TELNET: THE MOTHER OF ALL (APPLICATION) PROTOCOLS

Rohit Khare • *University of California, Irvine* • *www.ics.uci.edu/~rohit/*

> The equipment was state of the art, but having a room cluttered with assorted computer terminals was like having a den cluttered with several television sets, each dedicated to a different channel. "It became obvious," [ARPA IPTO Director Robert] Taylor said many years later, "that we ought to find a way to connect all these different machines."
>
> *Where the Wizards Stay Up Late*
> —Katie Hafner and Matthew Lyon

In the beginning, there was Arpanet—commissioned not, as popular myth has it, to provide survivable military command and control, but rather to reduce ARPA's grant overhead of buying every fledgling computer science department its own computer.

Time-sharing had already demonstrated that a single computer could service an entire research group; networking would allow the nation's researchers to work together off of specialized machines. Ma Bell, of course, could already install dedicated data circuits to link terminals directly to remote hosts, like the three teletypes to MIT, Berkeley, and RAND in Robert Taylor's office. The promise of packet-switching technology, by contrast, was to allow any terminal to access any host.

In fact, Telnet was the first application demonstrated on the four-IMP (Interface Message Processor) network installed by the end of 1969. It took one more year to make a real host-to-host protocol and another year for the "glitch-cleaning committee" to complete its work. Request for Comments #97, "First Cut at a Proposed Telnet Protocol" by J.T. Melvin and R.W. Watson, was finally published on February 15, 1971.

The final edition took 12 more years to develop, culminating in Internet Standard #8 in 1983, three years after the final TCP specification was ratified. Arguably, Telnet even predates internetworking and the modern IP packet and TCP transport layers.

Table 1 describes the six drafts in the canonical Telnet standards suite. As simple as Telnet seems in retrospect (it's barely a tenth the size of the HTTP 1.1 specification), it represents decades of experience in application-layer protocol design.

Telnet's true value is not the abstraction of how-to-wire-terminals-to-hosts (standardizing connections); instead, it's the abstraction of terminals (standardizing endpoints). Telnet's application-level semantics are captured in its external interface, the Network Virtual Terminal. Its internal interfaces have close ties to TCP transport facilities, an option negotiation scheme, and symmetric treatment of client and server roles.

Notice that none of these three internal functions has any bearing on remote-login, the most popular *application* of Telnet. Instead, in considering Telnet against the taxonomic criteria set forth in the premiere installment of Seventh Heaven (*IEEE Internet Computing*, Vol. 2, No. 2, Mar.-Apr. 1998, pp. 80-82), we need to separate Telnet from the services it can access. That helps highlight the enduring lessons of Telnet for future seventh-layer protocol designers.

What: The NVT

The original spectrum of host computers connected to the Arpanet formed a motley crew: varying keyboards, character sets, display sizes, line lengths, and speeds—and those were just the physical incompatibilities. The terminal sessions themselves were governed by time-sharing systems, each with its own peculiar ways of stopping and starting processes, controlling the flow of output, and so on.

Rather than writing pairwise adaptors in the form of idiosyncratic terminal drivers for each host system, the Network Virtual Terminal provided a common baseline: 7-bit US ASCII, three mandatory control characters (Table 2), five optional control characters (Table 3), and a basic signal set (Table 4).

The fundamental characteristic that an NVT can't abstract away from a "real" terminal is *latency*. Characters take time to be delivered, unsynchronized and with variable delay. Rather than two points of control—the dataflow in and out of a terminal—there are four: two at the local and two at the remote ends of each connection in each direction.

For example, typing a character doesn't result automatically in an echoed response to display back. RFC 857 outlines options for echoing characters at any of those four control points (as well as a fifth, none at all). Similarly, software flow control can be gated in three places: locally, in the Telnet client application, or at the server side (RFC 1080).

New features can be activated between consenting Telnet processes to upgrade the NVT to more closely approximate the actual terminal. For example, internationalization demands broader character set support; RFC 2066 presents how Telnet can initiate negotiation over character sets, then subnegotiate over actual sets and translation tables.

The effort of converting in and out of such a character set may also need to be shifted. Using Telnet to emulate a 3270 or 5250 terminal connected to an IBM mainframe centralizes the burden of mapping EBCDIC to ASCII. A TN3270 client offers to cook the data at the client (RFC 1647 and others).

Still, the NVT can accommodate fairly obscure terminal features. Interested readers are directed to RFC 1097, "Telnet subliminal-message option."

Finally, there are out-of-band signals for controlling the NVT session itself, usually issued by typing a local escape sequence (typically Ctrl-]). Table 4 enumerates commands to interrupt processes and erase information from terminal buffers.

