALDEN

<u>The interface message processors for the ARPA network</u>

These Proceedings

LIST OF FIGURES

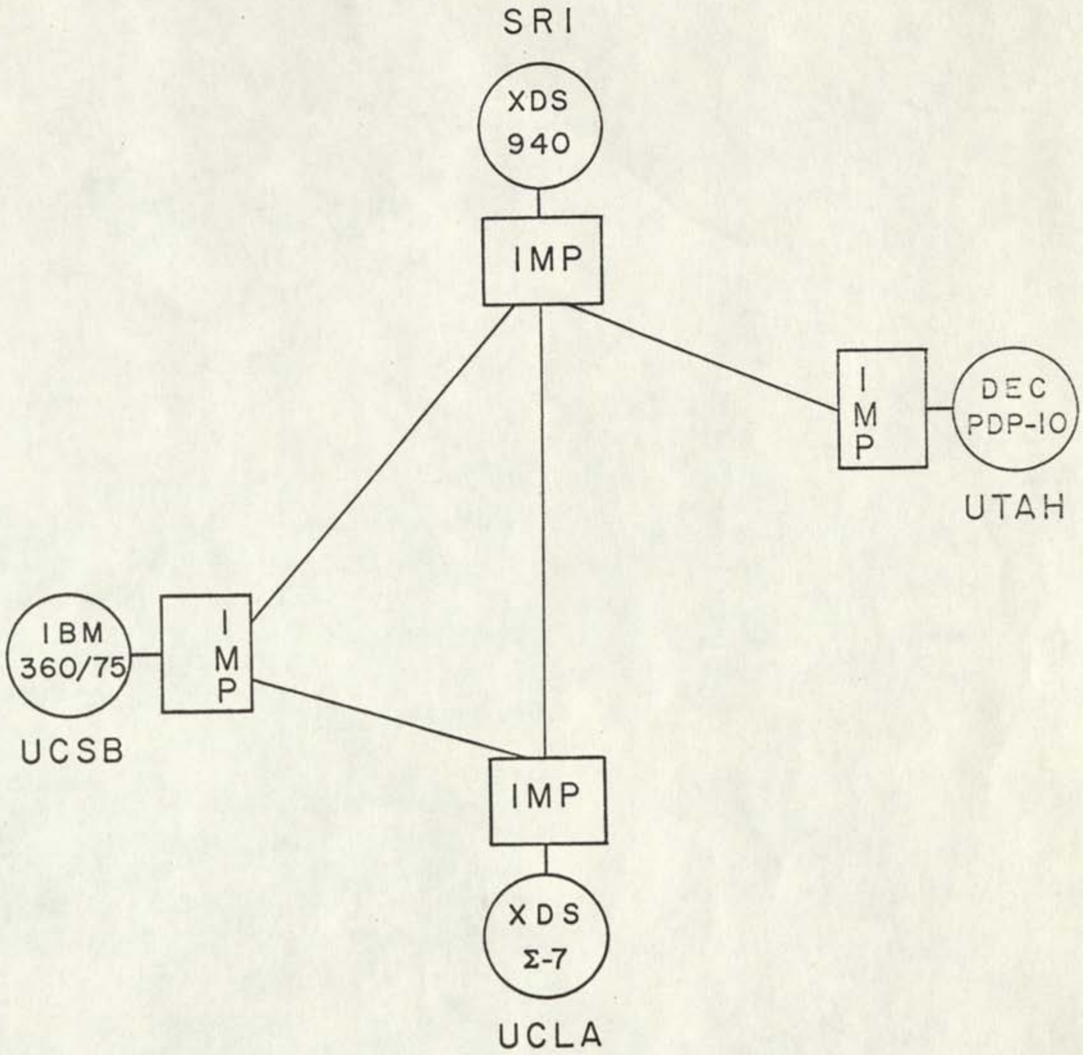

Figure 1 Initial network configuration

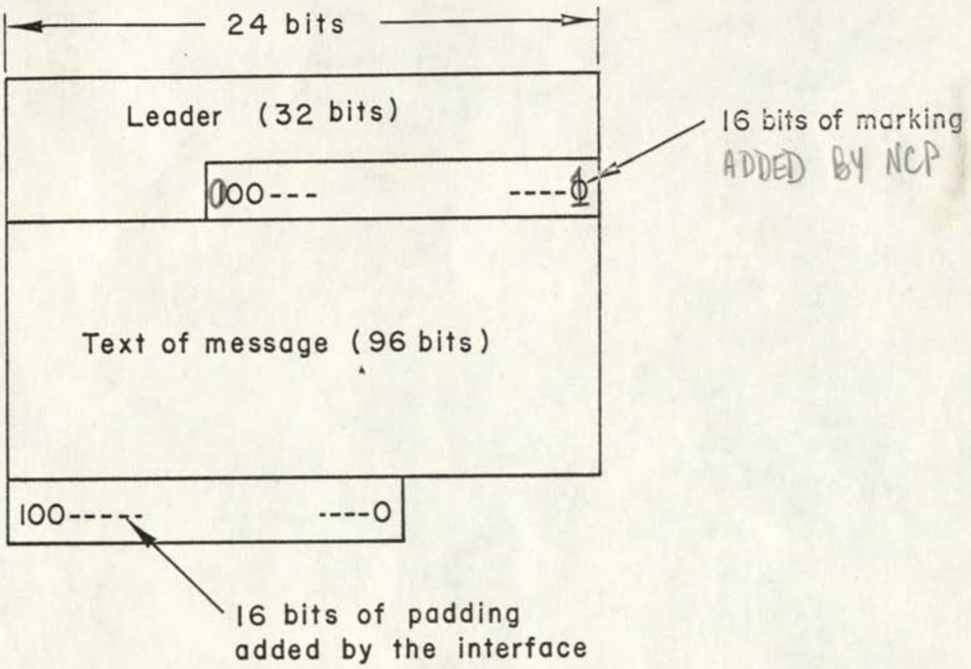

Figure 2 A typical message from a 24-bit machine

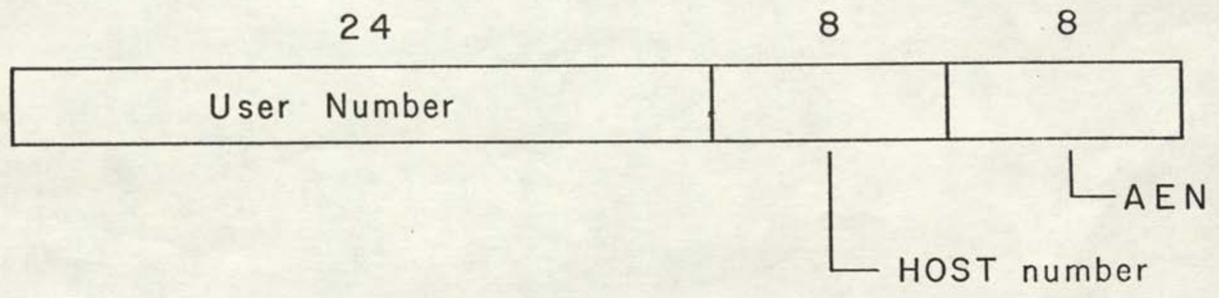

Figure 3 A typical socket

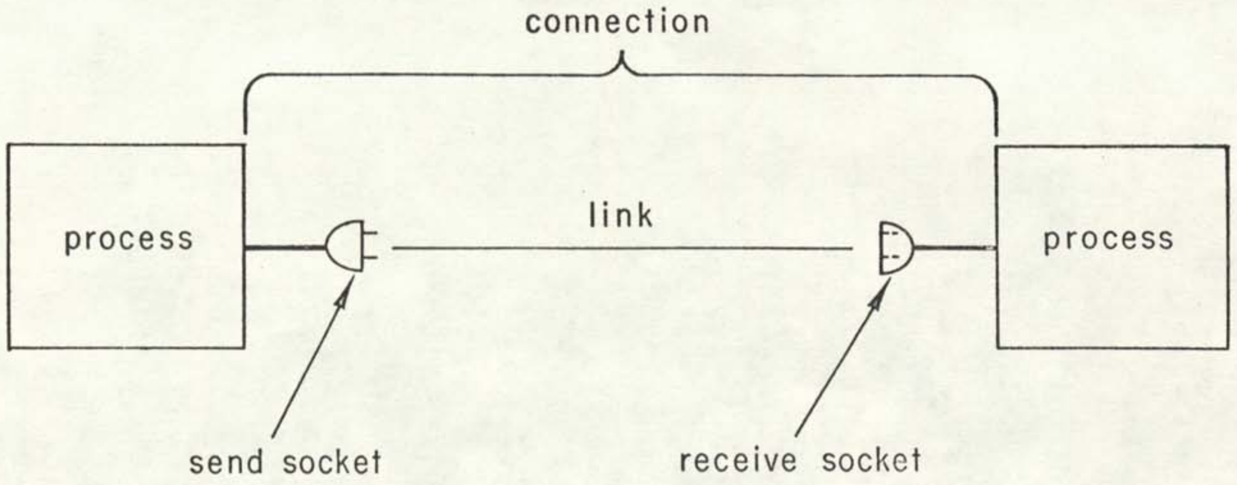

Figure 4 The relationship between sockets and processes

```
(i)     .LOGINⓒⓡ

(ii)    .R TELNETⓒⓡ

(iii)   ESCAPE CHARACTER IS #ⓒⓡ

(iv)    CONNECT TO SRIⓒⓡ

(v)     @ENTER CARR.ⓒⓡ

(vi)    @CAL.ⓒⓡ

(vii)   CAL AT YOUR SERVICEⓒⓡ

(viii)  >READ FILE FROM NETWRK.ⓒⓡ

(ix)    #NETWRK:←DSK:MYFILE.CALⓒⓡ
```

Figure 5 A typical TELNET dialog
Underlined characters are those typed by the user

4736

Network Working Group
Request for Comments No. 34

Bill English   SRI/ARC
26 February 1970

SOME BRIEF PRELIMINARY NOTES ON THE ARC CLOCK

The ARC clock system provides a time reference that is written into the core memory of the XDS 940 Computer. There are two types of time information available--absolute and relative.

