Function-oriented protocols for the ARPA Computer Network

by STEPHEN D. CROCKER
Advanced Research Projects Agency
Arlington, Virginia

JOHN F. HEAFNER
The RAND Corporation
Santa Mon ca, California

ROBERT M. METCALFE
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

JONATHAN B. POSTEL
University of California
Los Angeles, California

INTRODUCTION

Much has been said about the mechanics of the ARPA Computer Network (ARPANET) and especially about the organization of its communications subnet.[1,2,3,4,5] Until recently the main effort has gone into the implementation of an ARPANET user-level communications interface. Operating just above the communications subnet in ARPANET HOST Computers, this ARPANET interface is intended to serve as a foundation for the organization of function-oriented communications.[6,7] See Figures 1 and 2 for our view of a computer system and the scheme for user-level process-to-process communications. It is now appropriate to review the development of protocols which have been constructed to promote particular substantive uses of the ARPANET, namely function-oriented protocols.

We should begin this brief examination by stating what we mean by the word "protocol" and how protocols fit in the plan for useful work on the ARPANET. When we have two processes facing each other across some communication link, the protocol is the set of their agreements on the format and relative timing of messages to be exchanged. When we speak of a protocol, there is usually an important goal to be fulfilled. Although any set of agreements between cooperating (i.e., communicating) processes is a protocol, the protocols of interest are those which are constructed for general application by a large population of processes in solving a large class of problems.

In the understanding and generation of protocols there are two kinds of distinctions made. Protocols in the ARPANET are *layered* and we speak of high or low level protocols. High level protocols are those most closely matched to functions and low level protocols deal with communications mechanics. The lowest level software protocols in the ARPANET involve reliable message exchange between ARPANET Interface Message Processors (IMPs).[2,5] A high level protocol is one with primitives closely related to a substantive use. At the lowest levels the contents of messages are unspecified. At higher levels, more and more is stated about the meaning of message contents and timing. The layers of protocol are shown in Figure 3.

A second way of structuring sets of protocols and their design is bound up in the word *factoring*. At any level of protocol are sets of format and timing rules associated with particular groupings of agreements. In the IMPs we find certain protocols pertaining to error handling, while others to flow control, and still others to message routing. At the ARPANET's user-level communications interface there are, among others, separable protocols associated with establishing connections and logical data blocking. These protocols do not nest, but join as modules at the same level.

Before moving on to consider the higher level function-oriented protocols, let us first make a few statements about underlying protocols. There are three lower level software protocols which nest in support of the user-level communications interface for the ARPANET. The lowest of these is the IMP-IMP protocol which provides for reliable communication among IMPs. This protocol handles transmission-error detection and correction, flow control to avoid message congestion, and routing. At the next higher level is the IMP-HOST protocol which provides for the passage of messages between HOSTs and IMPs in such a way as to create virtual communication paths between HOSTs. With IMP-HOST protocol, a HOST has operating rules which permit it to send messages to specified HOSTs on the ARPANET and to be informed of the dispensation of those messages. In particular, the IMP-HOST protocol constrains HOSTs in their transmissions so that they can make good use of available

