THE ARPANET TELNET PROTOCOL:

ITS PURPOSE, PRINCIPLES, IMPLEMENTATION, AND

IMPACT ON HOST OPERATING SYSTEM DESIGN

J. Davidson
Institute for Advanced Computation
Sunnyvale, California

W. Hathaway
Computation Division
NASA Ames Research Center
Moffett Field, California

J. Postel
USC - Information Sciences Institute
Marina del Rey, California

N. Mimno
R. Thomas
D. Walden
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

Preface

The protocol discussed in this paper was developed by many members of the ARPANET community starting in 1969 and continuing through the present. Many individuals and institutions have been members of this community at one time or another over the years. A review of the documents, both working and published, written on the subject of this protocol reveals that the following individuals were among those who contributed to the protocol design: A. Bhushan, R. Braden, R. Bressler, J. Burchfiel, S. Carr, V. Cerf, B. Cosell, D. Crocker, S. Crocker, W. Crowther, J. Davidson, D. Dodds, W. Duvall, G. Grossman, R. Gumpertz, W. Hathaway, W. Kantrowitz, R. Long, J. McConnell, A. McKenzie, R. Merryman, J. Melvin, R. Metcalfe, E. Meyer, N. Mimno, L. Nelson, T. O'Sullivan, M. Padlipsky, K. Pogran, J. Postel, M. Reese, J. Rulifson, R. Schantz, R. Thomas, R. Tomlinson, D. Walden, R. Watson, D. Wells, J. Winett, and S. Wolfe. No doubt others also contributed to the design and dozens of other individuals contributed to the many implementations of the protocol. We acknowledge all of their contributions. Many of the above named individuals were offered an opportunity to collaborate on the writing of this paper. The authors are those who responded. We apologize for the oversight to any individual who would have liked to help write this paper but was not apprised of the opportunity.

1. INTRODUCTION

The ARPANET [1] provides a capability for geographically separated computers, called Hosts, to communicate with each other. The Host computers typically differ from one another in type, speed, word length, operating system, etc. Each Host computer is connected to the network through a small computer called an Interface Message Processor or IMP [2]. The complete network is formed by connecting these IMPs, all of which are virtually identical, by means of leased wideband circuits; thus the IMPs form a subnetwork through which the Hosts communicate. Data is sent through the communications subnetwork in messages up to about 8100 bits long. A Host passes to its own IMP a message which includes the "network address" of a destination Host. The message is then passed from IMP to IMP through the network until it finally arrives at the IMP to which the destination Host is attached, and this IMP passes the message to its Host. It should be noted that any simple terminals accessing the network do so via a Host (even if the Host is operated by the network authority). This Host "local" to the terminal performs message-formatting functions for the terminal; as we shall see it may perform other functions as well.

Specifications exist for the physical and logical message transfer between a Host and its IMP [3]. These specifications are generally called the IMP/Host "protocol". This protocol is not sufficient by itself, however, to specify the methods of communication between processes running in two possibly dissimilar Hosts. Rather, the processes must have some agreement as to the method of initiating communication, the interpretation of transmitted data, and so forth. Although it would be possible for such agreements to be reached by each pair of Hosts (or processes) interested in communication, a more general arrangement is desirable in order to minimize the amount of implementation necessary for network-wide communication. Accordingly, the Host organizations formed a group (called the Network Working Group or NWG) to facilitate an exchange of ideas and to formulate additional specifications for Host to Host communications.

The NWG adopted a "layered" approach to the specification of communications protocols [4,5,6], wherein the higher layers of protocol use the services of lower layers; the advantages and disadvantages of the layered approach are discussed in the references, especially in [6]. As shown in Figure 1, the lowest

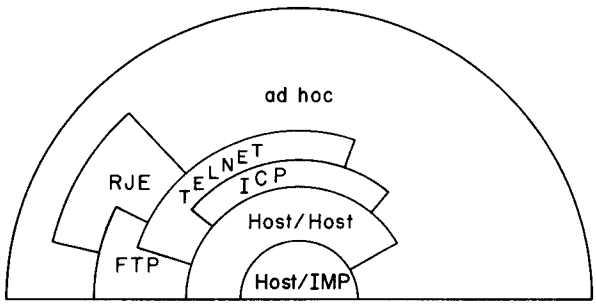

Figure 1 -- Layered Relationship of the ARPANET Protocols

layer is the IMP/Host protocol. The next layer (called the Host/Host layer in the figure) specifies methods of establishing communications paths between Hosts, managing buffer space at each end of a communications path, etc.* Next, the Initial Connection Protocol or ICP [5] specifies a standard way for a remote user (or process) to attract the attention of a network Host, preparatory to using the Host. The ICP provides the analog of the user pressing the attention button at a local terminal on a Host.** In the next layer is the Telecommunications Network or TELNET protocol which was designed to support terminal access to remote Hosts. TELNET is a specification for a network standard terminal and the protocol for communicating between this standard terminal and a Host. The next logical protocol layer consists of function oriented protocols [5], two of which, File Transfer Protocol (FTP) and Remote Job Entry protocol (RJE), are shown in the figure. Finally, at any point in the layering process, it is possible to superimpose ad hoc protocols.

The focus of the present paper is the TELNET protocol. TELNET includes many novel aspects which have not been presented in detail in the open

*Two separate Host/Host protocols have gained wide enough acceptance within the ARPANET community to be called standards. One, the NCP-based protocol [5], has been in widespread use for several years and has been implemented for almost every Host in the network; at this date, almost all data is transmitted through the network via the NCP protocol. The second, the TCP-based protocol [7], is more general than the NCP-based protocol, and is in use by a subset of the network Hosts for certain types of communication. Note that either the NCP- or TCP-based protocol provides a suitable base for the remaining layers of protocol, as might any number of other suitable Host/Host protocols.

**While this protocol was specified at an early date in the network's development, the community has come to understand that a separate protocol for this function is not strictly necessary and, indeed, in the context of the TCP-based Host/Host protocol, the ICP protocol is not used.

literature, and the primary aim of this paper is to provide that previously unavailable description and discussion. In the following pages we sketch the evolution of the TELNET protocol (Section 2), present its principles in detail (Section 3) and discuss its implementation (Section 4). Throughout these sections we attempt to point out, wherever possible, the impact the TELNET protocol and its capabilities have had on Host operating system design.

2. THE EVOLUTION OF TELNET

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

As early as 1969 a few Hosts had been programmed on an ad hoc basis to permit terminal access from another Host [8]. In 1971 an NWG subcommittee was formed to consider the general problem of supporting interactive use of arbitrary Hosts by users at arbitrary remote terminals. There was great controversy in the committee discussions, focusing on four issues: character set, connection establishment, echoing, and interrupt capability. By late 1972 there was enough consensus so that widespread implementation of an early version of the TELNET protocol had been accomplished.

