Host-to-Host Protocols

David C. Walden
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

PREFACE

This paper is based largely on my experience with the ARPA Network, whose most important single characteristic is that it switches packets rather than lines. Everything I shall say here about Host-to-Host Protocol assumes a packet-switching network, as I have given no thought to protocols for the line-switching approach.

A few words of acknowledgment are in order. This paper draws heavily on the thoughts and privately circulated writings of my colleagues at Bolt Beranek and Newman, including Bernie Cosell, Bill Crowther, Bob Bressler, Bob Thomas, and, in particular, Alex McKenzie. I also acknowledge the large number of people who struggled to design the Host-to-Host protocols for the ARPA Network. This number includes, although it is not limited to, the names listed at the end of [1] and, of course, the authors of that paper, particularly Steve Crocker of ARPA, who was the chairman of the Host-to-Host Protocol design committee. I would also like to acknowledge the assistance of Bob Brooks and Julie Moore in the preparation of the present paper.

1. INTRODUCTION

In my experience, with the ARPA Network [2,3], the construction of a computer network may be broken into three parts: the layout of the location of the nodes and the paths of the communication circuits; the design, construction and operation of a communication subnetwork; and the design, implementation, and use of a set of conventions for communication between Hosts. We call this set of conventions the Host-to-Host protocols or "protocols" for short. Of the three parts of network construction, the first two seem reasonably manageable as they can be assigned to single groups for the provision of coherent solutions. The protocol portion of network construction, however, seems very difficult to solve due to the large number of (perhaps incompatible) Host systems, opinions, etc., involved. For example, in the ARPA Network, the sponsoring Advanced Research Projects Agency (ARPA) decides the location of the nodes and the topology of the communications circuits (as shown in figures 1 and 2) with technical support for the latter effort from the Network Analysis Corporation. The responsibility for the design, implementation, and operation of the communications subnetwork (as shown by the black dots and encircled T's in figure 2) for the ARPA Network rests with Bolt Beranek and Newman Inc. [4,5,6], the prime contractors for the subnetwork. But in the ARPA Network, the Host-to-Host Protocol design was undertaken by a committee consisting of whomever was interested from all of the Host sites in the network. The rationale for design by open committee is, of course, that in a confederation of otherwise unrelated Hosts, every Host has its own point of view and its own problems, and must be given a chance to express them. I think it is the consensus about the ARPA Network that, because the Host-to-Host Protocol design was done by the committee and because each Host has its own set of

ARPA NETWORK, GEOGRAPHIC MAP, JUNE 1974

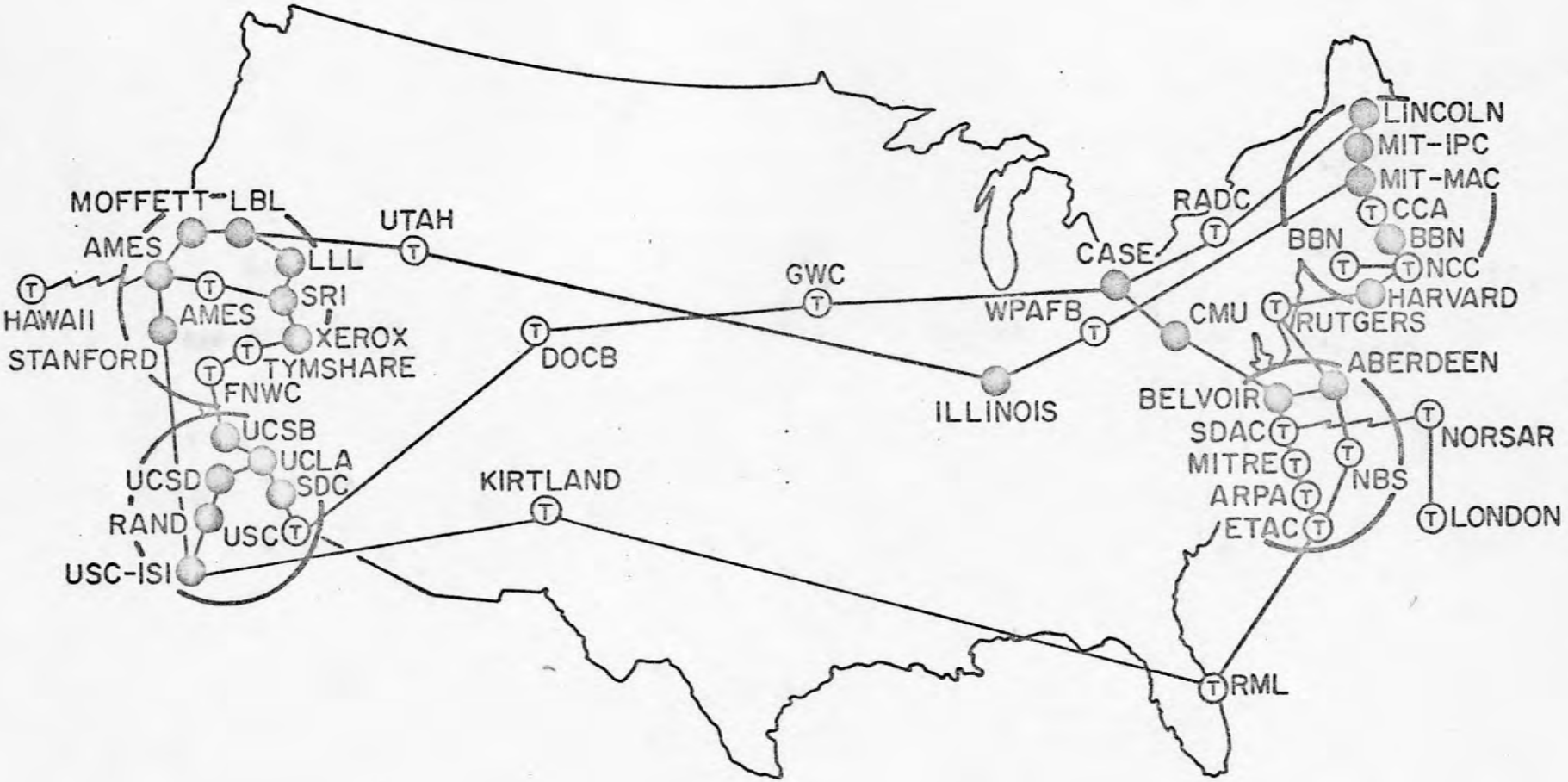

Figure 1

ARPA NETWORK, LOGICAL MAP, JUNE 1974

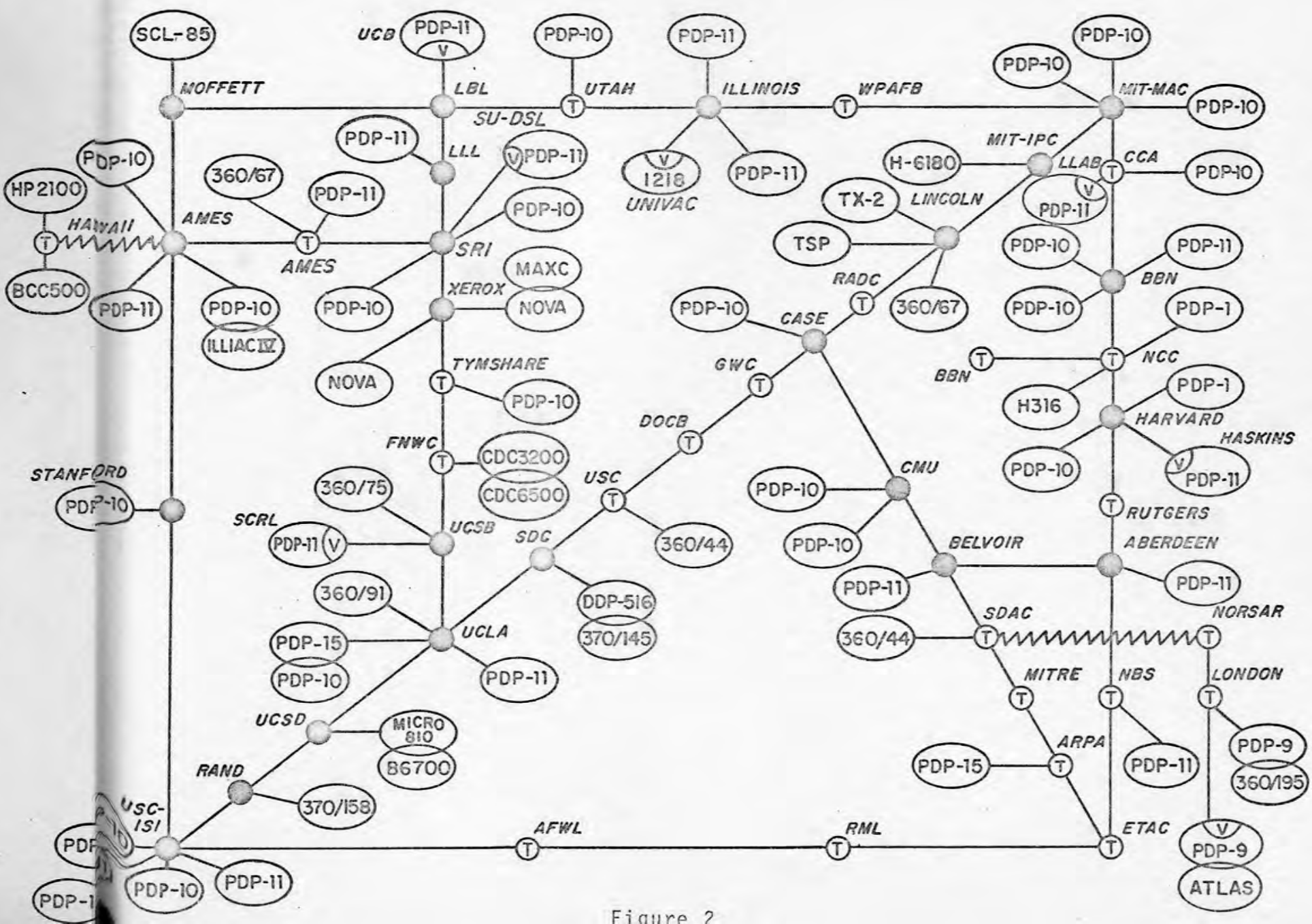

Figure 2

implementation problems, the protocol development effort has not gone particularly well — although a number of usable protocols have been developed and others are under development. The current ARPA Network protocols have been long in coming, difficult to implement, frequently inefficient, lacking in robustness, etc. Despite this, a large volume of data is being transmitted every day using these protocols. To emphasize the disparity that exists between the computers in the ARPA Network (many of which are shown in figure 2), the computers range in computational capacity from a PDP-1 to an IBM 360/91 to the ILLIAC IV, and in expanse of operating system from a tiny terminal concentrating computer to MULTICS.