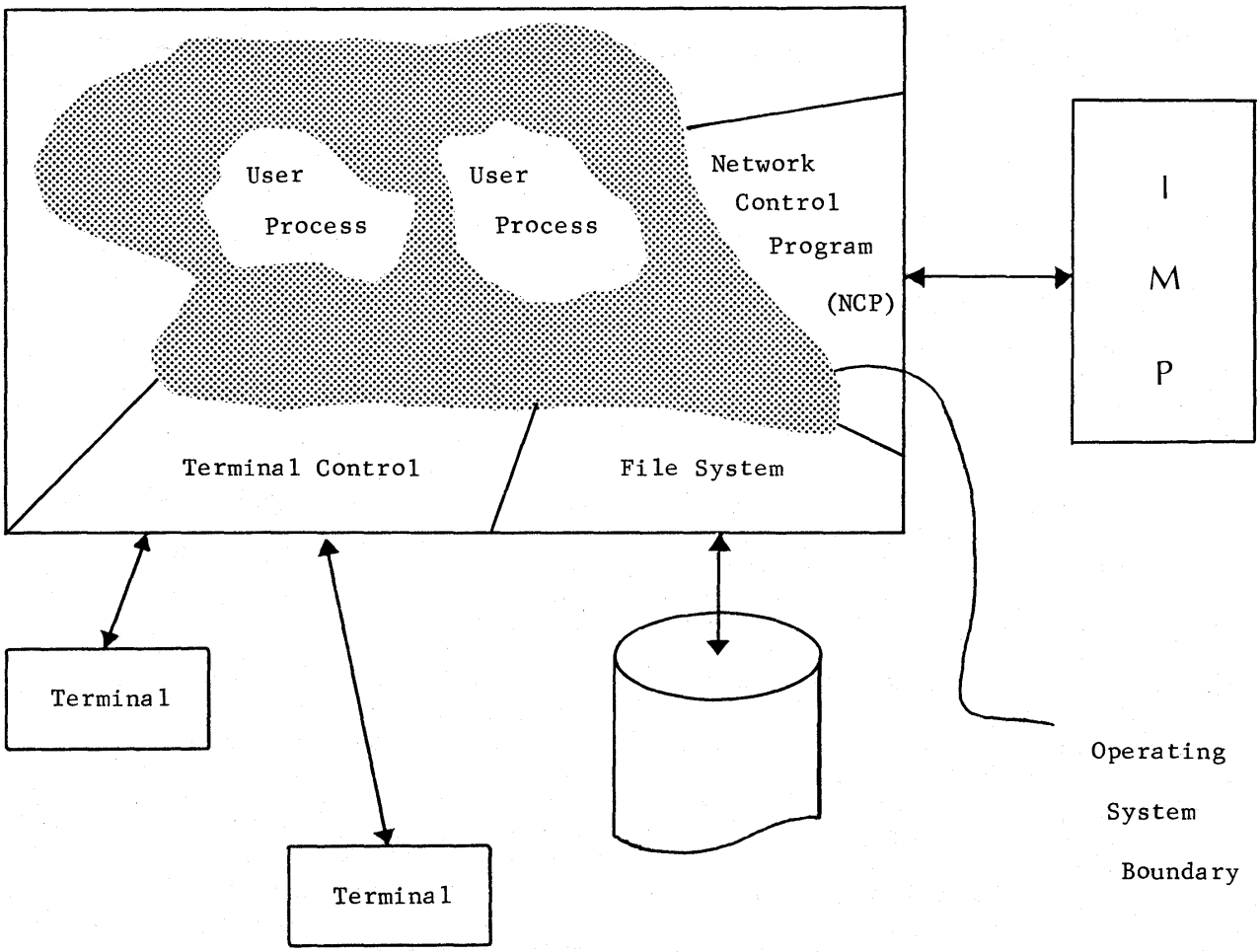

Figure 1—Our view of a computer system

communications capacity without denying such availability to other HOSTs.

The HOST-HOST protocol, finally, is the set of rules whereby HOSTs construct and maintain communication between processes (user jobs) running on remote computers. One process requiring communications with another on some remote computer system makes requests on its local supervisor to act on its behalf in establishing and maintaining those communications under HOST-HOST protocol.

In constructing these low levels of protocol it was the intention to provide user processes with a general set of useful communication primitives to isolate them from many of the details of operating systems and communications. At this user-level interface function-oriented protocols join as an open-ended collection of modules to make use of ARPANET capabilities.

The communications environment facing the designers of function-oriented protocols in the ARPANET is essentially that of a system of one-way byte-oriented connections. Technically speaking, a "connection" is a pair: a "send socket" at one end and a "receive socket" at the other. Primitives provided at the user-level interface include:

1. Initiate connection (local socket, foreign socket),
2. Wait for connection (local socket),
3. Send, Receive (local socket, data),
4. Close (local socket),
5. Send interrupt signal (local socket).

Processes in this virtual process network can create connections and transmit bytes. Connections are subject to HOST-HOST flow control and the vagaries of timing in a widely distributed computing environment, but care has been taken to give user processes control over their communications so as to make full use of network parallelism and redundancy. The kind of agreements which must be made in the creation of