The absolute time is written into two adjacent words of core with the following format:

First Word --

Bits 0 thru 7 contain the month code in straight binary with a range of 1 to 12

Bits 8 thru 15 contain the day code in straight binary with a range of 1 to 31

Bits 16 thru 23 contain the year code in straight binary with a range of 0 to 99

Second Word --

Bits 00 thru 7 contain the hour code written in straight binary with a range of 0 to 23

Bits 8 thru 15 contain the minute code written in straight binary with a range of 0 to 60

Bits 16 thru 23 contain the second code written in straight binary with a range of 0 to 60

These 2 words are written once each second. It is antcipated that the accuracy of the initial setting will be on the order of 1 second, as referred to WWV, and that the oscillator drift rate will not account for an accumulated error of more than 1 second every 250 days. The oscillator and clock are provided with standby power in order to maintain the accuracy of the system. Because of variable delays in the time required to obtain access to the 940 core memory, it is anticipated that the short-term accuracy will be on the order of 10 to 20 microseconds.

The relative time, which is written into one word of core memory, is simply the contents of a 24 bit binary accumulator. The rate at which the accumulator is updated can be chosen to be either once very 100 micro seconds or once every millisecond. In either case the core location is written each time the accumulator is updated. As above the short-term accuracy will be about 10 to 20 microseconds and the long-term accuracy will be the equivalent of one second every 250 days.

4737

Network Working Group
Request for Comments: 35

S. Crocker
UCLA
3 March 1970

NETWORK MEETING

I expect to have the details of the new network protocol as outlined in NWG/RFC #33 ready in two weeks. Some interest has been expressed in a live presentation, so we will host a network meeting on Tuesday, March 17, at UCLA at 9:00 a.m. To facilitate interaction, please limit attendance to one programmer from each site. It is also wise to leave the 18th open in case discussion continues.