In the remainder of this paper I will draw primarily on my experience with the ARPA Network protocols to illustrate the actual development of a set of protocols, the classes of network usage supported by these protocols, some difficulties with these protocols, and some protocol improvements and alternate protocol developments.

2. SUMMARY OF THE ARPA NETWORK PROTOCOLS

The ARPA Network protocols group nicely into six classes: 1) Host/IMP protocol, the lowest level of protocol with which Hosts must send and receive addressed messages; 2) various *ad hoc* protocols between pairs (or sometimes larger groups) of Hosts; 3) Host/Host protocol (to be distinguished from the title of this paper), a network wide, connection-based protocol which most Hosts in the ARPA Network have implemented and on which the below-mentioned protocols are based; 4) the Initial Connection Protocol, a standard protocol for "logging into" a network Host; 5) the Network Virtual Terminal and TELNET protocol, a specification of a network-standard, Teletype-like terminal and a protocol for communicating between a standard terminal and a Host; and, 6) a number of function-oriented, higher level protocols. The remainder of this section elaborates briefly on five of these six protocol classes; no further *brief* description is required for the remaining Initial Connection Protocol.

2.1 Host/IMP Protocol

As previously stated, the IMP/Host is the lowest level protocol [7], and Hosts must use it to send messages in the format shown in figure 3. Hosts send messages which are either IMP/Host control messages or messages for another Host as indicated by the type field. Messages to another Host have the destination specified (the source when they arrive) and are identified by the message-id. The messages can also have priority and have a few other characteristics. The message must contain at least the 32 bits of control information plus data up to 8095 bits total; there are no further specifications. In particular, an arbitrary binary stream may be sent. Actually the IMP/Host protocol also specifies the electrical connection between an IMP and Host. This electrical protocol has three main characteristics: it is asynchronous, avoiding timing mismatches; it is bit serial, avoiding word length mismatches; and it is pattern independent, so as not to limit message contents.

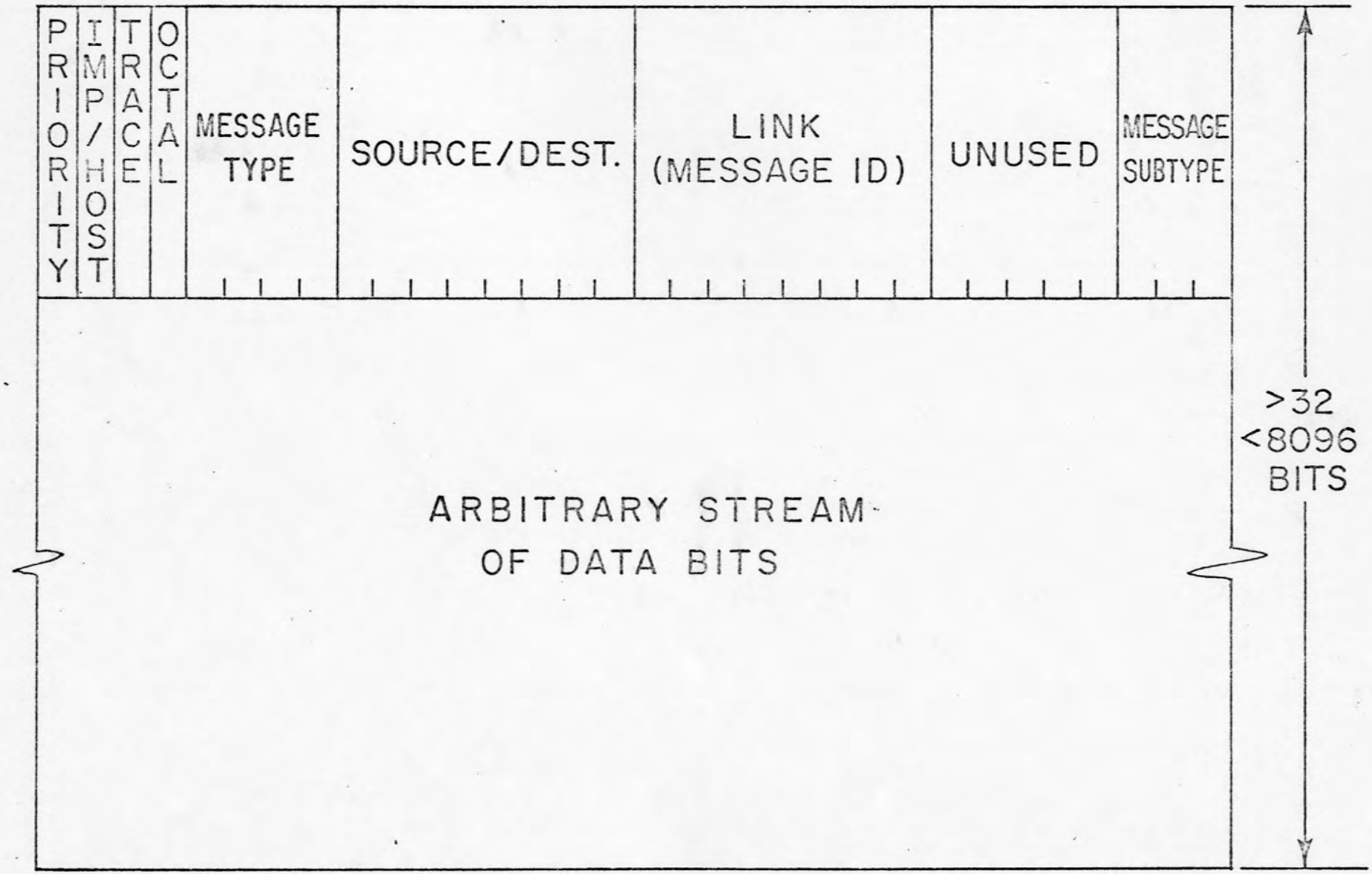
PRIORITY
IMP/HOST
TRACE
OCTAL
MESSAGE TYPE
SOURCE/DEST.
LINK (MESSAGE ID)
UNUSED
MESSAGE SUBTYPE
ARBITRARY STREAM OF DATA BITS
>32 <8096 BITS

Figure 3

2.2 Ad Hoc Protocols

Because the lowest level IMP/Host protocol puts so few constraints on message content, any number of ad hoc higher level protocols are possible. A few examples follow:

- COPYNET was an interim protocol for file transfer between PDP-10 TENEX systems in a 36-bit word format — it was discontinued with the "acceptance" of a general file transfer protocol.
- The UCLA 360/91 wanted customers and couldn't afford to wait for an official RJE protocol — so they invented their own interim RJE protocol.
- Similarly for the UCSB 360/75.
- Two air force bases experimented with message switching (vs. Autodin) high bandwidth traffic between 2 U418-III's. The experiment successfully concluded, the bases left the network.
- The TIP mag tape option hasn't the space available to implement the file transfer protocol, so uses its own convention for specifying records and end of files.
- The IMPs communicate among themselves and with the Network Control Center with a variety of formats; for example, messages containing IMP debugging commands, or binary lists of IMP parameters.
- Experiments are under way to transmit digitized speech using a special protocol.

- Alternate Host/Host protocols vying with the present Host/Host protocol for officialdom are tested through concurrent operation in the network.

By now the reader should have the point — virtually any ad hoc protocol is supported, and that's a good thing.

2.3 Host/Host Protocol

Figure 4 illustrates the ARPA Network Host/Host protocol [8]. Before two processes can communicate, a *connection* must be set up between a send socket in one process's monitor and a receive socket in the other process's monitor (a *socket* is an element in a network-wide name space into which each monitor maps its own internal name space). For two processes to make connection, each process makes a connection request to its own monitor. The two monitors then exchange these requests via messages over the *control link*, a special logical link between each pair of Hosts which is always reserved for control messages. If both monitors are in agreement, the connection is established and a link is assigned for use by the new connection. This new connection exists until it is explicitly terminated, again using inter-monitor control messages over the control link. During the "life" of the connection, many messages may be sent from the socket at one end to the socket at the other, and all these messages are sent over the link assigned to the connection. However, since the connection exists over a series of many messages, a mechanism is provided to stop the flow of messages when the receiving process's Host is overloaded. This mechanism is an allocation system whereby the sender is notified of the receiver's buffering capacity as part of the connection setup, and the sender stops transmitting when it has transmitted enough messages to exhaust the capacity, unless a control message carrying notification of replenishment of the buffering capacity at the receiver arrives first. Multiple connections may be in effect simultaneously.