The early version was based on the idea of an asymmetric interactive connection between the so-called "server Host" providing interactive computation and the so-called "user Host" to which the remote terminal was attached. The four issues mentioned above were dealt with as follows. A standard network terminal known as the Network Virtual Terminal (NVT), using a 7-bit ASCII character set, was adopted for data transmission over this logical connection; the ICP was selected as the means for establishing the connection; some special control commands were provided the user so that it could instruct the server to echo or not echo individual characters; and an innovative "synch" strategy was developed for alerting the Server TELNET that a "special" character had been sent by the user. This last facility, described in detail in section 3.1.3, allows the user to bypass a potentially clogged user-to-server data path when attempting (for instance) to stop an errant process.*

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

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

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

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

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

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

There was such enthusiasm for the new version that a schedule for "rapid" (within the year) implementation was laid out. However, the implementation of the new TELNET protocol proceeded more slowly than expected. Only in the past year have implementations been widely available. In retrospect, there were several reasons for the delay in the implementation: 1) at the time the revised protocol implementation was scheduled, implementation of the initial version had been completed and Host system managers had not budgeted resources for a second implementation; 2) about this time ARPA's research interest in the network was declining and the network was entering a period of status quo operation; 3) despite initial belief that a clean method of phasing over from the initial protocol to the revised protocol existed, none was found by most implementors and consequently most chose to provide a complete implementation of the revised protocol to operate in parallel with the initial protocol; and, 4) implementation for the most prevalent user Host, the TIP [9], proved to be very difficult (because of the TIP's limited memory) and time consuming, thus implicitly relieving pressure on the server Hosts to implement the revised protocol.

At the time this is being written, in early 1977, the new TELNET protocol has been the accepted standard for several years, and it is widely implemented and used. We believe that the (new) TELNET protocol has many advantages over alternative methods and represents an advance in the art of computer communication. The rest of this paper describes the current version of the TELNET protocol in some detail in the hope that others may benefit from our experience.

3. PRINCIPLES

The purpose of the TELNET Protocol is to provide a general, bi-directional, character-oriented communications facility. Its primary goal is to define a standard method for interfacing terminal devices to terminal-oriented processes.* The protocol may also be used for terminal-to-terminal and process-to-process communication.

The TELNET Protocol is built upon three main ideas: the concept of a "Network Virtual Terminal"; the principle of negotiated options; and a symmetric view of terminals and processes (which allows the protocol to easily and naturally support terminal-to-terminal and process-to-process communication). The remainder of this section discusses the first two principles in detail. The benefits of symmetry are illustrated in the section on option negotiation.

3.1 The Network Virtual Terminal

A TELNET connection consists of a full duplex connection (provided by the Host/Host protocol layer) over which passes data interspersed with TELNET control information.

When a TELNET connection is first established, each end is assumed to originate and terminate at a "Network Virtual Terminal", or NVT. An NVT is an imaginary device which provides a standard, network-wide, intermediate representation of a canonical terminal. This eliminates the need for server and user Hosts to keep information about characteristics of each other's terminals and terminal handling conventions. All Hosts, both user and server,

*Implicit in the concept of the TELNET protocol was the fact that each implementation of the protocol would have to find a method of mapping the NVT and the TELNET protocol functions to and from the particular system's actual terminals, input/output facilities, and operating system functions. Since the mapping of the TELNET protocol and NVT to the local operating system and terminals is not explicitly treated in the TELNET protocol specification, we shall not treat it explicitily in this paper. However, problems and solutions relating to this mapping will be noted at a number of points throughout this paper.

*[10] has previously considered the related issue of specifying the functional characteristics of a typewriter-like time-sharing terminal.

map their local device characteristics and conventions so as to appear to be dealing with an NVT over the network, and each can assume a similar mapping by the other party. The NVT is intended to strike a balance between being overly restricted (not providing Hosts a rich enough vocabulary for mapping into their local character sets) and being overly inclusive (penalizing users with modest terminals).

The Network Virtual Terminal is a bi-directional character device with a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data which is sent over the TELNET connection and, if "echos" are desired, to the NVT's printer as well. "Echos" will not be expected to traverse the network. (See Sections 3.2 and 3.4.4 for further disucssion of echoing.) The code set is 7-bit ASCII in an eight-bit field, with some exceptions noted below. Any code conversion and timing considerations are local problems and do not affect the NVT.

3.1.1 Transmission of Data

To accommodate the needs of the largest possible segment of the user community, the Network Working Group chose to attribute some very fundamental properties to the NVT. One of these properties requires that even though a TELNET connection is intrinsically full duplex, the NVT (in default mode) is considered to be a half duplex device,* and that both user and server communicants must provide a "turn-the-line-around" indication, via the TELNET GO-AHEAD (GA) command, whenever it switches from an output to an input attitude. Hosts that wish may agree, via option negotiation, to operate in character-at-a-time and full-duplex mode.

This property of the NVT makes use of true half-duplex devices possible in a TELNET conversation and does not penalize full-duplex ones. For example, it allows a User TELNET to properly control the mechanical keyboard locking apparatus of an IBM 2741 terminal, which it would otherwise be unable to do since it would have no reliable indication of when the output from the remote serving Host had completed (end-of-line is normally insufficient).

Since the NVT is basically a half-duplex device, it is acceptable for the TELNET which forwards "keyboard input" to accumulate text for transmission until it is willing to relinquish the line. This is consistent with the fundamental property that echoes do not traverse the network. In addition, it has a beneficial effect on buffer considerations in the receiving Host, and it reduces the cost associated with processing multiple network input interrupts. Since many systems take some processing action at an end-of-line (even line printers or card punches tend to work this way), transmission should be triggered at the end of a line. In addition, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line. Therefore implementors are advised to provide a method of signalling when buffered data should be transmitted.

3.1.2 Standard Representation of Control Functions

In its purest formulation, the TELNET protocol makes no assumptions about the process which interfaces the NVT at either end of a connection. This is the characteristic which allows uniform treatment of terminal-to-process, terminal-to-terminal, and process-to-process conversations. However, to account for the fact that in many cases one controlling process will be human, certain mechanisms were introduced into the protocol which appear to have the most benefit for human participants. These mechanisms, while perfectly general in the abstract, serve mainly to standardize the interface through which the (human) user perceives his serving process or his serving Host.

Interestingly, the functions required by humans for controlling a process through a character interface are common to most serving systems, but the means for invoking a given function may vary widely from server to server (Host to Host). The TELNET protocol thus defines a standard representation for each of several functions so that each may be selectively invoked at an arbitrary Host (if that function is implemented) without requiring the user to know the particular Host's convention for invoking the function. (While the TELNET protocol acts to shield a human user at his User Host from some of the variations in operation of different Server Hosts, the nature of the user interface to the TELNET protocol and NVT varies from User Host to User Host.)

