Xmodem, Kermit, and Similar Protocols for Personal Computers

In this report:

Synopsis

Editor's Note
This report provides an overview of most of the PC file transfer protocols in use. It focuses on the two most commonly found in PC networks today—xmodem and Kermit. Each has an application environment, which this report will help identify. The report also examines newer and more efficient PC protocols that will be available in the future.

Report Highlights
The demand for the exchange of information among personal computers and between PC and minis or mainframes continues to increase. Unlike the minicomputer or mainframe environment, PCs have evolved to enjoy considerable freedom of transmission and file exchange between machines from different vendors. Today, taking advantage of this freedom presents a problem for established data centers, which must adapt and manage the information flow to and from PCs. A major problem is the informality with which most PC file transfer protocols have developed and continue to evolve. To date, no standards body has initiated a "standard protocol" for PC file transfers. No regard has been given, in what has been done, to integrating PCs into the OSI architectural model. What integration can be found has occurred by chance.

This report was prepared exclusively for Datapro by Donald E. Kimberlin, principal consultant of Telecommunications Network Architects, Safety Harbor, FL. Mr. Kimberlin has more than 30 years' experience in planning, designing, and implementing communications networks on five continents. He is a consultant member of STC and BICSI, a Certified Broadcast Technologist of SBE, and a member of the Advisory Board for *Datapro's Management of Data Communications*.

Problems with Protocols

The term "protocol" causes considerable misunderstanding among data communications users. All too often, a narrow meaning is applied to a broad term. A dictionary definition calls a protocol a set of negotiated agreements and arrangements that allow parties to accomplish a function. The meaning is so broad that the term has been used to describe diplomatic relations for centuries.

Computers and data processing are relative newcomers to the concept of a protocol. A standard may apply to a device operating in isolation; however, a protocol always intimates the exchange of signals or messages. Obviously, a protocol does not become a standard unless a range of unrelated parties agrees on its use or accepts it as a given.

For example, the widely used IBM Binary Synchronous Communications (BSC or Bisync) protocol is often misclassified as a standard. Many vendors claim adherence to the Bisync standard, but users have found hundreds of detail variations of the use of Bisync in IBM networks. Most of them can cause a communications failure when messages are exchanged between two IBM computers using different versions of Bisync. In addition, non-IBM suppliers have their own variations in the operating rules their version of Bisync uses—the details of these variations are not public information. Their descriptions only exist in the proprietary documents of the respective vendors.

Protocols reside at many layers of a communications network and must function correctly at each layer for the network to operate. A good example lies in the tendency of engineers to speak of the RS-232 protocol. In fact, beyond the definition of interface hardware names and functions, there is a sequence of steps required to start a data transmission link between two terminal devices. These steps represent a set of rules that must be followed. (Surprisingly, however, neither EIA RS-232 nor its international equal, CCITT V.24 combined with V.28 and ISO 2110, state those sequenced steps. The steps are merely implicit in achieving the end goal.)

As ISDN evolves, CCITT Signaling System Number 7 (SS7) is bandied about, often with an air of considerable simplicity. In fact, SS7 fills a whole volume of the CCITT Recommendations describing a packet network much like X.25 for use between automated switching machines. The standards for SS7 describe not only that information transfer protocol but also the function and responsibility of supporting processors such as the Signaling Transfer Point (STP). The entire volume could be called the "protocol" of SS7.

This report focuses on message transfer protocols for personal computers. These protocols are needed in higher layers of networks to manage the flow of information after a data link has been established. Message transfer protocols provide functions such as character error checking and recovery, traffic flow control when buffers become overfilled, and similar functions more complex than those that can be done by lower level hardware operations.

For example, the core function of a modem is to encode and decode transmission channel signals into digital pulses on an interface wire to a terminal. While a modem may declare an error if it loses the carrier signal from the far end, a noise pulse on the line may destroy a few bits and send erroneous data to the terminal. The modem itself does not read the terminal traffic, so a higher level protocol is needed to check for corrupted bits or characters that may have been transmitted without a modem alarm. In terms of the modem layer of a network, the modem will output whatever is input (within limits). Passage along the link, however, can also corrupt even a properly structured data message. Message and file transfer protocols with some processing capability are needed to detect and, in most cases, correct these errors.