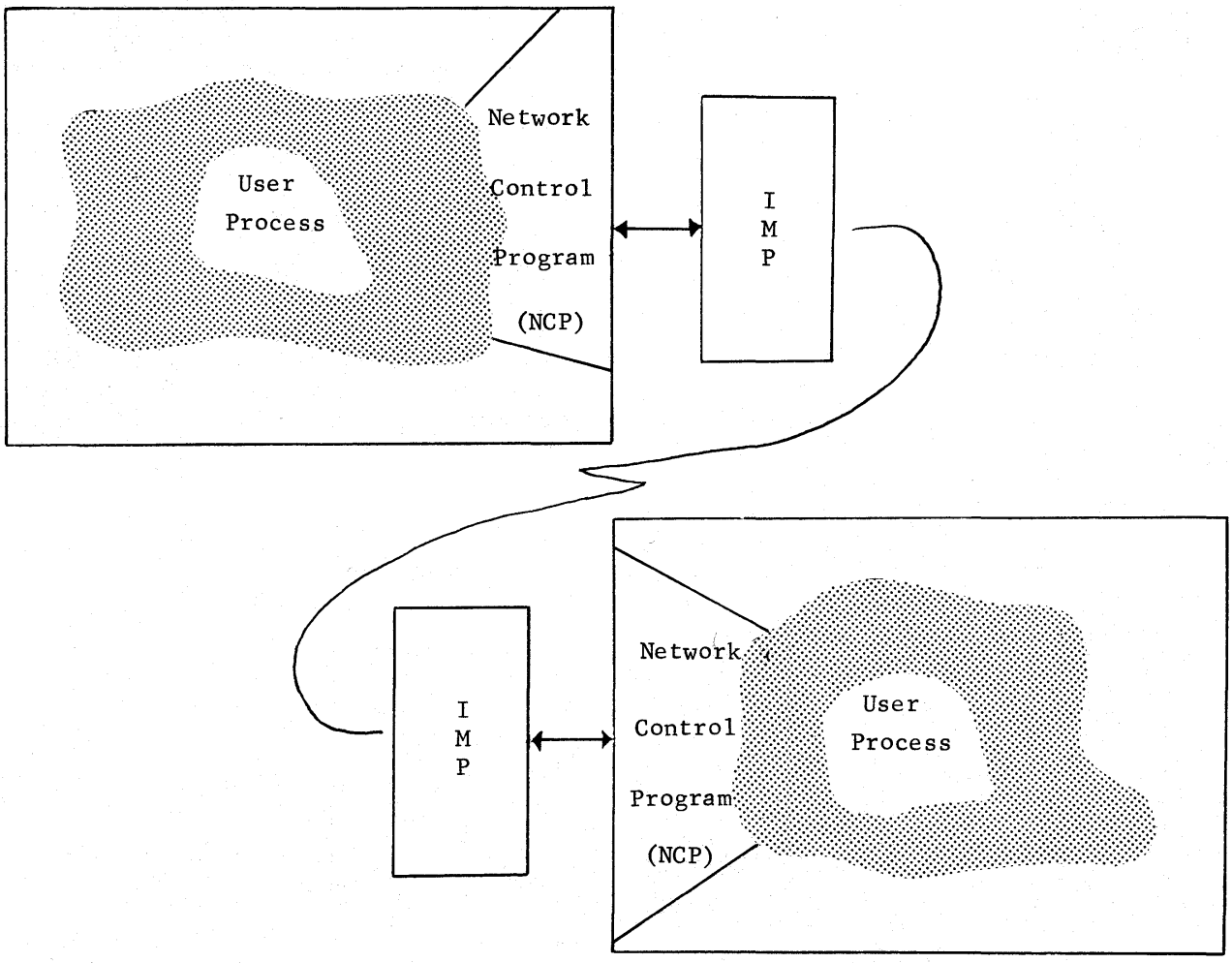

Figure 2—Two communicating processes

function-oriented protocols relate to rules for establishing connections, to the timing rules which govern transmission sequences, and to the content of the byte-streams themselves.

USE OF REMOTE INTERACTIVE SYSTEMS

The application which currently dominates ARPANET activity is the remote use of interactive systems. A Telecommunications Network (TELNET) protocol is followed by processes cooperating to support this application.[8] A user at a terminal, connected to his local HOST, controls a process in a remote HOST as if he were a local user of the remote HOST. His local HOST copies characters between his terminal and TELNET connections over the ARPANET. We refer to the HOST where the user sits as the *using HOST*, and to the remote HOST as the *serving HOST*. See Figure 4.

At the using HOST, the user must be able to perform the following functions through his TELNET user process ("user-TELNET"):

1. Initiate a pair of connections to a serving HOST,
2. Send characters to the serving HOST,
3. Receive characters from the serving HOST,
4. Send a HOST-HOST interrupt signal,
5. Terminate connections.

The user-TELNET needs to be able to distinguish between (1) commands to be acted on locally and (2) input intended for the serving HOST. An escape character is reserved to mark local commands. Conventions for the ARPANET Terminal IMP (TIP) user-TELNET are typical.[9]

In most using HOSTs, the above functions are provided by a user-TELNET which is a *user-level program*. A minimal user-TELNET need only implement the above functions, but several additional support func-

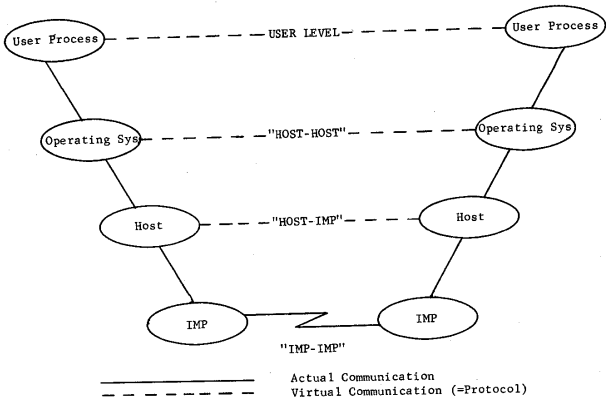

Figure 3—The layers of protocol

tions are often provided (e.g., saving a transcript of a session in a local file, sending a file in place of user-typed input, reporting whether various HOSTs are or have been up).

In the serving HOST it is desirable that a process controlled over the ARPANET behave as it would if controlled locally. The cleanest way to achieve this goal is to generalize the terminal control portion (TCP) of the operating system to accept ARPANET terminal interaction. It is unpleasant to modify any portion of a working computer system and modification could be avoided if it were possible to use a non-supervisor process (e.g., "server-TELNET" or "LOGGER") to perform the job creation, login, terminal input-output, interrupt, and logout functions in exactly the same way