The User TELNET must provide a method for the user of an NVT to cause the requests described below to be generated.

INTERRUPT PROCESS provides a function which interrupts the operation of a remote process (See Section 3.1.3). This function is frequently used when a user believes his process is looping, or when he has inadvertantly activated an unwanted process.

ABORT OUTPUT provides a function which allows a process, which is generating output to run to completion (or to reach the same stopping point it would reach if run to completion) without sending the output to the user's terminal. Further, this function typically clears any output already produced but not yet actually printed on the user's terminal.

ARE YOU THERE provides a function which gives the user some perceptible (e.g., printable) evidence that the system is still up and running. This function may be invoked by the user when the system is unexpectedly "silent" for a long time, because of a computation of unanticipated (by the user) duration, an unusually heavy system load, etc.

ERASE CHARACTER provides a function which deletes the preceding undeleted character or "print position" from the stream of data supplied by the user. This function is typically used to edit keyboard input when typing mistakes are made.

ERASE LINE provides a function which deletes all the data in the current "line" of input. This function is typically used to edit keyboard input.

The spirit of these "extra" keys is that they should represent a natural extension of the mapping between "NVT" and "local". For example, just as the NVT data byte 104 (octal) should be mapped into the local code for "uppercase D", the ERASE CHARACTER character should be mapped into the local "Erase Character" function (if such a function is locally implemented).

3.1.3 The TELNET "Synch" Signal

Most time-sharing systems provide mechanisms which allow terminal users to regain control of "runaway" processes; the INTERRUPT PROCESS and ABORT OUTPUT functions described above are meant to invoke these mechanisms. When a terminal is attached directly to such a system, the system has access to all of the terminal's generated signals, whether they are normal characters or special "out-of-band" signals such as those supplied by the Teletype "BREAK" key or the IBM 2741 "ATTN" key, and can react to each immediately to provide the indicated function. This is not necessarily true when terminals are connected to the system through the network, since the network's flow control mechanisms may cause such a signal to be buffered elsewhere, such as in the user's Host.

The TELNET "Synch" mechanism was developed to handle this problem. A Synch signal consists of a Host/Host protocol INTERRUPT signal [5] coupled with a TELNET DATA MARK command. The Host/Host protocol INTERRUPT command is not subject to the normal flow control for TELNET connections. When one is received, it invokes special handling of the TELNET data stream. In this mode, the data stream is immediately scanned for "interesting" signals and intervening data is discarded. "Interesting" signals which are processed in this mode include: the TELNET INTERRUPT PROCESS, ABORT OUTPUT, and ARE YOU THERE characters; local analogs (if any) of these standard characters; all other TELNET commands; and other site-defined signals which can be acted on without delaying the scan of the data stream. The TELNET DATA MARK command is the data stream synchronizing mark. It indicates that any special signals have already been received and that the recipient can resume normal data stream processing. When a DATA MARK arrives before its associated INTERRUPT, the recipient should defer processing the data stream further until the matching INTERRUPT is received. This insures that the two ends of the connection remain synchronized. (For further discussion of this subtle mechanism see [4,5].)

3.1.4 The NVT Printer and Keyboard

The NVT printer has an unspecified carriage width and page length and can produce representations of all 95 ASCII graphics. Of the 33 ASCII control codes and the 128 uncovered codes, the following have specified meaning to the NVT printer: NUL, which produces

*In this discussion we use the term "half duplex" to mean a situation in which data is allowed to flow in only one direction at a time between the parties at the two ends of a TELNET connection, regardless of the type of physical link used between a terminal and its local Host. "Full duplex" is used to mean that characters can flow in both directions simultaneously on the connection.

no-operation; LF, which moves the printer to the next print line; and CR, which moves the printer to the left margin. In addition, a few codes have defined but not required effects on the NVT printer: BEL, BS, HT, VT, and FF. All remaining control codes cause the NVT printer to take no action. The NVT does not have any cursor, cursor positioning, graphics, or other sophisticated terminal capabilities (although these can be added to TELNET via the option negotiation facility described below).

The sequence "CR LF", as defined, causes the NVT to be positioned at the left margin of the next print line (as would, for example, the sequence "LF CR"). Many systems and terminals do not treat CR and LF independently, and have to go to some effort to simulate their effect. (For example, some terminals do not have a CR independent of the LF; on some such terminals it is possible to simulate a CR by backspacing.) Therefore, the sequence "CR LF" is treated as a single "new line" character and used whenever their combined action is intended; the sequence "CR NUL" is used where a carriage return alone is actually desired; and the CR character is avoided in other contexts.

The NVT keyboard has keys for generating all 128 ASCII codes and the TELNET special commands. (Some of these may be generated by combinations or sequences of keystrokes on the actual terminal.) Note that although many have no effect on the NVT printer, the NVT keyboard is capable of generating them.

3.2 Option Negotiation

The principle of negotiated options takes cognizance of the fact that many sites wish to provide additional services over and above those available within an NVT, and that many users have sophisticated terminals and prefer elegant, rather than minimal, service. Various "options" are provided within the TELNET protocol to allow a user and server to agree upon more elaborate (or perhaps just different) conventions for their TELNET connection. Options may be invoked to specify the character set, the echo mode, the line width, the page length, etc.

The basic protocol for enabling an option is for either party (or both) to request that the option take effect. The other party may then either accept or reject the request. If the request is accepted, the option immediately takes effect. If it is rejected, the associated aspect of the connection remains as specified for an NVT. Since all parties must be prepared to support the NVT, a party may always refuse a request to enable, and must never refuse a request to disable, an option.

A flurry of option requests is likely to occur when a TELNET connection is first established, as each party attempts to obtain the best possible service from the other. Beyond that, options can be used to dynamically modify the characteristics of the connection to suit changing local conditions. For example, the NVT, as previously explained, uses a transmission discipline well suited for line-at-a-time applications but poorly suited for character-at-a-time applications. A server electing to devote the processing overhead required for character-at-a-time operation may (when it is suitable for a local process) negotiate into character-at-a-time mode. However, rather than permanently burden itself with the extra processing overhead, it may switch (i.e., negotiate) back to line-at-a-time when the "tighter" control is no longer necessary.

In the following, we use the example of echoing to motivate and illustrate the principles of option negotiation.

A basic observation to be made regarding echoing is that Hosts which supply interactive services tend to be optimized either for terminals that do their own echoing or for terminals which do not, but not for both terminal types. Therefore, a set of echoing conventions which would prohibit a server from initiating a change in echo mode would be excessively confining. Servers would be burdened with users who are in the "wrong" mode, in which they might not otherwise have to be, and users would be burdened with remembering proper echoing modes.