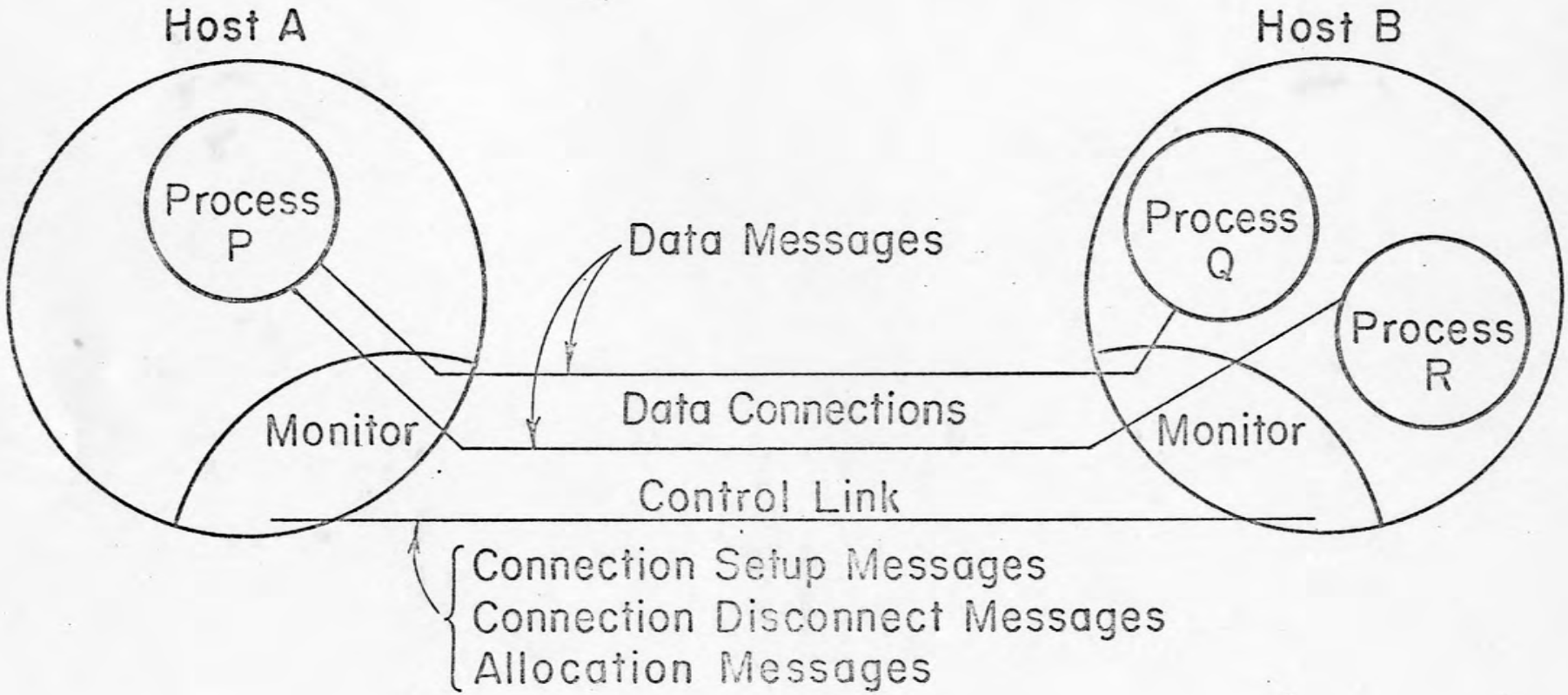
Host A
Host B
Process P
Process Q
Process R
Data Messages
Data Connections
Monitor
Monitor
Control Link
Connection Setup Messages
Connection Disconnect Messages
Allocation Messages

Figure 4

2.4 The TELNET Protocol

The TELNET protocol [9,10] has two parts, the specification of the network virtual (or standard) terminal (summarized on the left side of figure 5) and the specification of how the network virtual terminal converses with a server Host, the TELNET protocol itself (summarized on the right side of figure 5). The specification of the network virtual terminal is, of course, an attempt to reduce the N^2 problem (the problem of each of the N kinds of terminals having to be able to talk to all N others) to manageable proportions. The specification of the network virtual terminal includes a specification of its printer (for instance, what the printer carriage does in response to a carriage return or a form feed), a specification of the network virtual terminal's keyboard (for instance, a list of keys that all users of network virtual terminals must have a way to type), a specification of a conventional end of line convention (i.e., the end of line is indicated *only* by a carriage return followed by a line feed), and a specification of a mechanism whereby the user at a network virtual terminal can send an interrupt character to the server Host and vice versa. The rest of the TELNET protocol specifies how a network virtual terminal initiates a connection to a server Host (i.e., using the initial connection protocol); specifies the signals a network virtual terminal and a serving Host use to agree on which will supply echoes for user-typed characters and how certain events are synchronized between the network virtual terminal and the serving Host, etc.; specifies how the user at a network virtual terminal can stimulate some of the exchanges between his virtual terminal and the serving Host; and finally, the TELNET protocol specifies the character sets which can be used between the network virtual terminal and the serving Host.

Network Virtual Terminal

- Printer
- Keyboard
- End of Line Convention
- Break and Reverse Break Convention

TELNET Protocol

- Initial Connection Protocol
- Network Virtual Terminal
- TELNET Control Signals Echoing, Data-Mark, etc.
- User TELNET Signals Transmit Now, etc.
- Data Types ASCII, EBCDIC, etc.

Figure 5

2.5 Function-Oriented Protocols

The final class of protocol listed before was the class of function-oriented protocols [1]. Two of these are the Remote Job Entry (or RJE) protocol [11] and the file transfer protocol (or FTP) [12,13]. Basically the RJE protocol specifies the standard conventional method whereby human users and processes use the various systems around the network which expect their users to be at RJE terminals. The protocol is flexible enough to allow a random Teletype (or other keyboard printer terminal), card reader, and line printer to be substituted for the commercially available RJE terminals (e.g., IBM 3780).

The FTP specifies a standard conventional method for file and other structured data transfers between minicomputers, maxi-computers, terminal concentrators, and special purpose file systems in one of five formats — ASCII, wherein the sender converts from its internal character representation into standard ARPA Network ASCII and the receiver does the inverse conversion; image, in which mode arbitrary binary transfers are made by breaking the binary stream into a series of bytes of predetermined size; local byte, wherein it is the responsibility of the user to format to and unformat from the format convenient to the server; print file ASCII, a format suitable for printing on an ASCII printer at the server site; and print file EBCDIC, a format suitable for printing on an EBCDIC printer at the server site.

There are a number of other function-oriented protocols, such as MAIL and Graphics, but these will not be discussed in this paper.

2.6 Summary

The six classes of protocol have the inter-relationship shown in figure 6, where RJE and FTP are but two of the function-oriented protocols. Note that protocols are built on top of one another and that ad hoc protocols are possible at any level. We will return to this figure later.

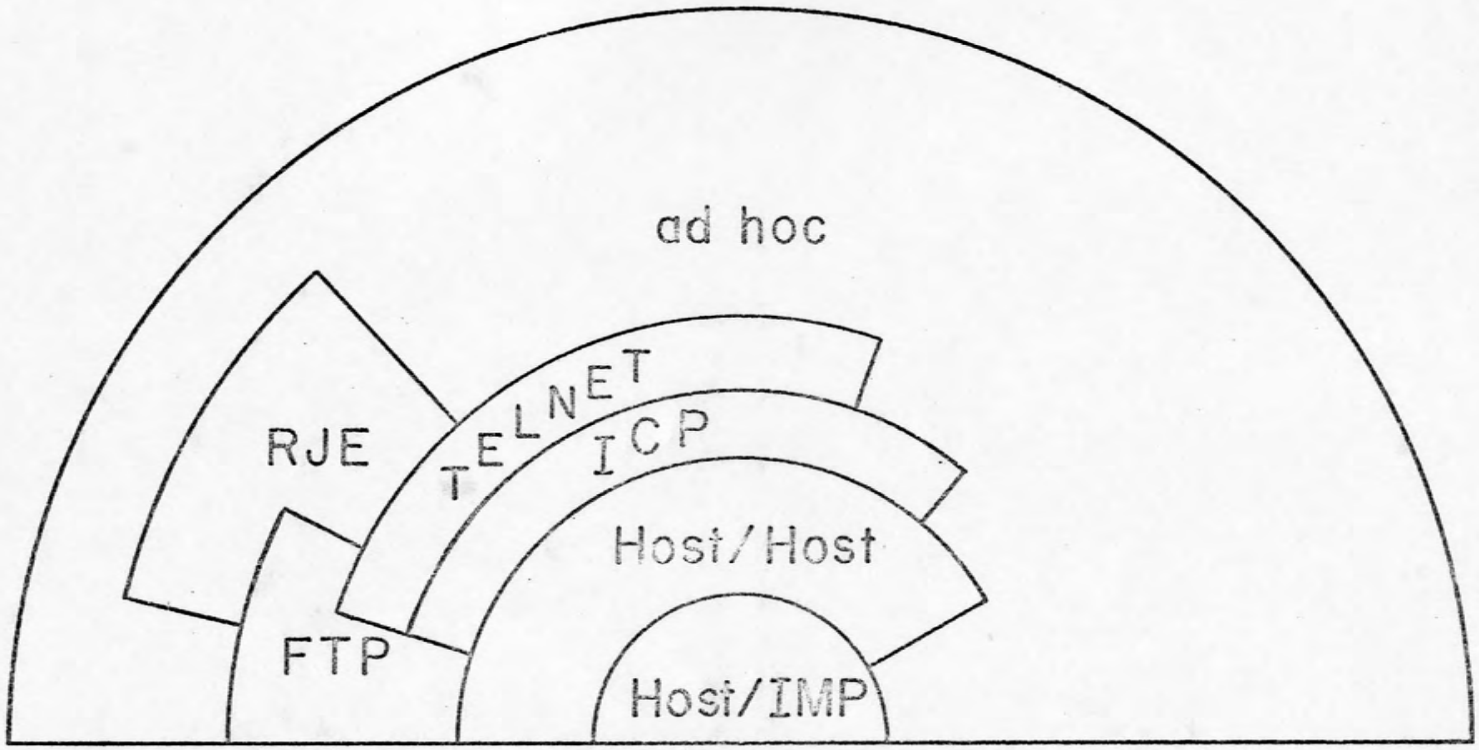

Figure 6

3. NETWORK USAGE AND THE SUPPORTING PROTOCOLS

The ARPA Network protocols are presently called upon to support a few relatively unsophisticated classes of network usage. Included are users at isolated terminals using remote Hosts, users of Hosts accessing selected subsystems (e.g., LISP or FORTRAN) at other Hosts, users of Hosts accessing remote file systems, Host-to-Host and tape-to-tape file transfers, and Remote Job Entry. Each of these classes of usage is pretty self-explanatory, and each has been or can be supported more or less well with the current ARPA Network protocols. At several points over the course of the rest of this paper, the quality of support the current protocols are able to provide will be discussed.