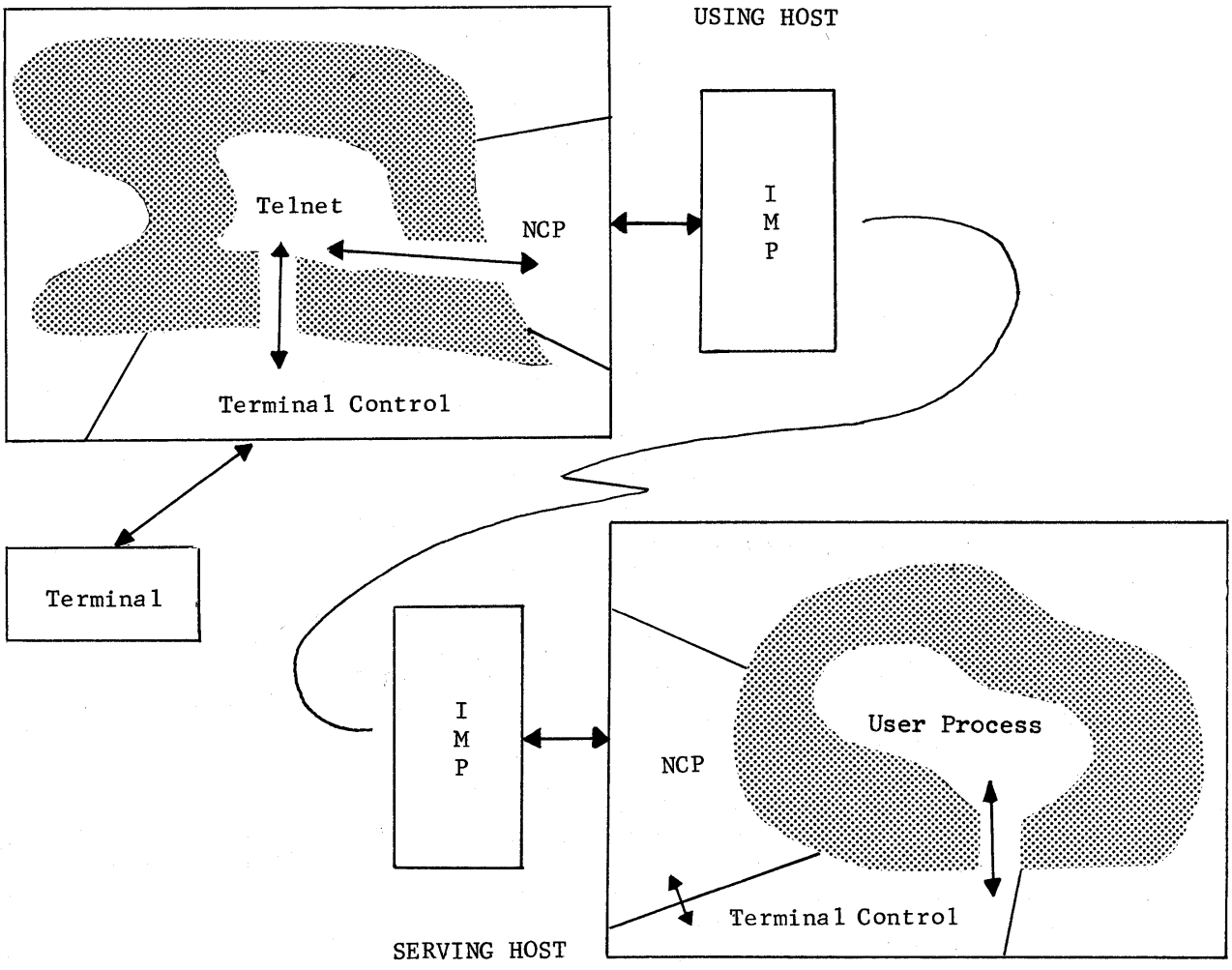

Figure 4—Data flow for remote interactive use

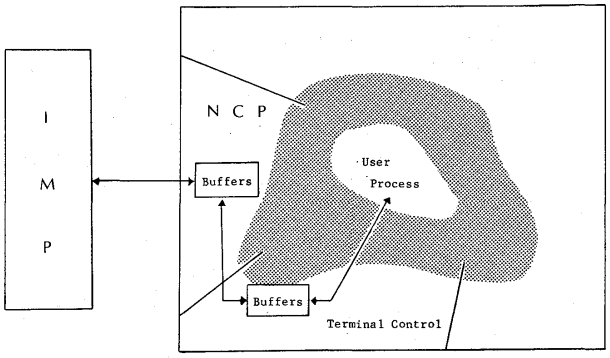

Figure 5—Data flow scheme for server

as a direct console user. Prior to the development of the ARPANET, no operating system provided these functions to non-supervisor processes in anywhere near the required completeness. Some systems have since been modified to support this generalized job control scheme. See Figures 5 and 6.

Efforts to standardize communications in the TELNET protocol focused on four issues: character set, echoing, establishing connections, and attention handling.

The chosen character set is 7-bit ASCII in 8-bit fields with the high-order bit off. Codes with the high-order bit on are reserved for TELNET control functions. Two such TELNET control function codes are the "long-space" which stands for the 200 millisecond space generated by the teletype BREAK button, and the synchronization character (SYNCH) discussed below in conjunction with the purpose of the TELNET interrupt signal.