TELNET echo mode negotiation is based on three assumptions. First, both the server and the user should be able to suggest the echo mode. Secondly, all terminals must be able to provide their own echoes, either internally or by means of the local Host. Thirdly, all servers must be able to operate in a mode that assumes that remote terminals provide their own echoes. The last two assumptions result from the desire for a universal, minimal basis upon which to build.

An implementation based on these rules has, in effect, the following commands (the actual commands are presented at the end of this section):

- 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 won't echo to you";
- NO ECHO, when sent by the user to the server, means "Don't you echo to me".

Whenever a TELNET connection is opened between a user and a server, both user and server must assume that the user is echoing locally. If the user would prefer the server to generate echoes, it can send the server an ECHO command. Or, if the server would prefer to do its own echoing, it can send the user an ECHO command. The recipient of an ECHO command is not required to change the way it handles echoing, but it may have to respond to the command. If the requested mode of operation is acceptable, the recipient begins operating in that mode; if "beginning" means changing from a previous mode, the recipient must also respond with the ECHO command to indicate that (and when) the changeover took place. If the requested mode of operation is not acceptable, the recipient must respond with the command's inverse to indicate its refusal (this must be NO ECHO, since neither party is allowed to refuse a change into NO ECHO).

Several properites of this scheme are worthy of note:

- NO ECHO is retained as the nominal mode; a connection will operate in ECHO mode only when both parties agree.
- 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.
- Servers are free to specify their preferred mode of operation; thus users, human or machine, need not learn the proper mode for each server.

As described so far, the interpretations of the ECHO command ("I'll echo to you" and "You echo to me") imply that both the server and user know which is which. This is a problem for connections where there is no clearly identifiable user or server, such as connections for linking terminals together. Bearing this in mind, one comes to understand that there are five reasonable modes of operation for echoing on a connection, as shown in Figure 2, and that four commands are sufficient to deal with completely

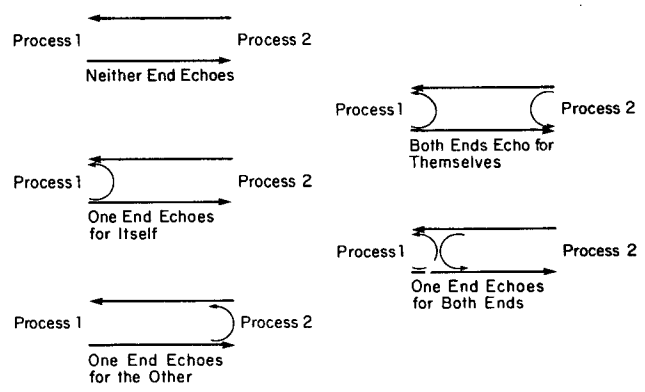

Figure 2 -- Five Echoing Modes

symmetric echoing. We have already mentioned the four commands: the two possible meanings of 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 WON'T ECHO TO YOU. Echoing is now the negotiation of two options for which the initial, default modes are DON'T ECHO TO ME and I WON'T ECHO TO YOU.

Actually, four basic commands (DO/DON'T/WILL/WON'T) are provided to support negotiation of any option, echoing included. WILL XXX

is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and DON'T XXX being its positive and negative acknowledgments. Similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing OPTION XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the default NVT is what remains when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, a Host TELNET implementation may be totally unaware of options it chooses not to support; it may simply refuse any option request that cannot be understood.

3.3 Command Encoding

Every TELNET command is a sequence of at least two bytes: an "Interpret as Command" (IAC) character followed by the code for a command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option. This format was chosen so that as more comprehensive use of the "data space" is made -- by negotiations from the basic NVT -- collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and inefficiency, of preceding the data bytes with an escape character to indicate that the bytes are data rather than commands. With this encoding scheme, only the IAC need be doubled when it is sent as data, and the other 255 codes may be passed transparently. The TELNET command set includes the following commands: IAC (indication that the next byte is a command), "WILL" option, "WON'T" option, "DO" option, "DON'T" option, indication that the following bytes concern the subnegotiation (discussed below) of the given option, termination of the bytes concerning subnegotiation of an option, the Go Ahead (GA) signal, the ERASE LINE, ERASE CHARACTER, ARE YOU THERE, ABORT OUTPUT, and INTERRUPT PROCESS functions, BREAK, DATA MARK, and NOP.

3.4 The Options

Because it is envisioned that options which prove to be generally desirable will eventually be supported by many Hosts, a system has been provided for coordinating the assignment of option codes and for carefully documenting and publishing options. The system also provides for temporary and experimental use of options and for the use among just a few Hosts of options which are not widely desired. Figure 3 lists currently "registered" TELNET options.

No.	Name
0	Binary Transmission
1	Echo
2	Reconnection
3	Suppress Go Ahead
4	Approximate Message Size
5	Status
6	Timing Mark
7	Remote Controlled Transmission and Echoing (RCTE)
8	Output Line Width
9	Output Page Size
10	Output Carriage-Return Disposition
11	Output Horizontal Tabstops
12	Output Horizontal Tab Disposition
13	Output Formfeed Disposition
14	Output Vertical Tabstops
15	Output Vertical Tab Disposition
16	Output Linefeed Disposition
17	Extended ASCII
255	Extended Options List

Figure 3 -- Registered Options

The currently defined options fall into three general classes: options to control or take into account characteristics of particular physical terminals; options which affect the operation of TELNET; and options which affect the operation of Host operating system modules or user processes. Naturally, there is some overlap between these categories of options, as will be seen below.

3.4.1 Controlling the Physical Terminal

Options 8 to 17 in Figure 3 are aimed primarily at controlling or accommodating the characteristics of various physical terminals which are in use in the network.

The Output Line Width option is representative of these options. There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about output line width:

1. the data sender wishes the receiver to use its (the receiver's) knowledge of the printer width to properly handle the line width;
2. the data receiver wishes the sender to use its (the sender's) knowledge of the data being sent to properly handle the line width;
3. the data sender wishes to use its knowledge of the data being sent to instruct the receiver in the proper handling of the line width; and
4. the data reciever wishes to use its knowledge of the printer to instruct the sender in the proper handling of the line width.

One example of line width handling is for the receiver to "fold" lines sent by the sender so that all characters in a line fit on the printer page. Another example might be to not fold lines even if they overflow the printer page should that be what the user desires (e.g., it is better to see only the left half of a chart or picture than to have the left and right halves intermixed. The option definition specifies commands that allow the command sender (which may be either the data sender or receiver) to 1) suggest that the command sender alone handle output line width considerations, 2) suggest that the other party handle line width considerations but with a suggested line width value (up to a fairly large number) for the other party to use, 3) suggest that the other party alone handle line width considerations but with a suggested line width of infinity, and 4) suggest that the other party alone handle line width considerations with no suggestions about how it be done. The commands are defined such that if neither data sender nor receiver wants to handle output line width considerations, the data receiver (which is presumed to have local knowledge of the printer) does whatever gets done. Should both want to handle output line width considerations, the handling is done by the data sender, which is presumed to have special knowledge about the data, but taking into account any suggestions the receiver makes. Notice again the value of the principle of symmetry.