In the foreseeable future, the ARPA Network protocols will also be required to support significant computations of a sophisticated and distributed nature. For these classes of usage, the current ARPA Network protocols appear to be insufficient and some of these areas of insufficiency will be noted. Some examples of distributed operating systems and distributed computations which the protocols will be required to support are given below.

An area which has caused considerable dissatisfaction among network users has been the myriad sets of executive control languages the user must master. There is some ongoing research in this area. A number of Hosts are banding together to decide upon a common executive language. With it users can, for example: ascertain the load of any system in the network; discover, by name, where other users are working and link to them or send them messages; or refer to files in file systems on many computers in a common manner. The TIP, a tiny terminal concentrator [5], is participating in these experiments. For instance, a TIP command

will be implemented which links to the nearest of these 'standard network execs'. (In some cases the TIP will automatically connect its users to one of these execs.) Once in this exec, the TIP user will have many of the facilities available to users on a big Host system. When the user has decided on a program to run, the exec will be able to direct the TIP to disconnect and reconnect the user to the appropriate site. When the program is finished, the TIP will automatically be disconnected from the program's Host and again connected to the 'network' exec.

Further services often requested or required by TIP users are elaborate device and processor status features, and a login and password facility. It is beyond the capacity of the TIP to deal with issues like these by itself. However, we have been led to an interesting observation: since these are capabilities that one normally associates with a big Host, cannot a big Host assist the TIP in providing these features? In fact, a number of TIP features are already implemented with the assistance of a big Host. These are facilities for disseminating news about the network and changes to the TIP, for receiving user complaints and suggestions, for ascertaining which Hosts are 'up' around the network, etc. When a TIP user wants to use one of these facilities, he types a TIP command which causes a network connection to be made to the Host on which these facilities are maintained. When the user is done, the connection is automatically broken. Likewise, the TIP shall soon have a password facility and an accounting facility handled by a Host. Finally, there is no reason why each of the above facilities need be available on only one Host; distributing the facilities would result in greater reliability and in minimizing the load placed on any one Host. The possibility of such 'distributed computations' is clearly one of the advantages of a computer network.

There are presently about ten TENEX [14] systems operating on the ARPA Network. These offer the opportunity for a rather significant experiment with a distributed operating system. One can envision an experiment which is not with eight identical systems communicating with each other but with one system distributed over eight (albeit identical) computers. There could be a single system-wide file system with the capability, for instance, of page faults calling up pages from across the network (although, of course, not swapped back and forth across the network). There could be a single system-wide accounting system. There could be a provision for processes having subprocesses running on other machines with the necessary inter-process communication and protection. There could be one program library with some entries available on all machines and some entries only available on special machines (for instance, the matrix manipulation package might only be available on the TENEX system which will schedule work for the ILLIAC-4).

Further discussion of such distributed operating systems and computations is given in [15] and [16].

4. DIFFICULTIES WITH THE ARPA NETWORK PROTOCOLS

4.1 Difficulties with the Host/IMP Protocol and Effects on Host/Host Protocol

The Host/IMP protocol has significantly affected the design and performance of the other protocols. Particularly influential have been the acknowledgment system initially suggested by the Host/IMP protocol, the retransmission system initially suggested by the Host/IMP protocol, and the message identification system required by the Host/IMP protocol.

It is crucial for the IMPs to limit the rate of flow of Host traffic into the net to the rate at which it is being taken out in order to prevent subnetwork congestion. The IMPs' first attempt (which was insufficient [6,17]) took the form of suggesting that each message in a conversation should be held by the sending Host until an end-to-end acknowledgment for the previous message was received [4]. This suggestion was adopted as a part of Host/Host protocol upon which all the higher standard protocols are based. As a consequence the bandwidth of a single Host/Host protocol connection is severely limited; given the ARPA Network response times using 50 Kbs lines, waiting for a destination-to-source acknowledge between messages typically limits connection bandwidth to about 10 Kbs vs. the 40 Kbs possible with a constant stream of messages [18].

It was originally thought that the ARPA Network would lose a message so seldom that there was no point in Hosts ever bothering with message retransmission. Unfortunately, resolving various possible lockups has required the subnetwork to *discard* a message occasionally, and the topology of the network has evolved into

long series of machines and lines which increase the probability of involuntary message loss. However, the Host/Host protocol followed the initial thought and did not provide for message retransmission. Given the realities of the probability of message loss in the network and given the Host/Host protocol which is inordinately sensitive to any abnormality, the Host/Host protocol (and protocols based on it) has proved quite unreliable, and the original Host/IMP protocol must be held partly responsible.

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

The IMP/Host protocol has been changed [6,7] so that it is no longer necessary to wait for the end-to-end acknowledgment, message order is now preserved except for priority considerations, cases requiring message retransmission are unambiguously reported to the sending Host, and the message identifier has been expanded to a sufficient size. Unfortunately, there is great inertia in the Host/Host protocol due to the large number of implementations which would have to be changed, and therefore, the Host/Host protocol is still incapable of simultaneous high performance and perfect reliability.

4.2 An Alternative to the ARPA Network Host/Host Protocol

Perhaps the most significant characteristic of the Host/Host protocol is that it is based on the concept of logical line switching in contrast to logical message switching. This has profound implications for its simplicity, flexibility, and robustness. A way to understand these implications is to consider the alternative message-switching approach to Host/Host protocol. There are many possible message-switching-based Host/Host protocols but I will describe my own scheme [19]. Consider figure 7. Suppose two processes, P and Q, wish to communicate; in fact, suppose P has a message to send to Q. Rather than setting up a connection (and its attendant flow control mechanism) with which to pass messages from P to Q, for each message Q is willing to receive, it sends a receive control message to Host A. When P has a message to send, it sends the data to its monitor. In whichever order these messages (RECEIVE or data) reach the Host A monitor, they eventually rendezvous; and, at this time, the data is sent to Host B and the relevant entries in the table in the Host A monitor are cleared. When the data gets to Host B, it matches a table entry left when the RECEIVE passed out of Host B. The table entries in Host B are then cleared and the data is passed to process Q.

The significant feature of this is that connections are not maintained over a sequence of messages but instead are set up and expire on a message-by-message basis. There are several advantages to this approach:

1. There is no need for a large number of intermonitor control messages to set up and break connections and consequently no need for a special "out of band" control channel.

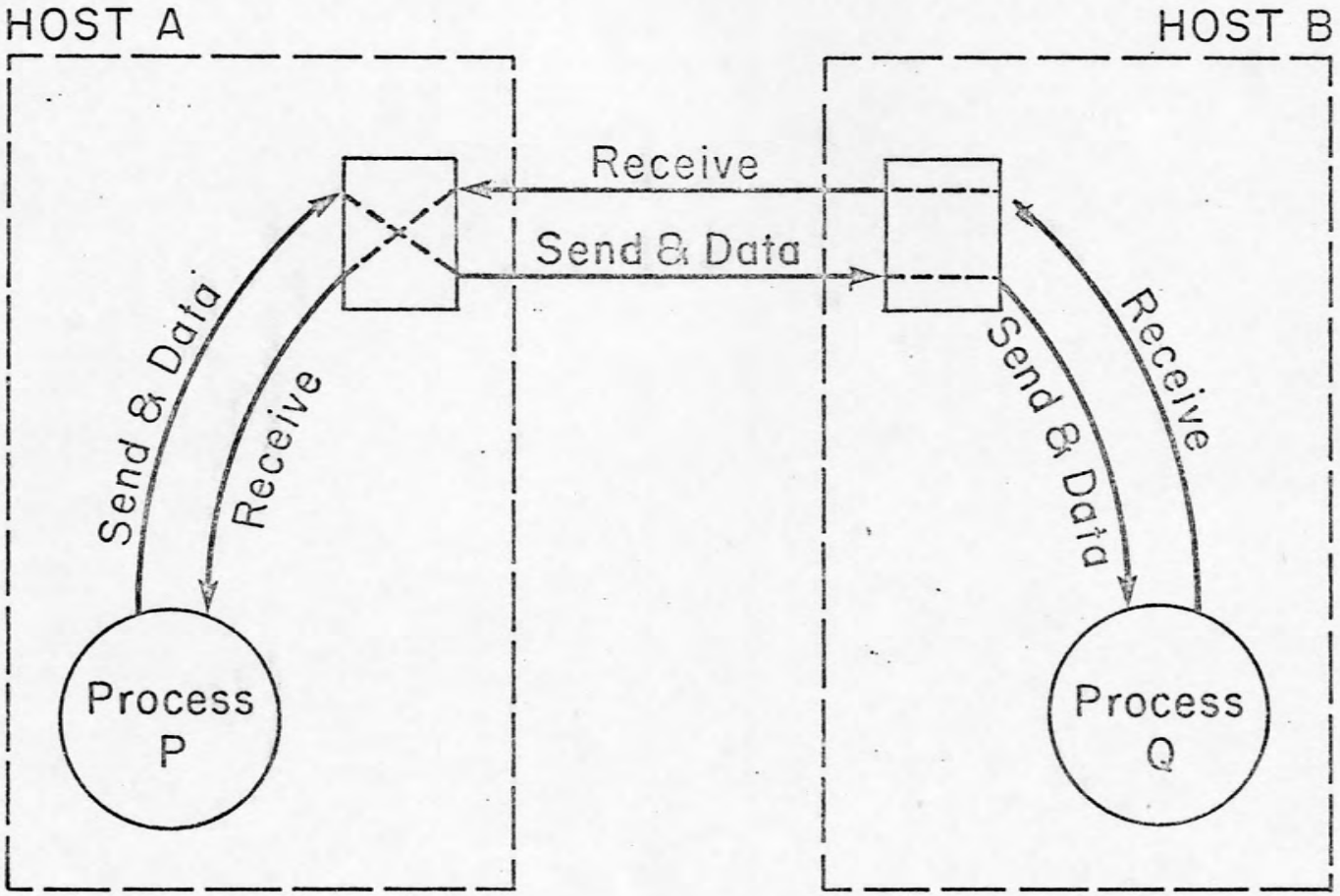
HOST A
HOST B
Receive
Send & Data
Send & Data
Receive
Send & Data
Receive
Process
P
Process
Q