Much controversy existed regarding echoing. The basic problem is that some systems expect to echo, while some terminals always echo locally. A set of conventions and signals was developed to control which side of a TELNET connection should echo. In practice, those systems which echo have been modified to include provision for locally echoing terminals. This is a non-trivial change affecting many parts of a serving HOST. For example, normally echoing server HOSTs do not echo passwords so as to help maintain their security. Terminals which echo locally defeat this strategy, how-

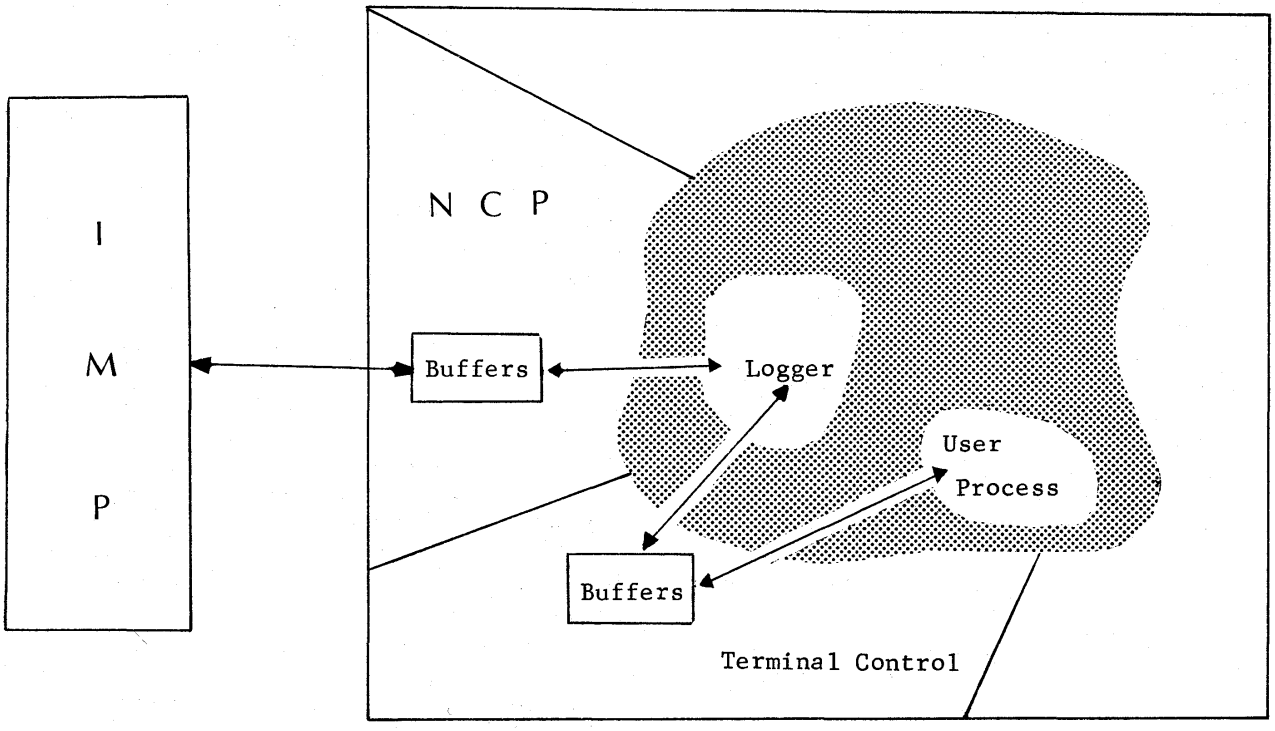

Figure 6—Alternate data flow scheme for a server

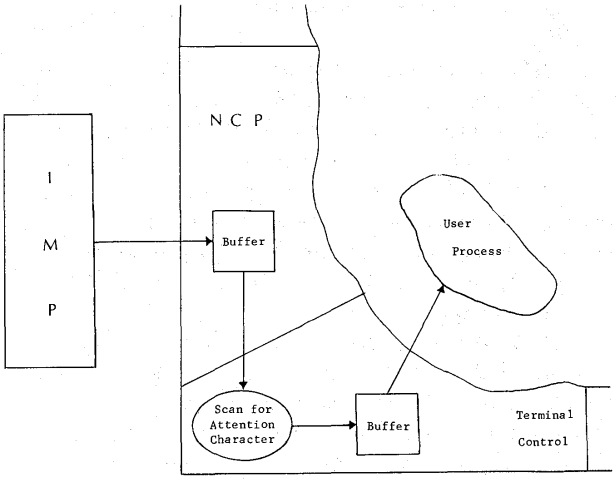

Figure 7—Data flow and processing of the character input stream

ever, and some other protection scheme is necessary. Covering the password with noise characters is the usual solution.

The HOST-HOST protocol provides a large number of sockets for each HOST, but carefully refrains from specifying which ones are to be used for what. To establish communication between a user-TELNET and a server-TELNET some convention is required. The Initial Connection Protocol (ICP)[10] is used:

1. Connection is initiated from a user-TELNET's receive socket to a serving HOST's socket 1 (a send socket).
2. When the initial connection is established, the serving HOST sends a generated socket number and closes the connection. This socket number identifies an adjacent socket pair at the serving HOST through which the user-TELNET can communicate with a server-TELNET.
3. TELNET connections are then initiated between the now specified pairs of sockets. Two connections are used to provide bi-directional communication.

Note that socket 1 at the serving HOST is in use only long enough to send another socket number with which to make the actual service connections.