In this sense, protocol operations are properly a function of communications processing as opposed to data processing. In the early era of DP machinery, a system had only one processor, so it had to provide the processing of communications protocols as well as its main data processing function. Today, microprocessors have moved that function into a front-end processor and unburdened the DP machine from such interrupts.

IBM's Basic Telecommunications Access Method (BTAM) was replaced early with better methods. Today's personal computers, however, still generally have only one processor and must function primarily as the early BTAM did. This is not typically a problem, since the personal computer is still used by one individual performing one task at a time. The processing power of PCs is increasing so rapidly that even 386-based machines can be shared by groups of a dozen or more users on dumb terminals, leading to another round of the operational and architectural evolution we have seen in mainframes.

Keyboard Communications

Requirements for error checking and correction in keyboard transmission systems are often sufficiently simple that there is no need for a protocol that processes message blocks. Here, a large responsibility is typically placed on the terminal operator because errors in text transmission are easily spotted and readily corrected in the input step. Just as a word processing computer echoes the characters typed into it back to its screen, distant computers echoplex back the characters they receive from the data link.

The error-checking and -correcting protocol then falls to the keyboard operators themselves. Of course, every error to be corrected becomes a stopping point, with a return to the errored data for reentry. This method is simply automated by many mechanized applications . . . the machines detect what seems to be an error and they back up to the declared error, retransmitting and rechecking that portion until it seems correct.

Bulk Transmissions

Even if the transmitted data is keyboard information, using batch transmission methods of saving the data and sending it in bulk brings forth the need for error-checking and -correcting protocols. In this situation, the operator (or more likely, multiple operators) is no longer in a loop that can correct errors as they enter the transmission. In fact, in many of today's dial-up data collection applications, the operators are not even present. Trying to hold individual transactions and correct them the following day is inefficient. The need to add message protocols to bulk or batch transmission is obvious.

Machine-Language File Transfers

Similarly, it is obvious that file transfers between machines are so difficult and expensive to repair by human action that protocols to catch and correct errors become essential. Personal computers have progressed very rapidly to techniques of compressing even text files to shorten transmission time by half or more. This is a sound move to avoid transmission errors. Data errors occur in random bursts on physical channels; minimizing the time to transmit a file reduces the need for significant repairs.

The process of compressing data, however, involves removing spaces and replacing repeated characters or phrases with short code symbols. Even a text file, when compressed, becomes humanly unreadable. Providing a mechanized way to trap and rectify errors in transport is far preferable.

The transmission of data such as the numerical entries parsed out of a spreadsheet program is very common with personal computers. Entire executable programs are sometimes transmitted between PCs. Large machines' data storage may be used to simply hold entire executable programs for the organization's PCs. The machine may not be capable of reading the stored programs; it just stores and ships them as binary files. A related application uses electronic communications to distribute revised and updated PC programs from a main center's storage.

Regardless, as soon as the human keyboard operator is removed from the immediate loop at

Where to Get More Information and Copies of Protocols

Xmodem, ymodem, and zmodem
Omen Technologies
17505-V Sauvie Island Road
Portland, OR 97231
(503) 621-3406
Modem (503) 621-3746
CompuServe: 70715,131

CBBS Computer Bulletin Board
Chicago, IL
(312) 849-6279
Modem (312) 545-8086

Many and varied PC bulletin boards around the nation and the world.

Several Special Interest Groups (SIGS) on CompuServe, BIX, GEnie and similar commercial services. These are particularly useful places for information on other than the most common protocols.

Kermit
Kermit Distribution
Columbia University Center for Computing Activities
7th Floor, Watson Laboratory, 612 W. 115th Street
New York, NY 10025

the instant of input, some form of automated error checking and correction is needed.