The subject of the meeting will be a detailed presentation of the network protocol, suitable for implementing unless major flaws are discovered. Documentation will be available at the meeting and if not obsoleted by the meeting, will be sent out as a NWG/RFC on March 20.

Please call Mrs. Charlotte LaRoche at (213) 825-2543 if you need help in making arrangements.

S. Crocker
16 March 70

4738

Protocol Notes

I Overview

The network protocol provides three facilities:

1. Connection establishment

2. Flow control

3. Reconnection

Reconnection is considered separately from connection establishment partly because of the complexity of reconnection and partly because I don't have enough experience with the protocol to present these concepts in an integrated fashion.

Connection Establishment

Connection establishment works essentially the same as in NWG/RFC #33. The major change is that a more general form of switching is provided indepently of establishment, so establishment is simplified by not including switching procedures.

A rough scenario for connection establishment follows:

1. Process PA in host A grabs socket SA and requests connection with socket SB. Process PA accomplishes this through a system call.

2. Concurrently with the above, process PB in host B grabs socket SB and requests connection with socket SA.

3. In response to process PA's request, the network control program in host A (referred to as NCPA) sends a Request-for-Connection (RFC) command to host B. NCPB in host B sends a similar command to host A. No ordering is implied; NCPB may send the command to NCPA before or after receiving the command from NCPA.

4. NCPA and NCPB are both aware the connection is established when each has received a RFC command and each has received the RFNM for the one it has sent. They then notify processes PA and PB, respectively, that the connection is established.

One of the rules adhered to is that either SA is a send socket and SB is a receive socket, or vice versa. This condition is sometimes stated as "SA and SB must be a send/receive pair."

5. The sending process may now send.

Flow Control

In order to prevent a sending process from flooding a receiving process, it is necessary for the receiving process to be able to stop the flow.* Flow control is integrated into the network RFNM handling. When a receiving host wishes to inhibit flow on a particular link, the host sends a special message to its IMP which causes the next RFNM on that link to be modified. The sending host receives a message of type 10 instead of type 5. The sending host interprets this message as a RFNM and as a request to stop sending. A confirming control command is returned.

When the receiving host is ready to receive again, it sends a command (RSM) telling the sending host to resume sending.

Reconnection

For a great many reasons it is desirable to be able to switch one (or both) ends of a connection from one socket to another. Depending upon the restrictions placed upon the switching process, it may be easy or hard to implement. To achieve maximum generality, I present here a scheme for dynamic reconnection, which means that reconnection can take place even after flow has started. It may turn out that for the majority of cases, this scheme is much more expensive than it needs to be; however, the following virtues are claimed:

1. All various forms of switching connections are provided.

2. Reconnection introduces no overhead in the processing of messages sent over a connection i.e., the whole cost is borne in processing the protocol.

II Data Structures

1. Connection Table
2. Process Table
3. Input Link Table
4. Output Link Table
5. Link Assignment Table

---

*BB&N argues that unlimited buffering should be provided. It is possible that this would be a proper strategy; but it is foreign to my way of thinking, and I have based the protocol design on the assumption that only a small buffer is provided on the receive end of each connection.

Connection Table

This table holds all information pertaining to local sockets, particularly whether a socket is engaged in a connection, and if so, what state the connection is in. Entries are keyed by local socket, but other tables have pointers into this table also. (See the Process Table, Input Link Table, and Output Link Table.)

Each entry contains the following information:

a) local socket (key)
b) foreign socket
c) link
d) connection state
e) flow state and buffer control
f) pointer to user's process
g) reconnection control state
h) queue of waiting callers

The local socket is a 32 bit number. If no entry exists for a particular socket, it may be created with null values.

The foreign socket is a 40 bit number. This field will be unassigned if no connection is established.

The link is an 8 bit number and is the link over which data is sent from the sender to the receiver. A socket is a receive socket off its low-order bit is zero.

Connection state refers to whether a connection is open or not, etc. The following possibilities may occur.

a) local process has requested a connection
b) foreign process(es) has/have requested a connection
c) connection established
d) reconnection in progress
e) close waiting
f) reconnection waiting

Flow state and buffer control refer to checking for RFNM's, sending and accepting cease, supended and resume commands, and keeping track of incoming or outgoing data.