3.4.2 Controlling TELNET

Options 0, 2, 3, 5, and 255 in Figure 3 are largely concerned with controlling use of TELNET connections. The Binary Transmission option provides a method for sending transparent binary data over a TELNET connection without resort to a higher level data transmission protocol. The Reconnection option provides a method of moving one or both ends of a TELNET connection from one Host to another; there are a variety of cases when this is useful (see [11] and pp. 81-90 of [5] for examples). While the NVT nominally follows a line-buffered mode protocol complete with Go Ahead (GA) signal, there is no reason why a full duplex connection between a full duplex terminal and a Host optimized to handle such terminals should be burdened with this protocol. The Suppress Go Ahead option provides a method of switching to the full-duplex mode of operation when possible. The Status option allows the party at one end of a TELNET connection to obtain the status of options as seen by the party at the other end of the connection. The Extended Options List option provides an expansion capability beyond the 256th option code.

3.4.3 Controlling the Operating System and Processes

Options 1, 4, and 6 from Figure 3 deal largely with controlling the operating system and processes. The Echo option has already been discussed.

The Approximate Message Size option provides a mechanism whereby the parties involved can attempt to agree on the size of messages to be transmitted over the connection. For instance, the knowledge that a transmitter will never send messages greater than a maximum size could be used by a receiver to more efficiently utilize its input buffer space.

The Timing Mark option provides a way for a user or process at one end of a TELNET connection to be sure that previously transmitted data has been completely processed, printed, discarded, or otherwise handled. This is useful for timing or synchronizing events (see pp. 101-104 of [5]).

3.4.4 RCTE

Option 7, the Remote Controlled Transmission and Echoing (RCTE) option, is one of the more elaborate TELNET options and combines the functions of all three option classes. This section discusses the motivation for RCTE and sketches its operation.

The ARPANET, like other communication networks, introduces a delay when transporting data from one point to another. In the ARPANET, this delay may be caused by a combination of factors including user and server system loads, network configuration,

retransmission, and satellite delays. Although most communicating programs do not know or care about delays, most users do. The round-trip delay for characters echoed by a serving system, for example, can be agonizingly apparent to a typist.

Consider as an example a system with a highly interactive command language interpreter which supports command recognition and completion. A user of such a system might type the following character sequence to copy one file (ABC) to another (XYZ):

CO[esc]ABC[esc]XYZ

The command language interpreter together with the terminal handling software would respond with the terminal printout:

COPY (FROM FILE) ABC (TO FILE) XYZ

(where the system printout is underlined) regardless of the speed at which the user types. The printout is a mixture of echoes to the user's type-in and responses by the command language interpreter. Propagation delays could render such highly interactive dialogues useless to remote users if echoes had to be relayed through the network from the server Host to the user's terminal.

It was felt necessary to develop a distributed terminal control protocol which could hide propagation delay for terminal interactions and could provide more efficient operation than sending character echoes across the network. The basic strategy developed is to distribute the responsibility for echoing between the user and server sites. The Server TELNET (in conjunction with the serving Host's terminal management software) decides generally what to echo and when to echo it, while the User TELNET generates the actual echoes. Because no server-to-user echoes are ever transmitted through the net, all echoing is performed at the instant the user expects to see it by the User TELNET. Thus propagation delay for echoes is no longer an issue. Any delay observed is due to the time required by the server to generate responses and by the network to transmit them to the user.

To develop this scheme, we began with a model for the echoing mechanism which might be employed by a non-distributed operating system to control full duplex terminals. This model assumes the existence of two distinct code modules which are together responsible for character stream management within the operating system. These are the "terminal component" which exchanges data and control information with the device (and executes as part of the interrupt logic), and the "process component" which exchanges data and control information with the serving process (and executes as a privileged extension of the user code).

These two components effect the proper integration of echoes with process outputs in the following way. The terminal component is designed to distinguish a few out of all the possible subsets of characters (for example, alphabetic, or numeric, or punctuation characters, etc.). The serving process is designed to use the members of one or more of these subsets as delimiters which mark the end of a user input. These delimiting characters are called "break" characters.

The process (via its privileged extension) tells the terminal component the subsets which collectively define its break character set, and instructs it to start echoing. The terminal component then places each incoming character into an input buffer (from which it will go to the process) and into an output buffer (from which it will go to the terminal as an echo). When the terminal component encounters a break character in the input, it suspends echoing while the serving process analyzes the input and responds. When the process has completed its response, it again issues a read. If the input buffer is empty, the terminal component is asked to resume its echoing function. If it is not empty, a (deferred) echo should be sent to the output buffer for each character read by the process, until the buffer is empty.

An important design decision is whether the terminal component or the process component should be responsible for generation of deferred echoes. In many existing systems the process component performs this function by providing an echo for each character it moves from the input buffer to the process workspace (after the initial break, and up until the input buffer empties). Immediate echoing is then resumed by the terminal component as before.

This solution is natural in a non-distributed environment since it is not immediately apparent, for example, how the (interrupt-driven) terminal component should be invoked to do this task. However, it introduces considerable complexity when the terminal and process components are separated from each other as in the ARPANET environment. The problems with this approach are discussed in detail in [12] and [13], and arise mostly because considerable synchronization is required for the distributed components to switch from deferred to immediate echo status. The fact that unechoed characters may be in the inbound (user-to-server) pipe, while the command to resume immediate echoing is in the outbound pipe, makes the synchronization untidy.

It is possible to design a system in which the terminal component is responsible for generation of deferred echoes. The way we choose to model this approach is to have the terminal component maintain two distinct input buffers, one for process input and one for unechoed characters. As shown in Figure 4,

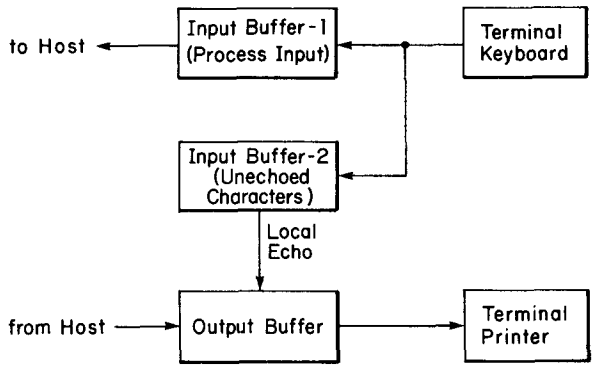

Figure 4 -- Operation of a Remote Controlled Echoing System

characters are placed into both buffers simultaneously (as they arrive from the terminal) and are removed to their respective destinations (the process workspace and the output buffer) in response to a read operation. The process component is virtually unchanged from what it was before, except that it need no longer worry about deferred echoes and instead must signal the terminal component when it attempts a read.