Proliferation of PC Protocols

Lacking the support of even one dominant vendor, PC protocols have evolved through several stages. Considering that the first PC-automated protocol is just more than a decade old, the evolution has been very rapid. Considering further that some PC protocols perform very advanced functions (even to the extent of recovering interrupted transmissions at the point of failure and resuming from that point), PC protocols may be more advanced than many operational large machine protocols.

This innovation and evolution of PC protocols also brings with it the problem of limited backward compatibility. Establishing a link between two PCs in which the generation of a broadly named protocol is dissimilar typically results in a nonfunctional link. This case is not uncommon when using the prototypical PC protocol called xmodem. Our report will detail the evolutionary variants of xmodem later.

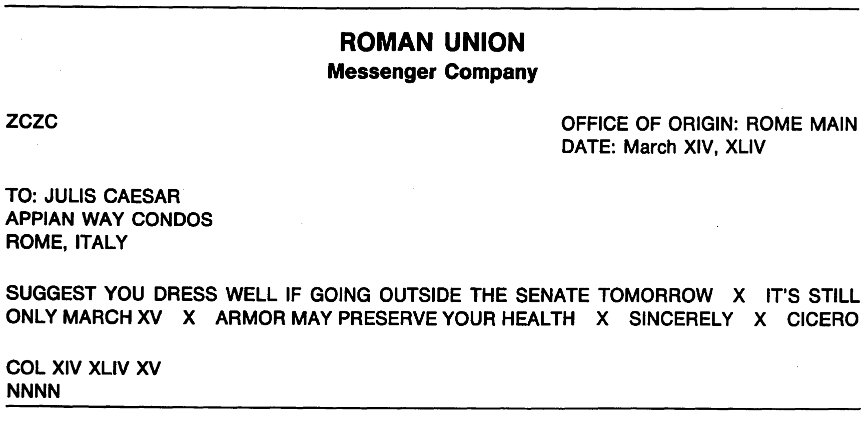
Common Elements of All Protocols

There is considerable commonality in the elements needed by any protocol. Telegraphers and postal clerks found that these same elements were needed in handling messages for the public more than 150 years ago. (See Figure 1.) First, there is need for a header that gives the message's destination, how to route it if there is a choice, and what priority or "class of service" the message is to get. Second, the message itself follows. Third, a "trailer" is sent, containing a form of error checking, so the receiving operator can confirm that the message was correctly received. Just as in telegraphy, the error correction method orders a retransmission in case of a questioned message. As seen in Figure 1, every protocol must contain three major parts: the header, the message, and the trailer.

Header: The header identifies how the following parts are to be handled and routed. In the case of a paper message, the actual message form itself conveys some of this information. For example, even those of us who have never personally received a telegram expect it to be on canary paper, with the name of the telegraph company highlighted in bold. Information that gives the source of the message (in case it must be returned or replied to), its age (which may be important to using the message or sequencing it among others), and the intended delivery point is included in the header.

Not all communications systems require all the subelements in a header. An example found in computer communications is protocol blocks for point-to-point (including dial-up) lines between a single transmitter and receiver. Here, the source and destination have been clearly identified before the protocol layer is started. In such a situation, it is clearly wasted time to send the address and source in every block, so they are often eliminated. If a point-to-point link is only part of a networking system, however, blocks may be coming from sources beyond the sender and traveling to points beyond the receiver. Here, the address (as well as source information) must be sent, so a return is possible if needed.

This structure is found in packet networks. Packets must add even more elements to provide for control functions such as sequencing the packets at the receiver, identifying where a packet has been so it is not sent repeatedly around a loop, and similar management functions.



ROMAN UNION
Messenger Company

ZCZC

OFFICE OF ORIGIN: ROME MAIN
DATE: March XIV, XLIV

TO: JULIS CAESAR
APPIAN WAY CONDOS
ROME, ITALY

SUGGEST YOU DRESS WELL IF GOING OUTSIDE THE SENATE TOMORROW X IT'S STILL ONLY MARCH XV X ARMOR MAY PRESERVE YOUR HEALTH X SINCERELY X CICERO

COL XIV XLIV XV
NNNN

Figure 1.
Common Elements

Even a messenger-borne telegram must have the three major elements of a computer message protocol. First, the header shows the source of the message and where it is to be delivered. The second major element is the message body itself. The last element is a trailer, here including the classic telegrapher's error check.