A pointer to the user's process is necessary if the process has requested a connection.

If reconnection is in progress, it is necessary to keep track of the sequence of events. A socket engaged in reconnection is either an end or a middle. If it's a middle, it is necessary to store the eight bit name of the other middle attached to the same process, and to record receipt of END and RDY commands.

Finally, if RFC's are received either when the socket is busy or when no process has engaged it, the RFC's are stacked first-in-first-out on a queue for the named local socket.

Process Table

This table associates a process with a socket. It is used to process system calls.

Input Link Table

This table associates receive links with local sockets. It is used to decide for whom incoming messages are destined.

Output Link Table

This table associates send links with local sockets. It is used to interpret RFNM's and RSM commands.

Link Assignment Table

Links are assigned by receivers. This table shows which links are free.

III Control Commands

Command Summary

0	<NOP>
1	<RFC> <me> <you> or <RFC> <me> <you> <link>
2	<CLS> <me> <you>
3	<RSM> <link>
4	<SPD> <link>
5	<FND> <me> <you> <asker>
6	<END> <link> <end>
7	<RDY> <link>
8	<ASG> <me> <you> <link>

Commands

No Operation

Form: NOP

NOP is X'00'

Purpose: This command is included for completeness and convenience.

Request for connection

Form: <RFC> <my socket> <your socket>
or <RFC> <my socket> <your socket> <link>

<RFC> is X'01'

<my socket> is a 32 bit socket number local to the sender

<your socket> is a 32 bit socket number local to the receiver

<link> is an eight bit link number.

<my socket> and your socket must be a send/receive pair.

<link> is included if and only if <my socket> is a receive socket

Purpose: This command is used to initiate a connection. When two hosts have exchanged RFC commands with the same arguments (reversed), the connection is established. Links are assigned by the receiver.

Close

Form: <CLS> <my socket> <your socket>

<CLS> is X'02'

<my socket> and <your socket> are the same as for <RFC>

Purpose: This command is used to block a connection. It may also be used to abort the establishment of a connection or to refuse a request. It may happen that no connection between the named sockets was established, or was in the process of being established. In this event, the <CLS> should be discarded.

Resume

Form: <RSM> <link>

<RSM> is X'03'

Purpose: This command is sent by a receiving host to cause the sending host to resume transmission on the named link. A sending host suspends sending if it receives a special RFNM for some message. (Special RFNM's are generated by the receiving IMP upon request by its host.)

Suspended

Form: <SPD> <link>

<SPD> is X'04'

Purpose: This command is sent by a sending host to acknowledge that it has stopped sending over the named link. Transmission will resume if a <RSM> command is received.

Final End

Form: <FND> <my socket> <your socket> <asker>

<FND> is X'05'

<my socket> is a 32 bit socket number of a socket local to the sender

<your socket> is a 32 bit socket number of a socket local to the receiver

<my socket> and <your socket> form a send/receive pair. A connection should be established between them.

<asker> is a 40 bit socket number of the same type as <my socket>

Purpose: If a process decides to short-circuit itself by connecting one of its receive sockets to one of its send sockets, the NCP sends out two <FND> commands -- one in each direction. Each one has <asker> initialized to <my socket> .

Upon receiving an <FND> command, the NCP checks its <your socket>. If <your socket> is already engaged in a reconnection, the command is passed on with a new <my socket> and <your socket>. However, before it is passed on, the <asker> is compared with the new <my socket>. If they are equal, a loop has been detected and both sockets are closed.

If <your socket> is not engaged in a reconnection, it is marked as the end of a chain of reconnections end an <END> is sent back.

If the connection named is not in progress, a <CLS> is sent back and the <FND> is discarded.

End Found

Form: <END> <link> <end socket>

<END> is X'06'
<link> is an 8 bit link

<end socket> is a 40 bit socket

Purpose: This command indicates which socket is at the end of a chain of reconnections. It is generated at <end socket> and passed back to the other terminal socket via all the intermediate sockets. If <end socket> is a send socket, <link> refers to a connection with the send socket in the sending host and receive socket in the receiving host. If <end socket> is a receive socket, <link> refers to a connection with the send socket in the receiving host and the receive socket in the sending host. ("sending" end "receiving" refer to the transmission of this control command.)