Figure 7

2. The RECEIVING process has the opportunity after *each* message to stop the flow of messages. Further, since the RECEIVE process can be required to provide a buffer, the monitor is relieved of the task of providing an indefinitely large buffering capacity.

3. Because connections exist only fleetingly, relatively complex operations such as dynamically switching "connections" between a variety of processes are easy.

4. Errors have a minimal effect. For instance, if a RECEIVE message is lost, the RECEIVING monitor will time it out and the process can do another RECEIVE just as is normally done when the SENDER refuses to SEND in response to a RECEIVE.

5. This message-switching protocol is suitable for implementation on small computers as well as large computers due to its simplicity and the non-necessity for a large buffering capacity in the monitor.

Other alternate protocols aimed in part at correcting some of the faults of the ARPA Host/Host protocol have been suggested, for example [20].

4.3 Difficulties with the Initial Connection Protocol

The Initial Connection Protocol (ICP) is noteworthy because its very existence is indicative of a mentality which would have every Host be distinct and competing rather than party to a synergistic, resource sharing, confederation of Hosts. The particular ARPA ICP [21] is further indicative of a paranoia the designers had about runaway or bad processes. The designers were convinced that once one connected to a process, there was constant danger of another process sneaking in and pretending to be the desired process. So they required a socket number at each of the two ends of a connection, when one would do, because that was somewhat like two passwords. Actually, things would be perfectly safe with one socket number. Further, in what I think was a mistaken attempt at generality, ARPA connections are simplex; thus, two connections and four sockets must be agreed upon for every full duplex conversation. Finally, ARPA Network Hosts insisted on explicit "Login" of each instance of Host use by network users (possibly a computerized process).

Figure 8, taken from [21], shows what is involved in ICP. Briefly, a server process at a site makes available a well-advertised send-socket L and listens for a user (S1). A user process initiates a connection to send-socket L from its receive socket U specifying link L (U1). When the server receives something at socket L (S2), it confirms the connection from send-socket L to receive-socket U (S3). The server then waits for an allocation from the user (S4). When the user receives confirmation of the connection (U2), it allocates m, messages and b, bits for the connection using link l (U3). The server receives the allocation and sends some data S in s-bit bytes as allowed by the m-message and t-bit allocations (S5). The data S specifies

Server	User
S1: Listen on Socket L.	U1: RTS $(U, L, \underline{\ell}_1)$
S2: Wait for a Match.	U2: Wait for Match.
S3: STR $(L, U, \underline{s}_1)$	
S4: Wait for Allocation	U3: All $(\underline{\ell}_1, \underline{m}_1, \underline{b}_1)$
S5: Send Data S in $\underline{s}_1$ Bit Bytes as Allowed by Allocation $\underline{m}_1$, $\underline{b}_1$.	U4: Receive Data S in $\underline{s}_1$ Bit Bytes.
S6: CLS (L, U)	U5: CLS (U, L)
S7: RTS $(S, U+3, \underline{\ell}_2)$	U6: STR $(U+3, S, \underline{s}_2)$
S8: STR $(S+1, U+2, \underline{s}_3)$	U7: RTS $(U+2, S+1, \underline{\ell}_3)$

Figure 8

that the server is ready to open a connection from socket S+1 and a connection at socket U+2 and another connection from socket U+3. Once the server has sent the user the sockets at which the server is prepared to open connections, it initiates the closing of the L-U connection (S6). Once the user has received the sockets at which the sender is prepared to open connections (U4), the user confirms the closing of the L-U connection (U5). Now, concurrently the user and server exchange commands opening the connection from socket U+3 at the user to socket S at the server using link l_2 and byte size s_2 (S7 and U6). Also concurrently, commands are exchanged opening a connection from server socket S+1 to user socket U+2 (S8 and U7).

Why did we go through all of that? Because at present one cannot call a library square root subroutine across the network without going through all that and considerably more to actually get the library routine running. How much more convenient it would be if the square root routine was just "hanging around" on a well-known socket to which a user could send a number directly along with a socket to which to return the answer. (Great simplification or complete elimination of ICP has been considered in [22] and in recent discussions of Host-to-Host protocols for network interconnection.)

4.4 The Echoing Problem

The Network Virtual Terminal and the TELNET protocol are most network users' introduction to protocols and network use. Because of the initial idea of a human user at a standard terminal communicating with a Host, the TELNET protocol is unduly limited, and a basically nice mechanism has not been able to elegantly support the usage it should have been able to support; for instance, process-to-process communication. As this has been discussed in detail in [23], we discuss in the rest of this section one detail of the TELNET protocol, the echoing issue. This issue, and the approach taken, is representative of the approach to many other TELNET issues.

The echoing issue is basically as follows. Some Hosts assume that terminals on the system provide their own echo (e.g., IBM 2741's on IBM 360 systems). Others assume the system provides the echoes to characters typed on system terminals (e.g., Model 33 Teletypes on TENEX [14]). In a network including both forms of terminals and both forms of Hosts, how is agreement reached between an arbitrary terminal and an arbitrary Host as to which is to echo characters typed on the terminal? The initial convention [9] adopted in the ARPA Network TELNET protocol is based on the fact that some terminals cannot turn off local echoing (e.g., the IBM 2741) and the (open to question) assumption that every Host can supply remote echoes or not supply remote echoes at will. Thus the TELNET echoing convention allows the user (terminal) site to request that the server site either supply or not supply remote echoes. The server must switch to the appropriate mode when asked and acknowledge the switch. We examine some problems with this (simplistic) echoing convention.

The problems come in five areas: server requirements for changing the echoing mode of the terminal; loops; races; the need for symmetric echoing; and highly interactive, long distance echo control.

The basic observation to be made regarding echoing is that servers seem to be optimized to best handle terminals which either do their own echoing or do not, but not both. Therefore, the initial TELNET echoing conventions, which prohibit the server from initiating a change in echo mode, seemed overly confining. The servers are burdened with users who are in the 'wrong' mode, in which they might not otherwise have to be, and users, both human and machine, are burdened with remembering the proper echoing mode, and explicitly setting it up, for all the different servers. This prohibition was imposed on the servers to prevent loops from developing because of races which might arise when the server and user both try to set up an echo mode simultaneously with insufficient care.

A revised version of the Telnet protocol [10] provides a method wherein both parties can initiate a change of echo mode without loops. This alternate specification relies on three primary assumptions. First, the server, as well as the user, should be able to suggest the echo mode. Second, all terminals must be able to provide their own echoes, either internally or by means of the local Host. Third, all servers must be able to operate in a mode which assumes that a remote terminal is providing its own echoes. Both of these last two result from the quest for a universal, minimal basis upon which to build.

An implementation based on these rules has the following commands: ECHO, when sent by the server to the user, means 'I'll echo to you'; ECHO, when sent by the user to the server, means

'You echo to me'. NO ECHO, when sent by the server to the user, means 'I'll not echo to you'; NO ECHO, when sent by the user to the server, means "Don't you echo to me'.

Whenever a connection is opened between a server and user, both server and user assume that the user is echoing locally. If the user would, in fact, prefer the server to echo, the user sends off an ECHO command. Similarly, if the server prefers to do the echoing (for instance, because the server system is optimized for very interactive echoing), the server sends off an ECHO command. Neither is *required* to do anything; it is only a matter of preference. Upon receipt of either command by either party, if that is an admissable mode of operation the recipient begins operating in that mode, and if such operation reflects a *change* in mode, it should respond with the same command to confirm that (and when) the changeover took place. If the received command requests an inadmissable mode of operation, then the command's inverse should be sent as a refusal (this must be NO ECHO, since neither party is allowed to refuse a change into NO ECHO).

Several properties of this scheme are worthy of note: 1) NO ECHO is retained as the nominal connection mode; a connection will work in ECHO mode only when both parties agree to operate that way; 2) the procedure cannot loop; regardless of which party (or both) initiates a change, or in what time order, there are at most three commands sent between the parties; 3) servers are free to specify their preferred mode of operation; thus, human or machine, users do not have to learn the proper mode for each server.

One of the problems that must be solved about echoing is the resolution of the difference in time between when a command to change to echo mode is given and when it takes place, so some characters are not printed twice or not at all. To eliminate this ambiguity, whenever a party initiates a request for a change in echo mode, it could then buffer, without transmitting or processing, all data in the user-to-server data stream until it receives an acknowledgment, positive or negative, at which time it deals with the buffered data in the newly negotiated mode. Since such a request is guaranteed to be acknowledged, the buffering time is bounded.