Generally, header items are most useful to have at the receiving end before the message arrives. Processes to get the message where it belongs can be started immediately. The logic can be shown easily using the postal letter as an example. How would the Postal Service function if the sorting clerks had to read every letter to determine where it should be forwarded?

Message: The second segment of a message block is the message itself. The message is where a protocol is determined to be character or bit oriented. If the protocol is character oriented, only whole characters containing the number of bits used in the code of that network are transmitted. In telegraph networks using the Baudot teleprinter (CCITT #2) code, a character slot could have 7.5 bit times, the length of one teleprinter character. The protocol would get maximum throughput when using this code. To send 10- or 11-bit asynchronous computer code characters (CCITT #5), some manipulation would be necessary and would likely waste line time. Unless considerable buffering and processing were available at each end, it would be best to use two 7.5-bit character slots for each single, longer character to make it fit.

Thus 15-bit times would be used for each 10- or 11-bit character, sending meaningless bits for 4 or 5 bit periods of each 15-bit "super character." The cost in throughput is obvious, either 10/15 or 11/15, about 67 or 73 percent as efficient as handling the code for which the protocol was intended. Conversely, a character protocol intended for 10-bit characters, if handling 7.5-bit characters, would have 2.5 bit times wasted in each character slot. The result would be 7.5/10 or only about 75 percent of its peak efficiency.

The second protocol is a bit-oriented protocol. Here, the message portion of the block carries a bit string of packed data so no space is wasted. The connected terminals must now have an even higher layer of operation, however, capable of identifying each character's beginning and end. If different codes are to be handled, each end's higher layers must recognize and adapt to the code changes. Such variability might be reflected in the protocol's header section, where a control bit or character identifies the contents of the message block.

For a human example, listeners to the Voice of America shortwave radio broadcasts hear a "protocol" announcement preceding each program. In one short sentence, the announcer says, "This is the Voice of America transmitting from Washington, DC, United States of America. The following program is broadcast in Spanish" (or Mandarin, or Cantonese, to give examples of the dialect variations in the code language we call Chinese).

It becomes obvious, then, how header information most sensibly belongs before the message in the transmission block. We can also see another example of why certain items should be transmitted before the message. Many protocols have now advanced to the point of transmitting a first packet (often numbered 0) to accomplish this. Such advance packets are generically called herald packets.

Many protocols have an agreed number of characters or bits contained in the message portion of the blocks. Telegrams, in fact, are set by international standard (CCITT Recommendation F.30) to be 50 words maximum. If a telegram exceeds 50 words, it will be broken into 50-word segments and sent as a series of separate messages. A 51-word telegram places the fifty-first word sent on a second sheet, which is a complete telegram with its own address, routing, and error-checking information.

Figure 2.
Examples of CCITT Character Control Strings*

Baudot String	**Meaning**
ZCZC	Start of Message
NNNN	End of Message
DFDF	Connect this message through (to . . .)
FFFF	Connect Auxiliary Device (tape punch)
DDDD	Disconnect Auxiliary Device

From CCITT "F" Series Recommendations for Public Telegraph Networks.

The CCITT-agreed strings of characters above can be seen in international telegrams today. They are strings of four characters, selected to have no meaning in any human language known. In earlier times trapped and acted on by relay logic, they have for years been used by computerized telegraph networks. Even today, vestiges of them can be found as protocol elements in computer networks.

Figure 3.
Examples of CCITT Character Control Strings and TWX/ASCII Character Equivalents

Baudot String	Meaning	TWX/ASCII Character
ZCZC	Start of Message	SOM
NNNN	End of Message	EOT
DFDF	Connect this message through (to . . .)	DLE
FFFF	Connect Auxiliary Device (tape punch)	DC1
DDDD	Disconnect Auxiliary Device	DC4

Here we see the TWX/ASCII Codes implemented, typically nonprinting control characters in place of the strings of four characters the Baudot code had to use for controls.