How: The Implementation

> The Telnet application is one of the simplest and more straightforward of all TCP/IP applications for end users.
>
> *TCP/IP Clearly Explained*
> —Pete Loshin

Simple as Telnet may seem to use, its internal implementation depends on understanding TCP, a careful ballet of option-negotiation steps, and symmetry as a design principle. A "textbook" implementation of the Telnet finite state machine weighs in at 70 pages (*Internetworking with TCP/IP*, Vol. 3, by Douglas Comer and David Stevens, Chpts. 25 and 26).

Table 1. The canonical Telnet standard suite, published on 1 May 1983 by Jon Postel and Joyce K. Reynolds.

RFC	STD	Title	Comments
854	8	Telnet Protocol Specification	NVT, commands, and negotiation
855	26	Telnet Option Specifications	How to register and document options
856	27	Telnet Binary Transmission	Allow 8-bit clean connections
857	28	Telnet Echo Option	Activate remote/local echo on each way
858	29	Telnet Suppress Go Ahead Option	Full-duplex NVT (rather than half)
859	30	Telnet Status Option	Recap current options state
860	31	Telnet Timing Mark Option	End-to-end synchronization point
861	32	Telnet Extended Options-List Option	Reserve option 255 to allow future options

Table 2. Characters that normatively control output of an NVT. (All other control characters and high-bit set characters are undefined.)

No.	Code	Name	Meaning
0	NUL	Null	No operation
10	LF	Line Feed	Move down one line, same horizontal position
13	CR	Carriage Return	Set horizontal position to the left margin

Table 3. Characters that may optionally control output of an NVT.

No.	Code	Name	Meaning
7	BEL	Bell	Audible or visible signal; no cursor movement
8	BS	Backspace	Move the cursor to the previous print position
9	HT	Horizontal Tab	Move cursor right toward the next tab stop (setting unspecified)
11	VT	Vertical Tab	Move cursor down toward the next tab stop (setting unspecified)
12	FF	Form Feed	Move cursor down to the next page, preserving horizontal position

Table 4. The basic commands abstracting control of an NVT.

#	Code	Name	Meaning
255	IAC	Interpret As Command	The next byte is a command, or an escaped 0xFF
244	IP	Interrupt Process	Suspend, interrupt, or abort the remote process
245	AO	Abort Output	Suspend, interrupt, or abort the remote process's output
246	AYT	Are You There	Check that the remote Telnet process is alive
247	EC	Erase Character	Delete the previous "print position"
248	EL	Erase Line	Delete the previous "line"
249	GA	Go Ahead	Turn over control, for half-duplex terminal equipment

Table 5. Negotiation messages in Telnet.

Request	Response	Interpretation
DO	WILL	Initiator begins using option
	WON'T	Responder must not use option
WILL	DO	Responder begins using option after sending DO
	DON'T	Initiator must not use option
DON'T	(WON'T)	Warning or notification Initiator deactivated option; respond if it's a change
WON'T	(DON'T)	Warning or notification Responder should deactivate; respond if it's a change

There's only one category of Telnet client programs: login tools. But Telnet itself can be used by many applications.

Transport Dependence. Many later application-layer protocols, such as the X Window System, state a requirement for "full-duplex, byte-oriented transport." Their semantics are specific enough that correct implementations always read and write information appropriately within finite buffer space.

Telnet, though, connects arbitrary processes, raising the specter of deadlock if information backs up. Hence, it carefully requires TCP, a richer transport facility. In particular, Telnet has URGent delivery of out-of-order segments. It resynchronizes a connection by sending an urgent packet ahead to warn the other end to dump its buffers and throw away all the now-extraneous data until the matching Data Mark arrives.

There are other ways for transport facilities to address terminal latency. While an NVT defaults to line-by-line buffering, it can be set to character mode. On a LAN, it makes sense to allow interactive use, even if a lone keystroke triggers an entire 41-byte TCP segment.

Across slow wide-area links, though, the TCP layer should coalesce segments. This Telnet behavior inspired the Nagle algorithm of limiting low-bandwidth connections to a single outstanding segment (and collecting outbound characters while the ACK is pending).

Negotiation. Latency also affects how negotiation proceeds. Even without worrying about symmetry—designers usually rely on a master-slave relationship to decide who moves first and how to break ties—the fact that data can be in flight in both directions ambiguates requests and responses. It's sufficiently complex and prone to infinite looping that Experimental RFC 1143, "The Q Method of Implementing Telnet Option Negotiation," was published in 1990.