An important aspect of this technique of eliminating the synchronization problem is that it need not ever become part of the official protocol. Since its operation is entirely internal to the server or user, each may independently weigh the value of elegance against the cost of the required code and buffer space.

In what we have described so far, the interpretations of the ECHO command — 'I'll echo to you' and "You echo to me' — implicitly assume that both the server and user know who is which. This is a problem for server-server connections, where it is not clear which is the user, and also for user-user connections, e.g., in linking Teletypes together, where it is not clear which is the server.

Responding to this, one comes to understand that there are five reasonable modes of operation for the echoing on a connection pair, as shown in figure 9, and that four commands are sufficient to deal with completely symmetric echoing. We have actually already mentioned the four commands — the two possible meanings

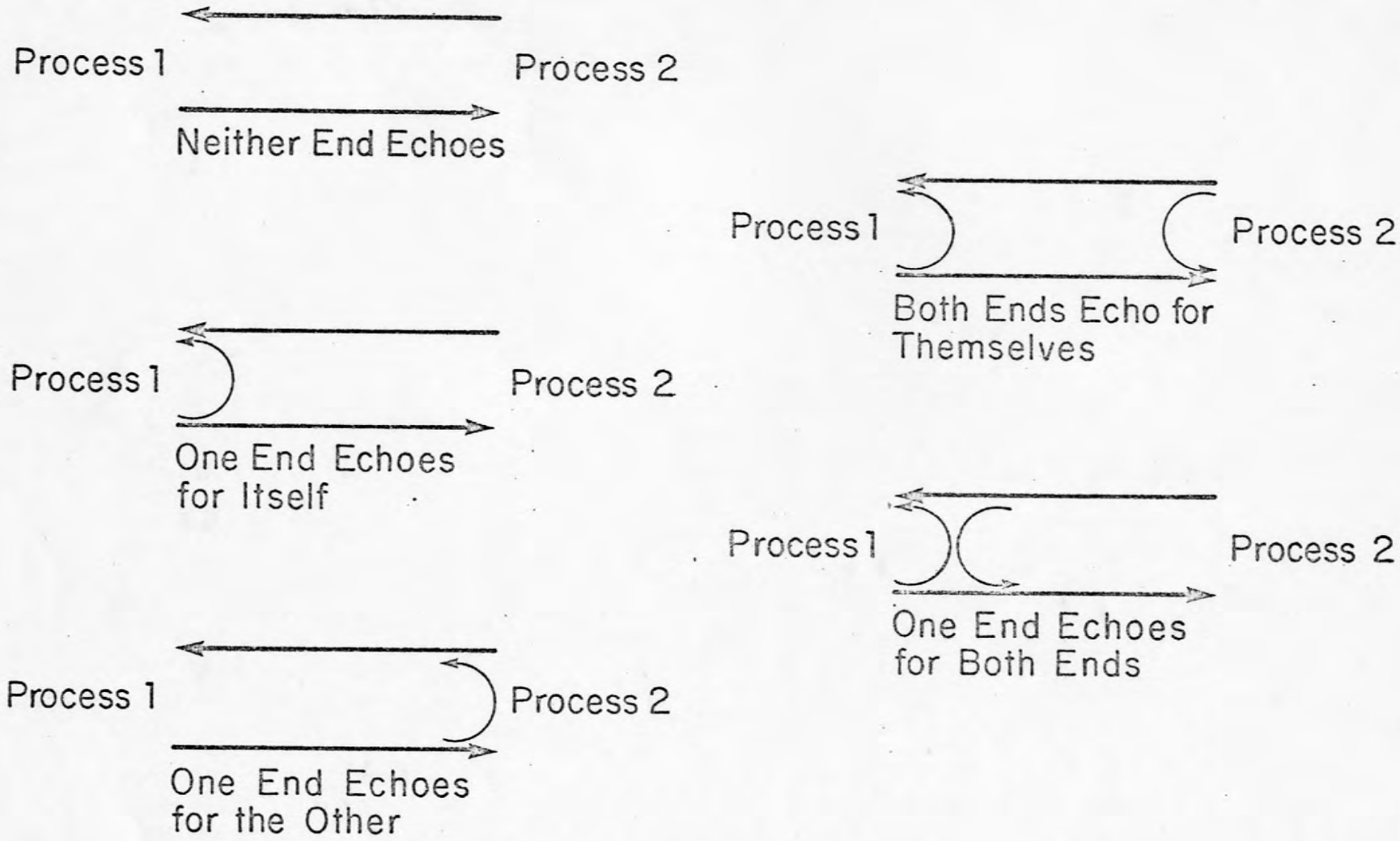
Process 1
Process 2
Neither End Echoes
Process 1
Process 2
Both Ends Echo for Themselves
Process 1
Process 2
One End Echoes for Itself
Process 1
Process 2
One End Echoes for Both Ends
Process 1
Process 2
One End Echoes for the Other

Figure 9

for each of ECHO and NO ECHO. Explicitly, the commands would be I'LL ECHO TO YOU, YOU ECHO TO ME, DON'T ECHO TO ME and I'LL NOT ECHO TO YOU. Echoing is now the negotiation of two options, and the initial, default modes are DON'T ECHO TO ME and I'LL NOT ECHO TO YOU.

We now consider highly interactive echoing across the network (even over satellite links). For example, as shown in figure 10, the server prints LOGIN, the user types his name (which should be echoed locally) followed by a terminator, the server then prints PASSWORD, the user then types his password (which should not be echoed by either the user or server system), the server prints ACCOUNT #, and finally the user types his account number (which should be echoed locally) followed by a terminator. As another example of the desired capability, suppose the user types the characters shown in the top line of figure 11. The resulting printout on the Teletype should look like the bottom line (where the server printout is underlined) no matter what speed the user types and no matter the distance between the user and server.

The mechanism required to permit such flexible, interactive echoing has several parts: server declaration of a list of "break" characters upon which special action is to be taken by the user system; server declaration to the user system of the action the user system is to take with the break characters (to either echo locally or not.echo locally); a server command to the user system to move characters from the terminal input buffer to the output buffer until a break character is scanned which is either echoed or not depending on the action previously declared by the server; a server command to the user system to remove characters from the terminal input buffer and discard them until a break character is received. Consider together figure 12 and the examples of figures 10 and 11.

LOGIN: WALDEN PASSWORD: ACCOUNT # 1ØØØ

Figure 10

CO$^{e}_{c}$ABC$^{e}_{c}$XYZ

COPY (FROM FILE) ABC (TO FILE) XYZ

Figure 11

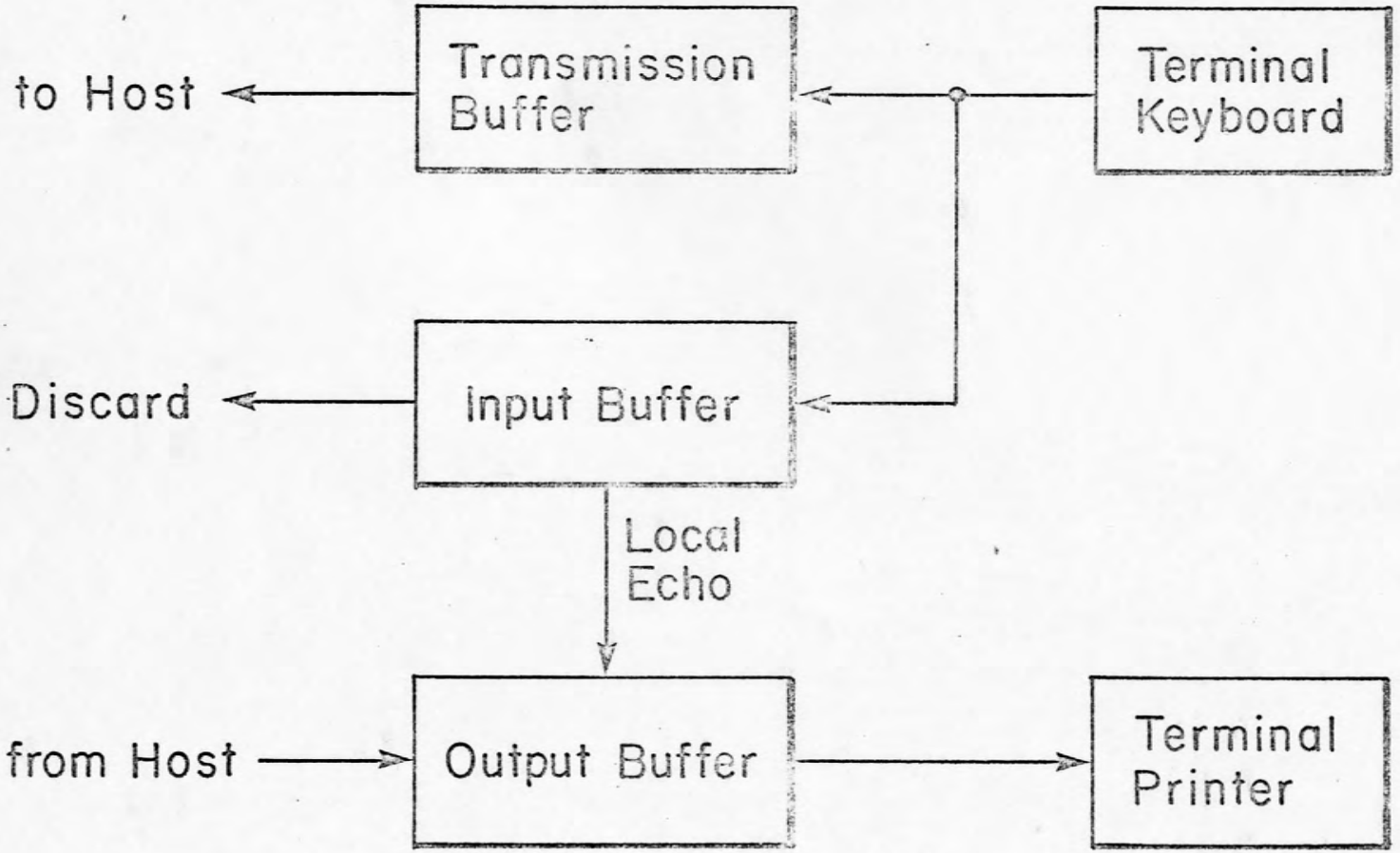
to Host
Transmission Buffer
Terminal Keyboard
Discard
Input Buffer
Local Echo
from Host
Output Buffer
Terminal Printer