In fact, the format of an international telegram is specified in the F Series of the CCITT Recommendations. The 50-word limit is also related to the counting of characters for messages. In the X Series, there has been recurrent talk about a type of service called a Datagram, which, by definition, would be one packet long—the packet network equivalent of an international telegram. While packet networks could handle such traffic, most objections come from the threatened operators of international telegram services, who could see packet networks replacing them.

The close relationship between telegraphy of an earlier era and computer networks of today is obvious.

Megalithic Protocols

Protocols for message control came early to electrical communications. From the outset, telegraphers found they needed the means to correct errors. Initially, it was a manual operation; error checking amounted to the receiving telegrapher judging the reasonableness of what was copied to paper. When it seemed unreasonable, the receiver merely interrupted the transmitter by opening the line, which silenced all transmissions, just as though the line had failed. This function is now called a reverse interrupt; in fact, several communications codes have a character named RVI for just that purpose. Saying in one character notifies the sender that the receiver seems to have detected an error. In PC protocols, the RVI function is typically performed by the control character NAK, for Negative Acknowledge.

Following the RVI, a short message tells the sender to back up and retransmit the intervening material again. In telegraphy, where line time was very precious, the signal consisted of the letters AA followed by the last word(s) received correctly. In modern computer systems, the receiver typically tells the sender to back up so many entire transmission blocks, giving rise to the name, "Go Back n," where n equals the number of blocks to back up and retransmit.

Between the time of the manual telegraph and today's computer networks, however, there were decades in which teleprinters were used. In teleprinter networks of the Baudot era, the code set was so severely limited that there was no room for single-character control signals. The telegraphers had to therefore agree on character strings for such error control. What resulted and can be found in the CCITT Recommendations for telegram networks are standardized sets of four-character strings that have agreed-upon meanings.The next logical step in evolving networks was to replace the character strings with single control characters if the code used had sufficient combinations available. This development occurred in the 1930s when Bell Laboratories was working on its system called Teletypewriter Exchange Service (TWX). Developed in the U.S. in parallel with telex, which was under development in Germany, the TWX developers chose a new code, while the Telex developers used the existing Baudot code and its control string functions.

Using Baudot and its limited code set would have required TWX users to become familiar with telegraphy. The TWX developers instead wanted a system and network that came as close to typing a letter on a typewriter as possible. (In fact, TWX could be considered the first electronic mail operation.) To emulate the typewriter, it was necessary to expand the number of information bits from the five bits per character of Baudot to seven bits per character. Doing so provided for 2^7 combinations—128 characters—in the TWX code. Of these, only 62 were needed for text characters and numerals in English. Even with a fair quantity of text punctuation marks added, there were still combinations available for control characters. The TWX developers added single characters that

could perform the functions of the former Baudot control strings.All the preceding message control work had been done in the era of relay logic and digital transmission. When computers were applied to such work, it was only sensible that they would function in the same way. (There were, in fact, and still are places in the world where computerized terminals operate with relay logic at the far end.) It made sense to have equivalents in computer codes for the controls in telegraph codes. As a result, direct equivalents to the controls of both Baudot and ASCII can be found reaching back to IBM's seven-unit BCDIC and continued in its eight-unit EBCDIC codes.

Figure 4.
Examples of CCITT Character Control Strings and TWX/ASCII Character Equivalents Plus BCDIC/EBCDIC Character Equivalents

Baudot String	TWX/ASCII Character	BCDIC/EBCDIC Character
ZCZC	SOM	PREFIX
NNNN	EOT	EOT
DFDF	DLE	BYPASS
FFFF	DC1	PN
DDDD	DC4	RESTORE

Here we can see that the IBM computer codes of BCDIC and EBCDIC have direct equivalents equating to telegraphy.

Recognition Conflicts

Although recent advances have made processing messages far more flexible, earlier workers in the development of message transmission had limited message capabilities. The limitations of their hardware-based logic made it impractical to have a character or string bear more than one meaning. If a particular control character had some meaning to the transport link, that character could not be used in higher layers of the network.