Telnet options affect each direction of dataflow separately. Either party can announce that it wants to begin using (DO), that it wants the other side to begin using (WILL), it refuses to use (DON'T), or refuses to let the other side use (WON'T) an option.

After first ascertaining that both parties understand the option and want to apply it, there's typically a subnegotiation over the actual parameters (arbitrary data bracketed by IAC SB ... IAC SE). Since both sides could initiate discussion simultaneously, the Q Method maintains six bits of information for each option in each direction: on, off, wanted-on, wanted-off, and which party initiated negotiation.

There are 255 possible option codes. Each new option should be documented in the RFC series according to the procedure outlined in RFC 855. However, if there's ever a 256th option, the designers planned ahead by reserving the last code for subnegotiating in an expanded code space (RFC 861). There's also an option for reporting which options are activated, allowed, or prohibited on the current connection (RFC 859).

Symmetry. Once established, there's nothing inherently client-server about a Telnet connection. True, the TCP port-numbering scheme can disambiguate who initiated the connection (that's the end connected to the well-known port—for example, 23 for Telnet; the reverse channel is randomly assigned). The designers carefully eliminated master and slave roles from Telnet's own semantics.

In fact, they wrote "The Telnet protocol is based on three main ideas, [one of which is] a symmetric view of terminals and processes." In other words, Telnet can connect users as well as programs, seemingly stretching Unix pipelines across the Internet.

Why: Telnet as a TP

It may seem odd to have dissected Telnet at such length without once mentioning logins, passwords, and all the other details of establishing a terminal connection. In fact, Telnet is entirely silent on this point—not that later writers haven't tried to shove authentication into this layer anyway (see RFC 1416). Telnet is entirely separate from the applications it accesses (unlike, say, the Unix rlogin program or the File Transfer Protocol, which do define accounts, passwords, and trusted hosts).

That separation is well hidden because there's only one category of Telnet client programs: login tools. By contrast, HTTP is routinely used and abused in myriad software packages to send Web pages, audio, print jobs, news tickers, even distributed RPC messages. Telnet itself can be used by many application programs beyond a login: library catalogs, router configuration panels, multiuser games, and more.

Before the Web, Telnet was the most popular way to proselytize a new application, from the Line Mode Browser installation at info.cern.ch, to WAIS, to Archie, even back to the Stanford AI Lab's Adventure server. Furthermore, you can point a Telnet client at another TP's port and debug it directly.

One of the classical virtues of IETF protocols is plain-text design: protocol messages are spelled out in natural language rather than packed binary structures. This makes it very easy to experiment with Internet information services (for example, typing in an HTTP GET message on port 80 or an SMTP MAIL command on port 25).

As open-ended as it is, it helps to think of Telnet as just another message transfer protocol. Rather than the crisply identifiable lump-of-data in a file transfer or e-mail message, though, a Telnet session is now smeared-out in space-time, a pair of synchronized input and output logs with interspersed commands. As unfamiliar as that seems, it's only a bidirectional equivalent of a RealAudio broadcast stream or QuickTime video, other kinds of "messages" with temporal structure.

The message is exchanged between NVTs, which are addressed as hosts (by domain name or IP number) since Telnet is an end-to-end protocol. The actual rules governing the contents are known only by the port number, so the port determines the name of the session ("it's connected to port 119, so it's an NNTP message stream"). Data transfer is bidirectional and synchronous, and uses the URGent interrupt facility of TCP.

Lessons Telnet Teaches

In the fast-paced world of Internet engineering, it's hard to accept that Telnet is older than I am. Its remarkable stability deserves close study.

Simplicity alone is held up as a classic cause: the NVT is an artful balance of features that can span the range from half-duplex teletypes to cell phone PCs. On the other hand, the steady march of whiz-bang progress would have upset that balance and triggered periodic reengineering of the protocol; e-mail reveals this tendency.

Instead, I'd hold up evolvability as its key survival skill. Option negotiation was a farsighted addition in its time; it did not become a hallmark of IETF protocol design until the late 1980s. True, many other systems had hooks for new headers, new verbs, and so on, but few planned ahead with administrative procedures for managing the option space. There are now 93 RFCs that describe Telnet and its options—almost 3 percent of the entire IETF fossil record!

File transfer is an equally ancient Internet application, and its protocol has been almost as stable, but doesn't exhibit such evolvability. More on that paradox in the next Seventh Heaven.

Rohit Khare is a graduate student in computer science at the University of California, Irvine. His personal Web site is at http://xent.ics.uci.edu.

URLs for this column

Anne and Lynn Wheeler have put together an excellent resource for surfing the RFC archives at http://www.garlic.com/~lynn/rfcietff.htm.