Figure 12

In the first example, space is a break character to be echoed. The mechanism starts up in local echo mode and the Host types LOGIN and the user types his name which is locally echoed. As the user types the space following his name, local echoing is disabled. The Host prints PASSWORD and instructs the user system to discard characters to the next break character. Thus, when the user types his password, it is discarded and not echoed. The space after the password is echoed and the discard path is disabled. The Host prints ACCOUNT # and then instructs the user system to echo characters to the next break character which causes the account number to be locally echoed. In our second example, ESC is a break character which is not echoed. In this example the user keeps typing his input without waiting for the Host output; but, nonetheless, the Teletype printout looks perfect since after each ESC both the discard and local echo paths are disabled and the input characters are buffered in the input buffer until a command to echo to the next break character is received from the Host. This remote controlled echoing system was originally described in [24].

4.5 Difficulties with the Higher Level Protocols

The higher level protocols such as Remote Job Entry and File Transfer have been hard to implement and frequently inefficient. I believe this is due to their excess generality and the fact that they are built upon layer upon layer of lower level protocols.

Consider layering first. As McKenzie has noted [25];

> "There are many advantages to a layered approach, for example, it certainly makes thinking about the protocols easier than a monolithic approach would. In addition, implementations divide cleanly, and there can be a well defined interface between programs implementing various portions of the protocol. Nevertheless, there is at least one important difficulty with the layered protocol approach. The difficulty is in the number of interfaces between processes which a message must cross between the time that the data is generated and the time that it is sent into the network. If there are many such interfaces, there is likely to be a lot of wheel spinning involved in massaging the data to translate it into the form suitable for the next layer of protocol."

For example, in the ARPA Network the Remote Job Entry protocol, as it is currently defined and as shown in figure 13, requires the use of the file transfer protocol and the Telnet protocol. The Telnet protocol in turn uses the initial connection protocol, the Host-Host protocol, and the Host-IMP protocol. The File Transfer Protocol, as shown in figure 14, requires the use of the Telnet protocol, the Initial Connection Protocol, the Host-Host protocol and the Host-IMP protocol, and the second occurrence of the Telnet protocol (within the File Transfer Protocol) also requires the Initial Connection Protocol, the Host-Host protocol and the Host-IMP protocol. While this may

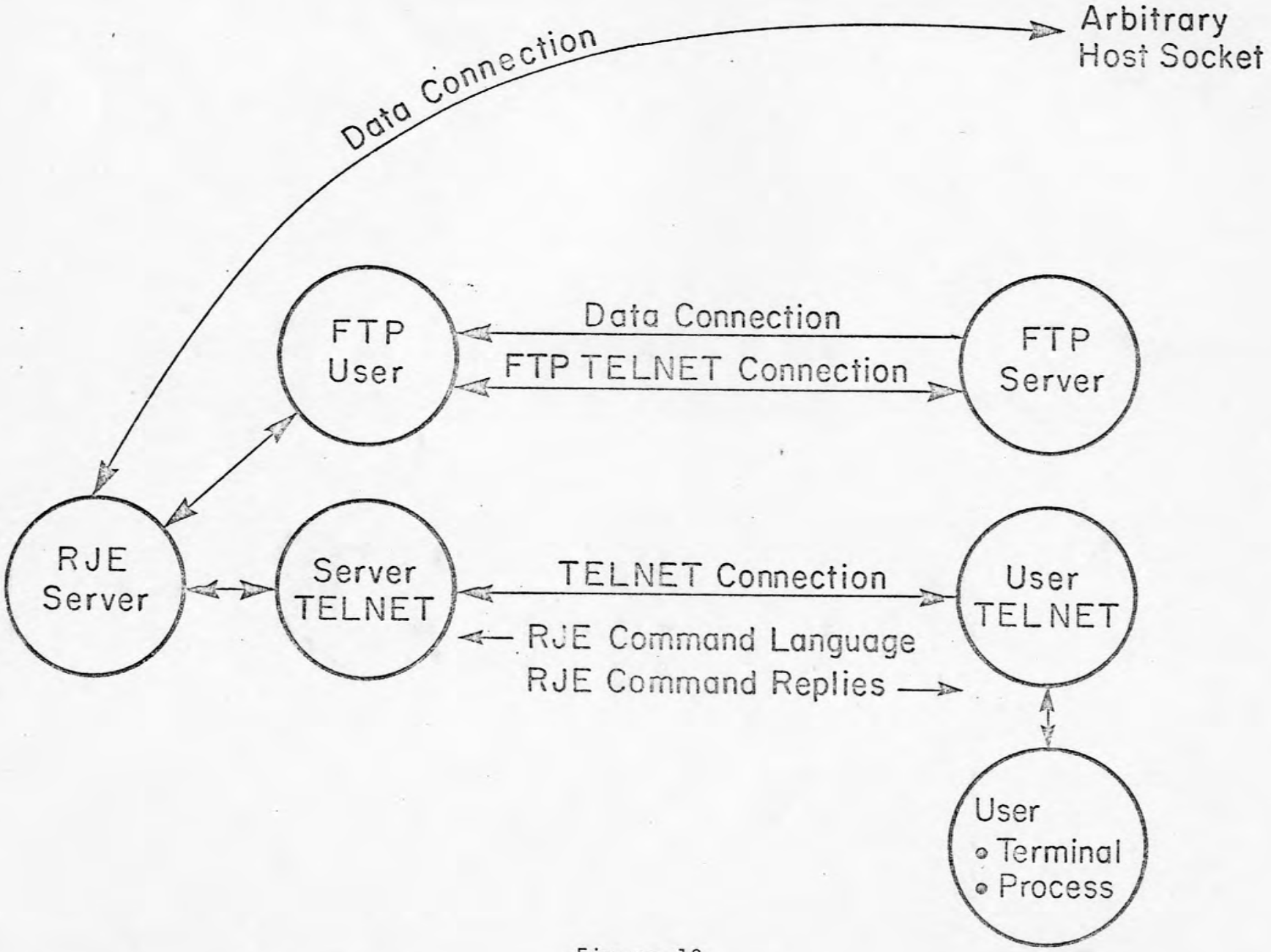
Arbitrary
Host Socket
Data Connection
FTP
User
Data Connection
FTP TELNET Connection
FTP
Server
RJE
Server
Server
TELNET
TELNET Connection
User
TELNET
RJE Command Language
RJE Command Replies
User
• Terminal
• Process

Figure 13

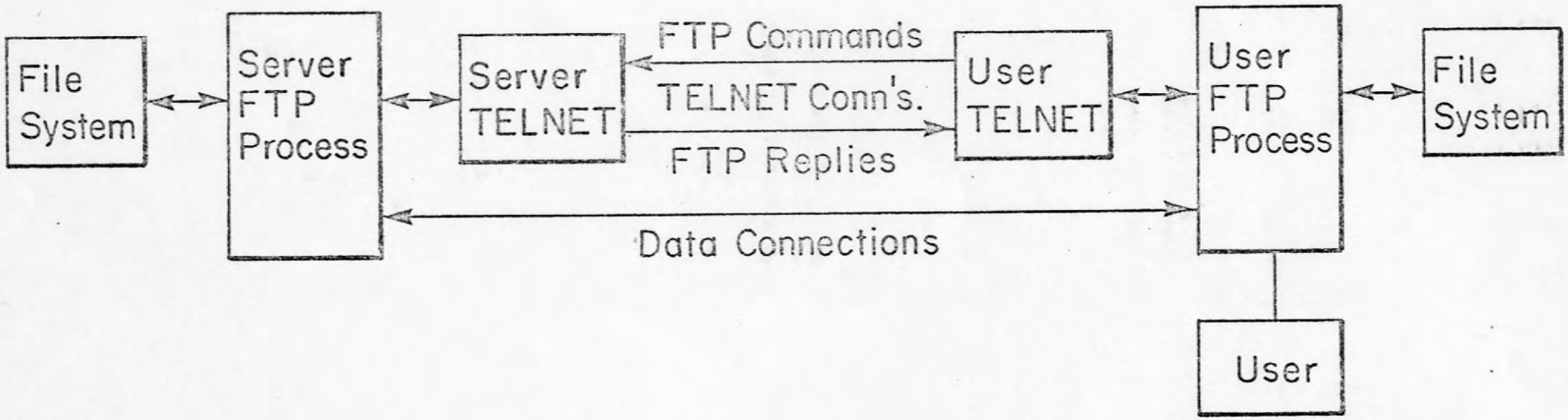
File System
Server FTP Process
Server TELNET
FTP Commands
TELNET Conn's.
FTP Replies
User TELNET
User FTP Process
File System
Data Connections
User

Figure 14

sound more cumbersome than is actually the case, it is clear that by building upon lower level mechanisms, there is a possibility of introducing considerable processing delay and processing expense.