Ready

Form: <RDY> <link>

<RDY> is X'07
<link> is an 8 bit link number

Purpose: This command is sent from a send socket to a receive socket to indicate that all messages have been forwarded and that reconnection may occur.

Assign New Link

Form: <ASG> <my socket> <your socket> <link>

<ASG> is X'08'

Purpose: This command completes a reconnection. It is sent from a receive socket to a send socket after the receive socket has received a <RDY>. A new link is assigned and transmission commences.

4739

Network Working Group
Request for Comments #37

S. Crocker
20 March 70
UCLA

Network Meeting Epilogue, etc.

The Meeting

On Tuesday, March 17, 1970, I hosted a Network meeting at UCLA. About 25 people attended, including representatives from MIT, LL, BBN, Harvard, SRI, Utah, UCSB, SDC, RAND and UCLA. I presented a modification of the protocol in NWG/RFC #33; the modifications are sketchily documented in NWG/RFC #36. The main modification is the facility for dynamic reconnection.

The protocol based on sockets and undistinguished simplex connections is quite different from the previous protocol as documented in NWG/RFC #11. The impetus for making such changes came out of the network meeting on December 8, 1969, at Utah, at which time the limitations of a log-in requirement and the inability to connect arbitrary processes was seriously challenged. Accordingly, the primary reason for the recent meeting was to sample opinion on the new protocol.

Recollections may vary, but it is my opinion that the protocol was widely accepted and that criticism and discussion fell into two categories:

1. Questioning the complexity and usefulness of the full protocol, especially the need for dynamic reconnection.

2. Other topics, particularly character set translation, higher level languages, incompatible equipment, etc.

Notably lacking was any criticism of the basic concepts of sockets and connections. (Some have since surfaced.) The following agreements were made:

1. By april 30, I would be responsible for publishing an implementable specification along the lines presented.

2. Any interested party would communicate with me (at least) immediately if he wished to modify the protocol.

3. If major modifications come under consideration, interested parties would meet again. This would happen in two to three weeks.

4. Jim Forgie of Lincoln Labs tentively agreed to host a meeting on higher level network languages, probably near Spring Joint time.

Network Working Group
Request for Comments #37

S. Crocker
20 March 70
UCLA

Mailing List Changes

Paul Rovner of LL is replaced by

James Forgie
Mass. Institute of Technology
Lincoln Laboratory C158
P.O. Box 73
Lexington, Mass. 02173

telephone at (617) 862-5500 ext. 7173

Professor George Mealy is added

George Mealy
Rm. 220
Aitken Computation Lab.
Harvard University
Cambridge, Massachusetts 02138

telephone at (617) 868-1020 ext. 4355

Processes

In all of our writing we have used the term process, to mean a program which has an assigned location counter and an address space. A program is merely a pattern of bits stored in some file. A new process is created only by an already existing process. The previous process must execute an atomic operation (fork, attach, etc.) to cause such a creation. Processes may either cause their own demise or be terminated by another (usually superior) process.

The above definition corresponds to the definition given by Vyssotsky, et al on pp. 206, 207 of "Structure of the Multics Supervisor" in the FJCC proceedings, 1965.

Because a process may create another process, and because in general the two processes are indistinguishable when viewed externally, I know of no reasonable way for two processes to request connection directly with each other. The function of sockets is to provide a standard interface between processes.

The Days After

In the time since the meeting I have had conversations with Steve Wolfe (UCLA-CCN), Bill Crowther (BBN), and John Heafner and Erick Harslem (RAND). Wolf's comments will appear as NWG/RFC #38 and fall into a class I will comment on below.

Network Working Group
Request for Comments #37

S. Crocker
20 March 70
UCLA

Crowther submitted the following:

"A brief description of two ideas for simplifying the host protocol described at the March meeting. These ideas have not been carefully worked out.

Idea 1. To Reconnect.