This alternate design approach adapts readily to the network environment. The terminal component can be implemented at the user site precisely as it was in the integrated system. The only difference is that now, in order to place its characters into the process buffer, it must send them through an inbound "pipe." Since it retains a copy of all keyed characters in its own buffer, it can provide both immediate and deferred echoing whenever it receives indication of a process read.

One reason we have assumed that an operating system might choose to perform these echoing tasks on a process's behalf is to avoid awakening the process at each and every character arrival. Of course, even when there is no echoing to be done it is still desirable to defer process activation until there is something significant for the process to do. In line with this, our model provides another facility by which a user process may designate when enough input has accumulated so that a significant amount of computing can be performed. It does this by specifying a "wakeup" character set in the same way it specifies the break character set. A wakeup request is delivered to the scheduler whenever a wakeup character is recognized.

In the network environment, the concept of wakeup characters can be used to help achieve better user-to-server channel utilization. If the process at the server site is not going to be awakened until receipt of a wakeup character, a User TELNET need not transmit any input until keying of a wakeup character. Thus it makes sense from a channel utilization viewpoint to pass the wakeup character set description along with the break character set description to the User TELNET. Wakeup characters have been dubbed "transmission" characters for the purpose they serve in the ARPANET.

This transmission and echoing strategy is that invoked by the TELNET RCTE option. The option specification defines the necessary commands for sending wake-up sets, etc., between the process and terminal components. For further detail see [5], pages 105-117.

4. IMPLEMENTATIONS

This section discusses some of the approaches taken in the implementation of User and Server TELNET and enumerates some of the implementation problems.

4.1 Approaches to Integration into the Operating Systems

With very few exceptions the Hosts interfaced to the ARPANET were existing systems designed with no thought of interconnection to other autonomous Hosts, and certainly not to one under different administrative control. As noted previously, each Host was designed to work with a specific class or set of terminals with a fairly narrow range of properties, for example, hardcopy line-at-a-time terminals, character-at-a-time terminals, or (in a few cases) display terminals. In each of these systems there is a portion or module of the operating system that allows applications programs or processes to interact with terminals. This module, which may be called the terminal control module or TCM, implements certain system calls that application programs can use to read (or write) a character or line from (to) a terminal. In the more flexible systems there may be system calls to set echo modes, to invoke character set translations, and so on. The TCM contains code to interact with the terminal as a device, and to control character buffering if necessary. Generally, applications programs use the system call interface to the TCM to interact with terminals.

Consider a program implemented to interact with local terminals that was written before the network existed. For that program to be used via TELNET, the server TELNET must present an interface to the program identical to the interface presented by the TCM. Since the TCM is usually operating system code, the Server TELNET must be at least partly operating system code.

The impact of this observation is that the most effective implementation approach is to integrate the TCM and Server TELNET into an expanded module that interacts both with terminal devices and with the Host/Host protocol module in the operating system. The expanded module (TCM plus Server TELNET) can be thought of as providing application program interfaces for both real and pseudo terminals (NVTs).

This reorganization of the TCM has in some cases been difficult since the NVT presented to the TCM may be quite different from the real terminals that the TCM was originally designed to control. In addition, it should be noted that the TCM for timesharing systems often plays a role in the startup and termination of user sessions. Typically, when a user strikes a particular character on an otherwise idle terminal, that character is interpreted as a signal to start a new session. This special session starting code must also be invoked when a TELNET connection is established.

4.1.1 TENEX Approach

TENEX is a time-shared operating system for the DEC PDP-10 processor [14]. At present there are 18 TENEX systems connected to the ARPANET as Hosts. ARPANET TENEX Hosts provide both Server TELNET, which supports terminal access to TENEX for remote Users, and User TELNET, which supports terminal access to remote Hosts for local users.

TENEX Server TELNET is implemented by a combination of system level software (code embedded in the operating system) and user level software (unprivileged code which executes under the control of the operating system). When a remote user attempts to gain terminal access to TENEX, a user level process is activated. This process acts to complete the initial connection protocol (ICP) exchange initiated by the user in order to establish a pair of standard ARPANET Host/Host protocol connections between TENEX and the remote user's Host. Next, the process instructs TENEX (via an operating system call) to treat the new connection pair from then on as a TELNET connection. The system level program responds by creating a new "pseudo terminal". From that point, TENEX acts to insure that the pseudo terminal appears to be an NVT to the remote Host and to be a local terminal to local processes. Finally, the user level program passes the pseudo terminal off to the standard TENEX software that handles terminals in a pre-login state. Until the connection with the remote Host is broken, the pseudo terminal and its remote user are treated no differently by local processes than a local terminal and user would be.

TENEX User TELNET is an unprivileged user level program which users invoke in the same way they invoke other TENEX "subsystems", such as text editors and language processors. The User TELNET program operates in two modes: in command mode it is responsive to user commands such as those to establish and break connections, to initiate option negotiations, etc.; in transparent mode it acts to pass characters between the user's terminal and a remote Host. The program allows a user to have several active connections to remote Hosts and to switch his attention (and terminal) back and forth among them. The program itself performs all of the necessary TELNET protocol actions including initiating ICP exchanges, observing NVT conventions for data transfer, and negotiating TELNET options. In particular, from its point of view the connections it uses for communicating with remote Server TELNET modules are two general purpose ARPANET Host/Host protocol connections. Although the operating system imposes no constraints on how these connections are used, the program, of course, uses them in the manner required by the TELNET protocol.

4.1.2 TSS/360 Approach

TSS/360 is a virtual memory time-sharing system for the IBM System/360 Model 67 and IBM System/370 computers [15]. Note, however, that although there are several 360 and 370 systems on the ARPANET, fundamental differences between TSS and other IBM operating systems make this implementation discussion relevant to TSS only.

The TSS NCP implementation attempts to obey the layering of ARPANET protocols exactly. There are distinct user language interfaces (macros or procedure calls) available for Host-Host, TELNET, and File Transfer protocol levels, with strict layering (e.g., TELNET uses standard Host-Host macros, File Transfer uses standard TELNET macros). This, plus the fact that the TSS terminal control module (called GATE) appears exactly the same for both conversational and nonconversational (batch) jobs, greatly simplified both User and Server TELNET implementation.