One of the functions performed by a terminal control program within an operating system is the scanning of an input stream for attention characters intended to stop an errant process and to return control to the executive. Terminal control programs which buffer input sometimes run out of space. When this happens to a local terminal's input stream, a "bell" or a question mark is echoed and the overflow character discarded, *after checking to see if it is the attention character.* See Figure 7. This strategy works well in practice, but it depends rather strongly on the intelligence of the human user, the invariant time delay in the input transmission system, and a lack of buffering between type-in and attention checking. None of these conditions exists for interactive traffic over the net: The serving HOST cannot control the speed (except to slow it down) or the buffering within the using HOST, nor can it even know whether a human user is supplying the input. It is thus necessary that the terminal control program or server-TELNET not, in general, discard characters from a network input stream; instead it must suspend its acceptance of characters via the HOST-HOST flow control mechanism. Since a HOST may only send messages when there is room at the destination, the responsibility for dealing with too much input is thus transferred back to the using HOST. This scheme assures that no characters accepted by the using HOST are inadvertently lost. However, if the process in the serving HOST stops accepting input, the pipeline of buffers between the user-TELNET and remote process will fill up so that attention characters cannot get through to the serving executive. In the TELNET protocol, the solution to this problem calls for the user-TELNET to send, on request, a HOST-HOST interrupt signal forcing the server-TELNET to switch input modes to process network input for attention characters. The server-TELNET is required to scan for attention characters in its network input, even if some input must be discarded while doing so. The effect of the interrupt signal to a server-TELNET from its user is to cause the buffers between them to be emptied for the priority processing of attention characters.

To flip an attention scanning server-TELNET back into its normal mode, a special TELNET synchronization character (SYNCH) is defined. When the server-TELNET encounters this character, it returns to the strategy of accepting terminal input only as buffer space permits. There is a possible race condition if the SYNCH character arrives before the HOST-HOST interrupt signal, but the race is handled by keeping a count of SYNCHs without matching signals. Note that attention characters are HOST specific and may be any of 129 characters—128 ASCII plus "long space"—while SYNCH is a TELNET control character recognized by all server-TELNETs. It would not do to use the HOST-HOST signal alone in place of the signal-SYNCH combination in attention processing, because the position of the SYNCH character in the TELNET input stream is required to determine where attention processing ends and where normal mode input processing begins.

FILE TRANSFER

When viewing the ARPANET as a distributed computer operating system, one initial question is that of how to construct a distributed file system. Although it is constructive to entertain speculation on how the ultimate, automated distributed file system might look, one important first step is to provide for the simplest kinds of explicit file transfer to support early substantive use.

During and immediately after the construction of the ARPANET user-level process interface, several *ad hoc* file transfer mechanisms developed to provide support for initial use. These mechanisms took two forms: (1) use of the TELNET data paths for text file transfer and (2) use of raw byte-stream communication between compatible systems.

By adding two simple features to the user-TELNET, text file transfer became an immediate reality. By adding a "SCRIPT" feature to user-TELNETS whereby all text typed on the user's console can be directed to a file on the user's local file system, a user need only request of a remote HOST that a particular text file be typed on his console to get that file transferred to his local file system. By adding a "SEND-FILE" feature to a user-TELNET whereby the contents of a text file can be substituted for console type-in, a user need only start a remote system's editor as if to enter new text and then send his local file as type-in to get it transferred to the remote file system. Though crude, both of these mechanisms have been used with much success in getting real work done.

Between two identical systems it has been a simple matter to produce programs at two ends of a connection to copy raw bits from one file system to another. This mechanism has also served well in the absence of a more general and powerful file transfer system.

Ways in which these early *ad hoc* file transfer mechanisms are deficient are that (1) they require explicit and often elaborate user intervention and (2) they depend a great deal on the compatibility of the file systems involved. There is an on-going effort to construct a File Transfer Protocol (FTP)[11,12] worthy of wide implementation which will make it possible to exchange structured sequential files among widely differing file systems with a minimum (if any) explicit user intervention. In short, the file transfer protocol being developed provides for the connection of a file transfer user process ("user-FTP") and file transfer server process ("server-FTP") according to the Initial Connection Protocol discussed above. See Figure 8. A user will be able to request that specific file manipulation operations be performed on his behalf. The File Transfer Protocol will support file operations including (1) list remote directory, (2) send local file, (3) retrieve remote file, (4) rename remote file, and (5) delete remote file.

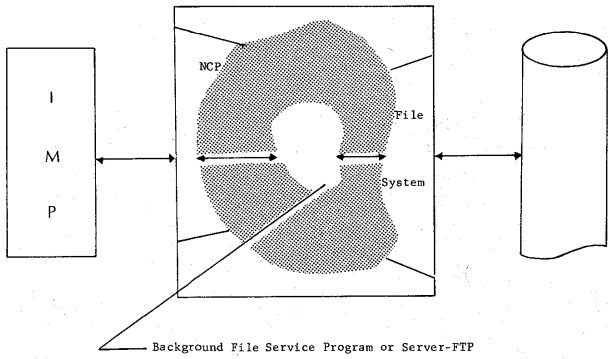

Figure 8—Data flow for file transfer

It is the intention of the protocol designers to regularize the protocol so that file transfer commands can be exchanged by consoles file transfer jobs engaged in such exotic activities as automatic back-up and dynamic file migration. The transfers envisioned will be accompanied with a Data Transfer Protocol (DTP)[11] rich enough to preserve sequential file structure and in a general enough way to permit data to flow between different file systems.