"A NCP wanting to Reconnect tells each of this neighbors "I want to reconnect". They wait until there are no messages in transit and respond "OK". He then says "Reconnect as follows" and they do it. In the Rare condition, the NCP gets back an "I want to reconnect instead of an "OK", then one must go and one must stop. So treat a "reconnect" from a higher Host user etc. as ok and from a lower as a "No-wait until I reconnect you" and do the connection.

Idea 2

"Decouple connections and links. Still establish connections, but use any handy link for the messages. Don't send another message on a connection until a RFNM comes back. Include source and destination socket numbers in the packet.

"To reconnect, say to each of neighbors "please reconnect me as follows...". Hold onto the connection for a short time (seconds) and send both packets and connection messages along toward their desintations. I havn't worked out how to keep the in-transit messages in order, but probably everything works if you don't send out a reconnect when RFNM's are pending."

Bill's first idea does not seem to me to be either decisively better or (after some thought) very different, and I am considering it. I have no strong feelings about it yet, but I am trying to develop some.

Bill's second idea seems contrary to my conception of the role of links. An argument in favor of decoupling connections and links that the number of connections between two hosts might want to exceed 255, and that even if not, it is sounder practice to isolate dependencies in design. On the other hand, the newly provided Cease on Link facility* (page 22 of the soon to be released BBN report #1822 revised February 1970) becomes useless. (Bill, who just put the feature in, doesn't care.) Another objection is that it seems intuitively bad to waste the possibility of using the link field to carry information. (Note the conflict of gut level feelings)

*The Cease on Link facility is a way a receiving host modifies RFNM's so as to carry a flow-quenching meaning. An alternative procedure is to use a host-to-host control command.

S. Crocker
20 March 70
UCLA

In a conversation with John Heafner and Eric Harslem of RAND, they pointed out that the current protocol makes no provision for error detection and reporting, status testing and reporting, and expansion and experimentation. Error detection and status testing will require some extended discussion to see what is useful, and I expect that such discussion will take place while implementation proceeds. Leaving room for protocol expansion and experimentation, however, is best done now.

I suggest that two areas for expansion be reserved. One is that only a fraction of the 256 links he used, say the first 32. The other area is to use command codes from 255 downward, with permanent codes assigned from the number of links in use to 32, I feel that it is quite unlikely that we would need more than 32 for quite some time, and moreover, the network probably wouldn't handle traffic implied by heavy link assignemnt. (These two things aren't necessarily strongly coupled: one can have many links assigned but only a few carrying traffic at nay given time.)

Some of Heafner's and Harslen's other ideas may appear in NWG/RFC form.

Immediate Interaction

During the next several days, I will still be interested in those editicisms of the current protocol which might lead to rejection or serious modification of it. Thereafter, the gocus will be a refinement, implementation, extension, and utilization. I may be reached at UCLA through my secretary Mrs. Benita Kirstel at (213) 825-2368. Also, everyone is invited to contribute to the NWG/RFC series. Unique numbers are assigned by Benita.

NETWORK WORKING GROUP
REQUEST FOR COMMENTS #38

STEPHEN M. WOLFE
20 MARCH 1970
UCLA CCN

Comments on Network Protocol from NWG/RFC #36

The proposed protocol does not allow for the possible multiplexing of connections over links.

Generally, this presents no problem, but it might cause loading restrictions in the future. Two cases where routing multiple connections over the same link are apparent:

a) Where a user has several high speed connections, such as between processes that transmit files over the network. Assigning these connections to the same link limits the percentage of network resources that may be used by that user. This becomes particularly important when several store-and-forward IMP's are used by the network to affect the communication.

b) When two hosts each have their own independent network and desire to allow access to the other host's network over the ARPA net, a shortage of links may develop. Again, the assignment of several connections to the same link could help solve the problem.

The following changes in the protocol would make possible the future use of multiplexed links. It is not necessary to add the multiplexing, itself, to the protocol at this time.

a) The END and RDY must specify relevant sockets in addition to the link number. Only the local socket name need be supplied.

b) Problems arise with the RSM and SPD commands. Should they refer to an entire link, or just to a given connection? Since there is a proposal to modify the RFNM to accommodate these commands, it might be better to add another set of commands to block and unblock a connection, but I am not convinced that that is the best solution.

c) The destination socket must be added to the header of each message on the data link. Presumably this would consist of 32 bits immediately after the header and before the marking.