In TSS the first contact from a user's terminal (e.g., dialing in or hitting the attention key) immediately creates a server task, which will then accept and validate a LOGON command. This approach was rejected in designing Server TELNET, however, in order to provide additional facilities present on many ARPANET Hosts (e.g., pre-LOGON system status). Thus a newly opened TELNET connection is first handled by one of several available "logger" tasks, which allow several pre-LOGON functions. When a LOGON command is recognized, a server task is created (the same as for normal TSS) and the TELNET connection is released by the logger task, and passed to the server task at the point at which it would normally "attach" the user's physical terminal. Whenever programs running in a Server TELNET task attempt I/O to the "terminal" (i.e., use GATE), appropriate TELNET macros are used internally by GATE rather than normal system calls; the programs themselves need not know whether they have a real terminal or an NVT (or for that matter, a batch input/output file). This approach facilitates implementing any server function (e.g., file transfer) which is built on TELNET, as it does not even require recompilation to move from debugging with a real terminal to operational use with an NVT.

TSS User TELNET is simply a user level program which interfaces with the user's real terminal on one side (using GATE reads and writes) and with the distant server task on the other (using TELNET reads and writes). This program allows initiation, termination, and control of multiple simultaneous connections, as well as the ability to generate the various TELNET control functions (e.g., INTERRUPT PROCESS). Character set mappings, etc., are handled by the TELNET macros themselves. The program also attempts to do the translation between the very limited support offered by TSS to real terminals (essentially limited to line-at-a-time half duplex protocol) and the more general NVT.

4.1.3 MULTICS Approach

Multics is a general purpose, time-shared operating system for the Honeywell 6000 series processor [16]. There are several Multics Hosts on the ARPANET. Multics supports both User and Server TELNET.

Server TELNET is integrated into the system through the Multics "answering service", a module whose function is to answer data set calls and other attempts to access Multics. It executes as a Multics User process (in a normal user ring) with special capabilities. When it answers a "call", the answering service attempts to authenticate the caller as an authorized Multics user. If the authentication succeeds, the answering service creates a new process (job) for the user and passes the data set line off to the new process for use as its primary I/O or control stream. From that point, access to terminal I/O functions for the new user process is through a TTY IOSIM (I/O system Interface Module) which also executes in the user's ring. This TTY IOSIM interacts with the

terminal device through a TTY DIM (Device Interface Module) which executes in a privileged system ring. When the "call" comes from the network, the answering service completes the ICP and then proceeds to authenticate the user. If authentication is successful, an NVT is passed off to the user's newly created process for use as its primary I/O stream. The TELNET protocol for the NVT is performed by a Server TELNET IOSIM which executes in the user process ring. In addition, this Server TELNET IOSIM acts much like the "normal" TTY IOSIM to provide the user process access to standard terminal (NVT) I/O functions. The Server TELNET IOSIM interacts with the network through the IMP DIM which (like the TTY DIM) executes in a privileged system ring.

The Multics User TELNET is provided by a User TELNET program which executes in a normal user ring. The User TELNET program interacts with the NCP to establish and break connections with server Hosts and with a User TELNET IOSIM which is responsible for performing TELNET protocol functions. Like the other IOSIMs discussed, the Server TELNET IOSIM executes in a user ring.

4.1.4 TIP Approach

The TIP's only function is that of a terminal concentrator to other ARPANET Hosts. The TIP software consists of a straightforward stand-alone implementation of the IMP/Host, Host/Host, ICP, and TELNET protocols along with the necessary terminal I/O software. No operating system is used. A TIP design decision closely ties together the notions of connection and terminal port with the result that a terminal can only be associated with one connection at a time. Users can and do make use of capabilities beyond the rudimentary ones provided by the TIP, by connecting through a TIP User TELNET to the Server TELNET of a Host (such as TENEX), and then calling the remote Host's User TELNET and connecting through it to other Server TELNET processes in the network. While one must be careful, e.g., so that no more than one TELNET provides character echoing, the user can in this way borrow features (such as multiple outstanding connections) not supported by his local TELNET.

4.2 Problems and Considerations

In this section we very briefly enumerate a number of problems encountered in the implementation of the TELNET protocol. We present these because we believe them to contain valuable lessons in how to structure a TELNET-like protocol which can be easily integrated into Host operating systems, and some hints on how to structure operating systems which can accommodate TELNET-like functions.

4.2.1 Integrity of Multi-character Commands and Problems of Synchronization

As previously noted, TELNET command sequences are sequences of two or more bytes. Because these sequences must pass over the TELNET connection in messages and buffers of arbitrary sizes, there is no guarantee that a received TELNET command sequence will be completely contained in any one message or buffer. Furthermore, in a given implementation, the sequences of command characters may share a message or buffer with data bytes. Thus, care must be taken to maintain the integrity of multi-byte commands.

Negotiation of an option can require several exchanges of commands between two Hosts. Also, several options may be negotiated simultaneously. Finally, it is usually undesirable to defer data traffic for the duration of these option negotiations. Thus, care must be taken to save the states of multiple, on-going option negotiations. Data structures must be provided to facilitate interpretation and handling of incoming parts of negotiations in order to match them with the previous parts of corresponding negotiations. Finally, care must be taken: to properly synchronize on-going data processing with negotiations, the aims of which might be to affect data processing; to synchronize the effects of separate negotiations which affect common TELNET parameters (or are even in conflict over them); and, to synchronize the two parties sending simultaneous (perhaps conflicting) commands about the same options or even reversing course in mid-negotiation.

4.2.2 Time-outs

The protocol should (but does not) specify reasonable time-outs and actions to be taken to reset the connection to a known state should a time-out occur. For example, with the protocol as currently specified, when a TELNET module initiates an option negotiation, it must wait for a reply. Since the module must store the fact that a request has been made, and since in general it must do this for many requests for many connections, if the other party is tardy in responding, storage may become exhausted. Further, a later and different negotiation might be confused by this left-over request. Clearly, timing routines must be provided that check periodically for such left-over requests. Since the TELNET specification does not adequately address the issue of time-outs, each implementor is left to choose a reasonable course for himself.

4.2.3 Maintaining Accurate Status

All TELNET processes must maintain the current state of the options they implement for each terminal or connection. In addition, to avoid requiring users to set parameters at every terminal session, the nature of options suggests that User TELNETs maintain information for what each terminal type desires or can accept. However, given the variety of terminals, users, and Server Host systems serviced by the User TELNETs, the choice of preferred settings can be a problem. For instance, the TIP policy is to maintain preferred settings judged to be well matched to the needs of the naive user and to allow explicit setting changes if desired. The TIP maintains the desired state of each option even though the terminal may not be in that state. Thus, automatic return to the preferred state is possible at the end of the terminal session. A further choice arises with regard to "automatic negotiation". The TIP will, of course, send an option request at a user's explicit command. However, when a connection opens and options are required to establish the preferred setting, the TIP acts as an advocate for the user and automatically initiates option negotiation.