USING THE ARPANET FOR REMOTE JOB ENTRY

A very important use of the ARPANET is to give a wide community of users access to specialized facilities. One type of facility of interest is that of a very powerful number-cruncher. Users in the distributed ARPANET community need to have access to powerful machines for compute-intensive applications and the mode of operation most suited to these uses has been batch Remote Job Entry (RJE). Typically, a user will generate a "deck" for submission to a batch system. See Figure 9. He expects to wait for a period on the order of tens of minutes or hours for that "deck" to be processed, and then to receive the usually voluminous output thereby generated. See Figure 10.

As in the case of file transfer, there are a few useful *ad hoc* ARPANET RJE protocols. A standard RJE protocol is being developed to provide for job submission to a number of facilities in the ARPANET. This protocol is being constructed using the TELNET and File Transfer protocols. A scenario which sketches how the protocol provides the RJE in the simplest, most explicit way is as follows:

Via an ARPANET RJE process, a user connects his

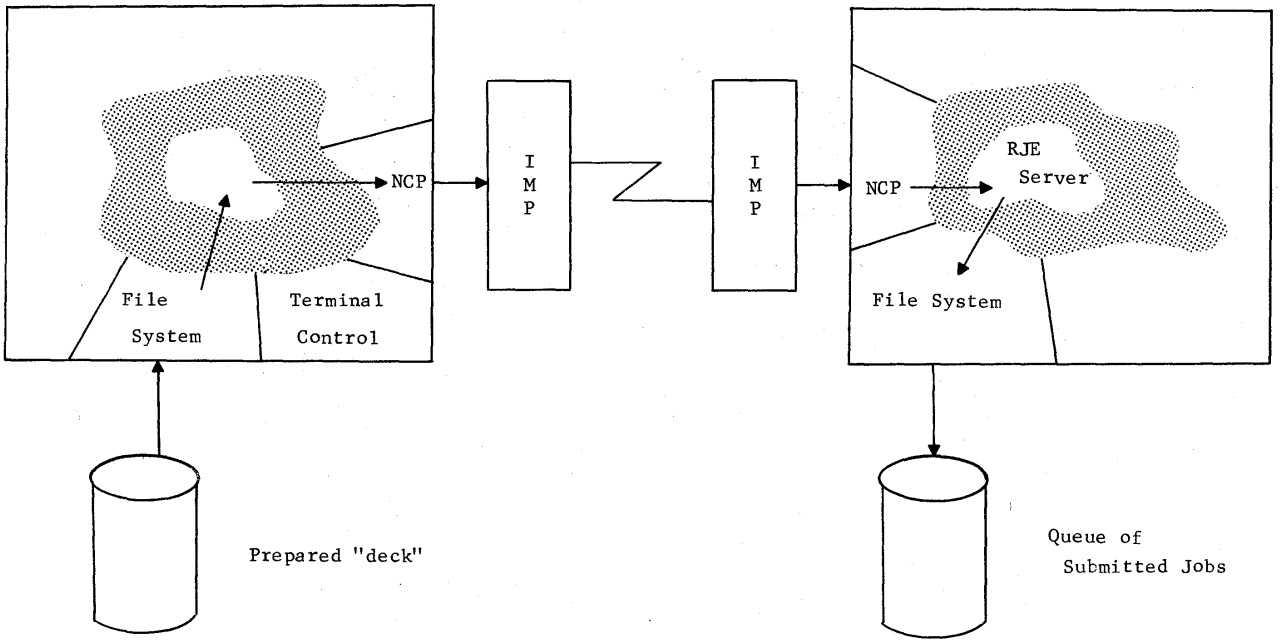

Figure 9—Submission of RJE input

terminal to an RJE server process at the HOST to which he intends to submit his job "deck." Through a short dialogue, he establishes the source of his input and initiates its transfer using the File Transfer Protocol. At some later time, the user reconnects to the appropriate RJE server and makes an inquiry on the