Another look at figure 6 will help the reader get the "big picture". Notice the embedding of FTP within RJE, TELNET within both FTP and RJE, ICP under TELNET, Host/Host beneath ICP and FTP, and Host/IMP below everything. Curiously, the recommended implementation of the Host/Host protocol, upon which all the standard higher level protocols reside, suppresses from the user the IMP/Host message boundaries which could be used to mark record beginnings and other critical points in the data stream.

Now consider the assertion of excess generality. We use the file transfer protocol as an example. As summarized in figure 15, the FTP tries to cover users as varied as tiny servers, big servers, data concentrators with almost no computing power, and the trillion bit store which will be part of the ARPA Network. It provides five different modes of representing files to be transferred, four different modes for actually transferring the files, and the necessary error recovery and restart mechanisms for all of these modes. It specifies dozens of commands and replies. To my mind, FTP gets its generality in a primitive way — by throwing in a little something for everybody.

My view, perhaps an extreme view, is that there should be a single, general, simple, albeit inefficient protocol which could be used by all casual users of RJE, file transfer, and similar functions. This would perhaps be embedded in TELNET. All communication between pairs (or other natural groups) of serious users should be done using special purpose protocols.

FILE TRANSFER PROTOCOL

EXTENT

Mini's, Maxi's, TIPs, Datacomputer Servers, Users

DATA TRANSFER FUNCTIONS

Establishing Data Connections

Data Representation and Storage

- ASCII
- Image
- Local Byte
- Print File ASCII
- Print File EBCDIC

File Structure and Transfer Modes

- Stream
- Test
- Block
- Hasp

Error Recovery and Restart

FILE TRANSFER FUNCTIONS

Commands

- Access Control
- Data Transfer
- Service

Replies

Figure 15

Another example, again from McKenzie [25] supports this view:

> "To cite what may be a worst case, there was an investigator at the University of Utah who desired to send a large array of floating point numbers to the UCLA 360/91. The throughput that this investigator obtained was very poor, and while I have not personally investigated the causes, I suspect that something of the following nature may have happened: The floating point numbers were turned into Utah's output string format (since it was a PDP-10 the format was very likely five 7-bit characters in a 36-bit word). Next, these characters were probably converted from internal typescript format to network virtual terminal code, namely, 8-bit ASCII characters. Following transmission of these characters through the network to UCLA, they were very likely translated from 8-bit ASCII to 8-bit EBCDIC and finally from 8-bit EBCDIC to UCLA's representation of floating point numbers. Imagine how much simpler and less costly this process would have been if the user at Utah had converted the numbers himself directly into UCLA's floating point format and shipped them across the network via a private protocol. Only one translation of the data would have been performed and it would have been performed on the computer which is probably best suited (of these two) to data translation. There would still remain the word length mismatch problem, but the program at Utah could have packed the UCLA-format floating point numbers properly before transmitting them. It should be noted that the reason for the poor performance of the data transfer is speculation on my part. Nevertheless, I think it is a valid illustration of the usefulness of private arrangements for the sake of efficiency. At the very least, the higher level function oriented protocols should reside directly on the IMP/Host protocol."

5. CONCLUSION

There follows a list of characteristics I think Host-to-Host protocols should have. This list is rather long, probably still not complete, and some of the entries are not very profound. Nonetheless, perhaps it will be a useful list.

• Flow control seems to be necessary at every level within a packet switching network: between IMPs, between Hosts, and between Hosts and their terminals. This is because there is the possibility for buffering and variable length delays within a packet-switching network. These cannot occur on a line-switching network.

• The problem of word length mismatch must be addressed in any protocol for a network of heterogeneous computers.

• At BBN we believe the measures of network performance are throughput, delay, reliability, and cost. It is the trade-off between reliability and cost that Network Analysis Corporation makes for the ARPA Network with respect to the layout of the wideband circuits. Consideration of this trade-off should be carried over to protocol design — one should always be concerned while designing protocols that the protocols are insensitive to transients in the network (and this is not necessarily true of the current ARPA Network Host/Host protocol). A reasonable price paid for mechanisms for discovery of and recovery from errors is of crucial importance in the design of any protocol. Turning to network delay and throughput, we find that most workers try to optimize one or the other. In fact, networks and therefore network protocols must simultaneously provide low delay for some traffic and high throughput for other traffic, and these two issues seem always to be in conflict. It is our view that the

protocol which does not have two reasonably distinct mechanisms for providing low delay and high throughput will be doomed to fail for one or the other.

• It seems to be true that network protocols must provide some mechanism between communicating systems which can be used for interrupts.

• The design of reliable network protocols would be much simplified if network-wide agreement could be reached on the length of timeouts of various (perhaps erroneous) events. Without consistent network-wide timeouts, protocols are likely to be fraught with races.

• A well-designed set of protocols would provide three levels of protocol: a single simple all purpose protocol for all casual use, efficient special purpose protocols for production use, and ad hoc protocols for any other use.

• Well designed protocols should provide for (although not necessarily require) end-to-end acknowledgments and retransmission of messages as necessary. A Host-to-Host protocol should not necessarily have to put its own checksums on messages, detect the need for retransmission, provide for message ordering, and provide for duplicate detection (although it should not be prevented either). A nice balance between work which has to be done by the Host/Host protocol and work which has to be done by the communications subnetwork is provided in the ARPA Network. The IMP subnetwork provides message ordering and end-to-end positive acknowledgments. The positive acknowledgment is passed to the source Host. In the rare event that the subnetwork is unable to deliver the message correctly, a negative acknowledgment is delivered to the source Host which unambiguously tells

the source Host that if it is ever going to bother with retransmission, this is the time to do it.

• Host-to-Host protocols should be instrumented, as this may be the only way we will be able to understand and learn from their performance.

• Host-to-Host protocols, inasmuch as they are committee efforts and inasmuch as they are to be implemented on a number of independent systems, must be accurately, clearly and unambiguously documented. Alex McKenzie has suggested that if a protocol design committee fails to produce a document from which it is possible to implement the protocol, then the committee has failed to design a protocol.

• A good protocol should not limit communication to be between pairs of entities. Resource sharing and distributed computation frequently require multiple user 'connections' and the users party to one of these 'connections' may dynamically vary.

• At all levels of protocol, there should be mechanisms to provide synchronization of events.

• Protocols would do well to utilize message or packet boundaries in natural ways.

One final point: while we are building standard Host/Host protocols trying to reduce the N^2 problem to manageable proportions as far as the operating systems are concerned, we must not push the N^2 problem onto the users leaving them with myriad formats and languages to learn.

REFERENCES

1. S. D. Crocker, J. F. Heafner, R.M. Metcalfe, and J.B. Postel, "Function Oriented Protocols for the ARPA Computer Network," AFIPS Conference Proceedings, Vol.40, June 1972, pp. 271-279.

2. L.G. Roberts and B.D. Wessler, "Computer Network Development to Achieve Resource Sharing," AFIPS Conference Proceedings, Vol. 36, June 1970, pp. 543-549.

3. P.M. Karp, "Origin, Development and Current Status of the ARPA Network," Seventh Annual IEEE Computer Society International Conference, 1973, pp. 49-51.

4. F.E. Heart, R.E. Kahn, S.M. Ornstein, W.R. Crowther, and D.C. Walden, "The Interface Message Processor for the ARPA Computer Network," AFIPS Conference Proceedings, Vol. 36, June 1970, pp. 551-567; also in "Advances in Computer Communications," W.W. Chu (ed.), Artech House, Inc., 1974, pp. 30-316.

5. S.M. Ornstein, F.E. Heart, W.R. Crowther, S.B. Russell, H.K. Rising, and A. Michel, "The Terminal IMP for the ARPA Computer Network," AFIPS Conference Proceedings, Vol. 40, June 1972, pp. 281-293.

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

7. BBN Report No. 1822, "Specifications for the Interconnection of a Host and an IMP," revision of March 1974.

8. A.A. McKenzie, "Host/Host Protocol for the ARPA Network," Network Information Center #8246, January 1972.

9. J. Postel, "TELNET Protocol," RFC #318, April 1972.

10. A.A. McKenzie, "TELNET Protocol Specification," Network Working Group RFC #495, May 1973.

11. R. Bressler, R. Guida, and A. McKenzie, "Remote Job Entry Protocol," Network Working Group RFC #407, October 1972.

12. A. Bhushan, "File Transfer Protocol," Network Working Group RFC #354, July 1972.

13. N. Neigus, "File Transfer Protocol," Network Working Group RFC #542, July 1973.

14. D.G. Bobrow, J.D. Burchfiel, D.L. Murphy, and R.S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10," Comm. ACM 15,2, March 1972, pp. 135-143.

15. N.W. Mimno, B.P. Cosell, D.C. Walden, S.C. Butterfield, and J.B. Levin, "Terminal Access to the ARPA Network -- Experience and Improvements," Proceedings of the Seventh Annual IEEE Computer Society International Conference, February 1973, pp. 39-43.

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

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

18. BBN Report No. 2491, "Throughput in the ARPA Network -- Analysis and Measurement," J. McQuillan, January 1973.

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

20. V.G. Cerf and R.E. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Transactions on Communications, Vol. COM - 22, No. 5, May 1974, pp. 637-648.

21. J. Postel, "Official Initial Connection Protocol," Network Information Center #7101, June 1971.

22. R. Bressler, D. Murphy, and D. Walden, "A Proposed Experiment with a Message Switching Protocol," Network Working Group RFC #333, May 1972.

23. B. Cosell, D. Walden, "TELNET Issues," Network Working Group RFC #435, January 1973.

24. J. Davidson (for W. Crowther, J. McConnell and J. Postel), "An Echoing Strategy for Satellite Links," Network Working Group RFC #357, June 1972.

25. A.A. McKenzie, "Host/Host Protocol Design Considerations," International Network Working Group, INWG Note 16, January 1973.