4.2.4 Logical Processing Control vs. Physical Terminal Control

One of the least tidy areas of TELNET specification and implementation is the three-way conflict among (a) the few keys available on physical terminals to indicate various control functions, (b) the several functions which must be specified, and (c) the existing operating system and terminal manufacturer assumptions about the functional meaning of various key strokes. For example, it is desirable for both the server Host operating system and the user at his keyboard to be able to cause the terminal print head to do a carriage return alone (e.g., to leave the print head in a position to overprint a line), to do a line feed alone, or to do both. Further, it is desirable for both the server Host operating system and the user to be able to signal the passing of logical control to the other (e.g., for the user to indicate that it is now time for the server Host to process a line of data). The fact that many Host operating systems have implicit means to indicate transfer of logical control, such as the arrival of a "new line" character, complicates the situation. Further, some terminals have only a single convenient key with which to indicate the various functions, and on some terminals a stroke of a key (such as carriage-return) physically causes print head motion. To cause minimal change to Host operating systems, minimal user inconvenience, and minimal requirement for physical terminal modification, the TELNET protocol adopted a convention whereby the characters CR, LF, and NUL are used in various combinations to control physical and logical functions. This approach has been acceptable for the most part but there have been certain problems with it, because of ambiguity of the meaning of various character sequences in various situations. Designers of TELNET-like protocols should be careful to provide sufficient unambiguous control sequences, and means of initiating them, to support necessary functions in the context of all the terminals and Host operating systems that use the protocol. Terminal manufacturers and operating system designers could lighten the burden on protocol designers by realizing that a wide variety of terminals and operating systems will be used together and therefore not tightly bind physical actions to logical functions.

5. CONCLUSION

The ARPANET TELNET protocol development has demonstrated the feasibility of constructing a protocol which dynamically adapts to support terminal communication between previously incompatible Hosts and remote terminals. The protocol has also proved useful for process-to-process and terminal-to-terminal communication. The iterative design and implementation experience leading to widespread implementation of the TELNET protocol revealed several fundamental principles of protocol design which we believe have broad application beyond the ARPANET. Further, through this experience, several approaches to operating system design which facilitate TELNET-like communication have become apparent.

One aspect of the TELNET protocol, the Network Virtual Terminal concept, has been widely utilized in other later networks, such as Cyclades [17], Telenet

[18], the European Informatics Network [19], Datapac [20], and EPSS [21]. (In at least one case, the European Informatics Network, the concept of the Network Virtual Terminal has been expanded to specify much more ambitious virtual terminal functions than are specified by the ARPANET NVT.) Furthermore, on behalf of France, Telenet, and themselves, the United Kingdom Post Office has submitted to C.C.I.T.T. a draft provisional recommendation for what is essentially a network virtual terminal. The other aspects of the TELNET protocol, such as sophisticated option negotiation and standard character sets, have been addressed to some extent by most other network designers; however, to date, we believe that the ARPANET TELNET protocol is the most complete, sophisticated, implemented, and widely used such protocol in existence.

REFERENCES AND BIBLIOGRAPHY

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

2. Heart, F.E., 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, 1970, pp. 551-567.

3. BBN Report No. 1822, "Specifications for the Interconnection of a Host and an IMP", revision of January 1976; available from the National Technical Information Service under accession number ADA019160.

4. Crocker, S., J. Heafner, R. Metcalfe, and J. Postel, "Function-oriented Protocols for the ARPA Computer Network", AFIPS Conference Proceedings, Vol. 40, 1972, pp. 271-279.

5. ARPANET Protocol Handbook, April 1976 edition,, pp. 51-174; this document is a compilation by E. Feinler and J. Postel (Network Information Center, Stanford Research Institute) of many documents written originally by many members of the Network Working Group or NWG; also available from the National Technical Information Service, Accession Number ADA027964.

6. Walden, D., "Host-to-Host Protocols", Network Systems and Software, Infotech State of the Art Report 24, Infotech Information Limited, Maidenhead, England, 1975, pp. 287-316.

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

8. Carr, S., V. Cerf, and S. Crocker, "Host-Host Protocol in the ARPA Computer Network", AFIPS Conference Proceedings, Vol. 36, 1970, pp. 589-597.

9. Ornstein, S.M., 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, 1972, pp. 243-254.

10. Dolotta, T.A., "Functional Specifications for Typewriter-Like Time-Sharing Terminals", Computing Surveys, Vol. 2, No. 1, March 1970, pp. 5-31.

11. Thomas R., "Reconnection Protocol", RFC 426 (see the note on RFCs below).

12. Tymes, L.R., "TYMNET - A Terminal Oriented Communication Network", AFIPS Conference Proceedings, Vol. 38, 1971, pp. 211-216.

13. Heckel, P.C. and B.W. Lampson, "The BCC Terminal System", Presented at The Seventh Hawaii International Conference on System Sciences, January 8, 1974.

14. Bobrow, D., J. Burchfiel, D. Murphy, and R. Strollo, "TENEX, A Paged Time-sharing System for the PDP-10", Communications of the ACM, Vol. 15, No. 3, pp. 135-143.

15. "IBM System/360 Time Sharing System: System Logic Summary", Form GY28-2009.

16. Vyssotsky, V.A., F.J. Corbato, and R.M. Graham., "Structure of the MULTICS Supervisor", AFIPS Conference Proceedings, Vol. 27, 1965, pp. 203-212.

17. Zimmermann, H., "Proposal for a Virtual Terminal Protocol", Reseau Cyclades, IRIA, Rocquencourt, France.

18. "Interactive Terminal Interface Specification", Telenet Communications Corporation, 1666 K Street, N.W., Washington, D.C. 20006, September 1975.

19. Schicker, P., and A. Duenki, "Virtual Terminal Definition and Protocol", ACM SIGCOMM Computer Communication Review, Vol. 6, No. 4, October 1976, pp. 1-18.

20. Twyver, D.A., and A.M. Rybczynski, "Datapac Subscriber Interfaces", Proceedings of The Third International Conference on Computer Communication, 1976, pp. 143-149.

21. Howard, V.J. et al., "An Interactive Terminal Protocol", EPSS Liaison Group, Study Group 2, HLP/CP(75)2, AERE Harwell, U.K.

The evolution of the TELNET protocol is thoroughly documented in a series of working papers known as Requests for Comment or RFCs. These RFCs are not publicly available although throughout the ARPANET community there are many complete sets, to one of which, no doubt, the serious researcher can gain access. The numbers of the relevant RFCs are listed immediately below.

RFCs on New TELNET Design and Specification: 357, 426, 435, 461, 495, 513, 529, 559-560, 562-563, 581, 587, 595-596, 651-659, 671, 698, 718-719.

RFCs on New TELNET Implementation: 559, 593, 669, 678, 688, 701-703, 718.

RFCs on Old TELNET Design and Specification: 15, 97, 109-110, 137, 139, 158, 295, 318, 328, 340, 391.

RFCs on Old TELNET Implementation: 206, 216, 452, 466.

RFCs on Satellite Considerations: 346, 355.