SMW/rb

4741

E. Harslem
J. Heafner
RAND

Network Working Group
Request for Comments: 39
25 March 1970

COMMENTS ON PROTOCOL RE: NWG/RFC #36

We offer the following suggestions to be considered as additions to the April 28th 1970 protocol grammar specifications.

ERROR MESSAGES

    <ERR> <Code> <Command in error>

It is desirable to include debugging aids in the initial protocol for checking out Network Control Programs, etc.

There are three classes of errors--content errors, status errors, and resource allocation or exhaustion. <Code> specifies the class and the offending member of the class. The command is returned to the sending NCP for identification and analysis.

Examples of status errors are: messages sent over blocked links and attempts to unblock an unblocked link. Examples of content errors are: an invalid RFC complete; a message sent on a link not connected; closing of an unconnected link; and an attempt to unblock an unconnected link. Examples of resource errors are: a request for a non-existent program and connection table overflow, etc. Resource errors should be followed by a <CLS> in response to the <RFC>.

QUERIES

```
     <QRY> <My   Socket> < >
or   <QRY> <Your Socket> <Text>
```

Queries provide an extension to the <ERR> facility as well as limited error recovery, thus avoiding re-initialization of an NCP.

The first command requests the remote NCP to supply the status of all connections to the user specified by the user number in <My Socket>. The second is the reply; <Text> contains the connection status information. If an NCP wants the status of all connections to a remote HOST, the <My Socket> is zero.

PROGRAM TERMINATION NOTIFICATION

<TER> <My Socket>

This command supplements rather than replaces <CLS>. It severs all communication between a program and those programs in a given HOST to which it is connected. This command performs what would otherwise be handled by multiple <CLS> commands. <My Socket> contains the sender's user number.

HOST STATUS

<HCU>
<HGD>

These messages (HOST coming up and HOST voluntarily going down) are compatible with asynchronous, interrupt-driven programs, as opposed to the more conventional post/poll method.

TRANSMIT AND BROADCAST

<TRN> <Body>
<BDC> <Body>

Unlike the previous commands, these are not sent over the control link, but rather over links assigned to user programs. The prefix of <TRN> or <BDC> indicates, to the receiving NCP, the disposition of the message body. <TRN> indicates a message to be passed to a single process. <BDC> specifies to the destination NCP that the message is to be distributed over all receiving connections linked to the sender. In response to a system call by the user to an NCP requesting <BDC>, the NCP generates one <BDC> to each HOST to which the sender is connected.

RFC AND DYNAMIC RECONNECTION

This protocol is complex; it proliferates control messages; it causes queues (to become associated with re-entrant procedures) that are artificially imposed via the protocol (remote AEN assignment); and discounts the situation where only controlling process "A" has knowledge that slave process "B" should be "rung out" in a dynamic reconnection.

The <ERR>, etc., are suggestions for inclusion as additions in the April 28th protocol specifications. The above criticism is, of course, not intended to affect modification of the RFC structure by April 28th, nor to reflect on those who planned it. We have not studied the problem. It is meant, however, to voice our concern about complexity and resulting response times. This is a difficult problem and it deserves more study after we have exercised the current RFC specifications. We hope to offer constructive suggestions with respect to the RFC in the future.

JFH:hs

Distribution

1. Abhai Bhushan, MIT
2. Steve Carr, Utah
3. Gerry Cole, SDC
4. Steve Crocker, UCLA
5. Bill English, SRI
6. Jim Forgie, LL
7. Jim Fry, MITRE
8. Nico Haberman, Carnegie-Mellon
9. John Heafner, RAND
10. Bob Kahn, BB&N
11. George Mealy, Harvard
12. Thomas O'Sullivan, Raytheon
13. Larry Roberts, ARPA
14. Robert Sproull, Stanford
15. Ron Stoughton, UCSB