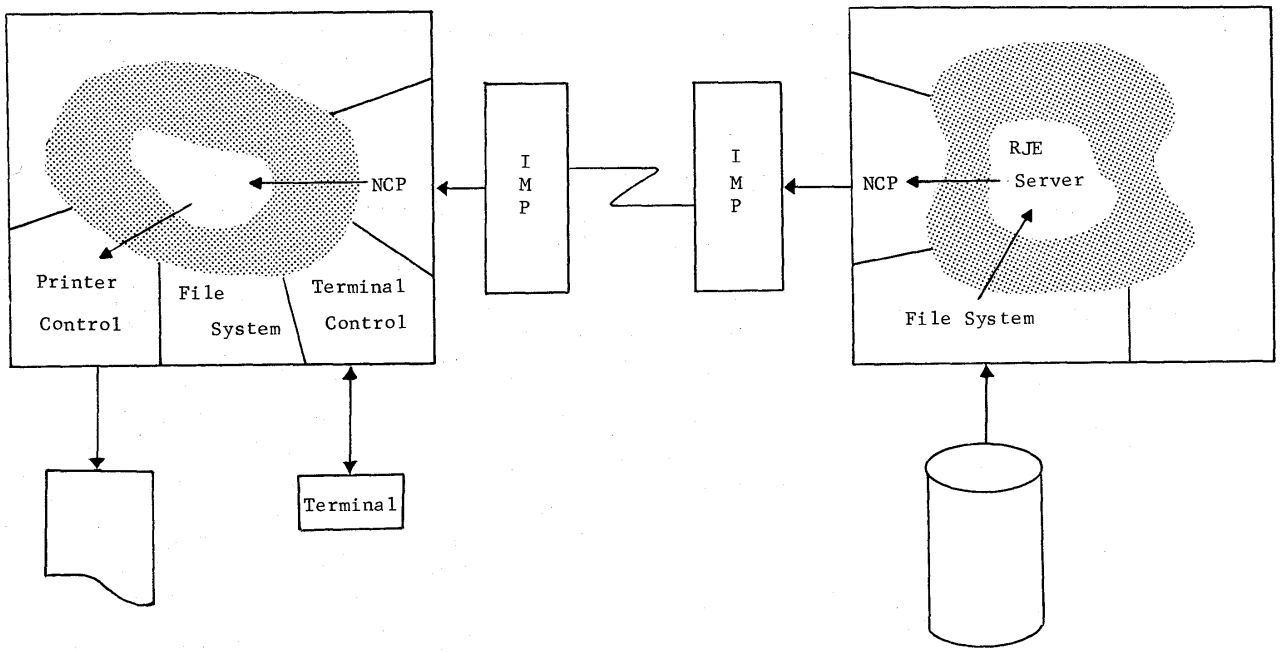

Figure 10—Retrieval of RJE output

status of his job. When notified that his input has been processed, he then issues commands to the serving HOST to transfer his output back.

We can of course imagine more automatic ways of achieving these same functions. A user might need only type a job submission command to his local system. Automatically and invisibly, then, the local system would connect and converse with the specified RJE server causing the desired output to later appear in the users file area or perhaps on a local line printer. The intention is to design the RJE protocol so that the explicit use can start immediately and the more automatic RJE systems can be built as desired.

OTHER PROTOCOLS AND CONCLUSIONS

One of the more difficult problems in utilizing a network of diverse computers and operating systems is that of dealing with incompatible data streams. Computers and their language processors have many ways of representing data. To make use of different computers it is necessary to (1) produce a mediation scheme for each incompatibility or (2) produce a standard representation. There are many strong arguments for a standard representation, but it has been hypothesized that if there were a simple way of expressing a limited set of transformations on data streams, that a large number of incompatibilities could be resolved and a great deal of computer-computer cooperation expedited.

The bulk of protocol work is being done with the invention of standard representations. The TELNET protocol, as discussed, is founded on the notion of a standard terminal called the Network Virtual Terminal (NVT). The File Transfer Protocol is working toward a standard sequential file (a Network Virtual File?). So it is also with less advanced protocol work in graphics and data management.

There is one experiment which is taking the transformational approach to dealing with incompatibilities. The Data Reconfiguration Service (DRS) is to be generally available for mediating between incompatible stream configurations as directed by user-supplied transformations.[13]

ACKNOWLEDGMENTS

Function-oriented protocols have been the principal concern of the ARPANET Network Working Group (NWG). A list of people who have contributed to the development of the protocols discussed would include, Robert Braden, Howard Brodie, Abhay Bhushan, Steve Carr, Vint Cerf, Will Crowther, Eric Harslem, Peggy Karp, Charles Kline, Douglas McKay, Alex McKenzie, John Melvin, Ed Meyer, Jim Michener, Tom O'Sullivan, Mike Padlipsky, Arie Shoshani, Bob Sundberg, Al Vezza, Dave Walden, Jim White, and Steve Wolfe. We would like to acknowledge the contribution of these researchers and others in the ARPA Network Working Group, without assigning any responsibility for the content of this paper.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
AFIPS Conference Proceedings May 1970

2 F E HEART et al
The interface message processor for the ARPA computer network
AFIPS Conference Proceedings May 1970

3 L KLEINROCK
Analytic and simulation methods in computer network design
AFIPS Conference Proceedings May 1970

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA Computer network
AFIPS Conference Proceedings May 1970

5 *Specifications for the interconnection of a Host and an IMP*
Bolt Beranek and Newman Inc Report No 1822
February 1971

6 C S CARR S D CROCKER V G CERF
HOST-HOST communication protocol in the ARPA Network
AFIPS Conference Proceedings May 1970

7 *HOST/HOST protocol for the ARPA Network*
ARPA Network Information Center #7147

8 T O'SULLIVAN et al
TELNET protocol
ARPA Network Working Group Request For Comments (RFC) #158 ARPA Network Information Center (NIC) #6768 May 1971

9 S M ORNSTEIN et al
The Terminal IMP for the ARPA Computer Network
AFIPS Conference Proceedings May 1972

10 J B POSTEL
Official initial connection protocol
ARPA Network Working Group Document #2 NIC #7101 June 1971

11 A K BHUSHAN et al
The data transfer protocol
RFC #264 NIC #7812 November 1971

12 A K BHUSHAN et al
The file transfer protocol
RFC #265 NIC #7813 November 1971

13 R ANDERSON et al
The data reconfiguration service—An experiment in adaptable process/process communication
The ACM/IEEE Second Symposium On Problems In The Optimization Of Data Communications Systems October 1971