In the early days of TWX, the processes were so simple that an adequate number of characters were available. The characters assigned for a higher level function such as "horizontal tab stop" on the printed message, Control I has the same meaning now as it did 50 years ago. A character that once meant "Stop sending, I have run out of tape," Control Y is widely used in Wordstar and other word processing programs to mean "delete a line."

What happens if we send a Control-Y down a communications line to signal stop sending, and the receiving end thinks it is a word processing command? Or, what if the word processing program should send it through some line equipment that understands it as a command to shut down? (The command to shut down is the *standardized* meaning of the character in both ANSI and CCITT documents.)

That scenario is one example of the many conflicts that occur and cause problems with the design and operation of communications networks. If anything, Control-Y should have been used for the more modern purpose of flow control when receiving buffers are filled, but by the time the technology that needed buffer flow control was available, that character had been usurped for word processing. The result was that makers of stat muxes selected the ASCII characters "DC1" as a substitute, only then to find that yet another vendor's word processing system *did* use DC1 for tabulation control. Muxes now need options to select which control characters they can insert into the link traffic depending on what sort of terminals and software the customer is using. Setting these options incorrectly causes either link shutdowns or strange reactions on terminals. Explaining to users what has to be done goes beyond what most users want to know; it is usually just fixed and never explained.

The "trickle-down problems" that can occur in communications links, however, now show clearly. Misuse of one character in the code set causes problems later on. Character recognition, therefore, becomes increasingly important in alleviating such problems.

Assuming error-free transmission, we could set up some simple rules we might call a "transmission block protocol," understanding that if Control-Y appears in certain positions within a block it is a line control character; in other positions it is a word processing control character. Such rules are the essence of message block protocols. From them, vendors make equipment that can use or ignore the characters as appropriate; however, the possibility of conflicts in use of control characters always looms over network designers.

Table 1. Modem (Checksum) Operation Sequence

SENDER		RECEIVER
(Following an exchange in which sender and receiver agree on the file to be transferred....)		
	←	[NAK]
[SOH][001][254][..128 bytes][chksum]	→	
	←	[ACK]
[SOH][002][253][..128 bytes][chksum]	→	
	←	[ACK]
(... the third block of data is errored and the receiver rejects it..)		
[SOH][003][252][..128 bytes][chksum]	→	(chksum
incorrect)		
	←	[NAK]
(... the sender again sends the third block and it does check correctly...)		
[SOH][003][252][..128 bytes][chksum]	→	
	←	[ACK]
[SOH][004][251][..128 bytes][chksum]	→	
	←	[ACK]
(...this time, the ACK from receiver to sender is errored, so the sender times out and resends...)		
(unexpected response from receiver)		
[SOH][004][251][..128 bytes][chksum]	→	
(...the receiver again ACKs, and is received ok...)		
	←	[ACK]
[SOH][005][250][..128 bytes][chksum]	→	
	←	[ACK]
(...the file has been completed, so the sender notifies the receiver with a single character...)		
[EOT]	→	
(...the receiver ACKs now to tell the sender to return to normal operation...)		
	←	[ACK]
(...and the XMODEM session is completed.)		

Mainframe and Minicomputer Protocols

So far, our discussion has been about message protocols in general. At the outset it was far easier to recognize, trap, and handle characters than to reconstruct the meaning of bit patterns for line controls. As a result, even computer lines used protocols planned around handling characters. In a character-oriented protocol, the number of information bits in a character must be known and agreed to by both sender and receiver. The seven ASCII information bits provide 127 usable character combinations. ASCII code, as properly used, contains two characters (Shift Out [SO] and Shift In [SI]) for users to switch to an extended set of character meanings. The U.S. military, for example, uses SO and SI properly to switch to weather symbols in its weather networks.

Makers of mainframes and minicomputers similarly interpreted ASCII and used SO and SI for extended characters. Thus, mainframes and minis use seven-bit ASCII in its standard sense. In this case, the eighth bit of each character is used for a per-character Vertical Redundancy Check (VRC) called parity for error checking. Large machine protocols are therefore based on seven information bits in async transmission.

The personal computer had different needs. Its early computing power was so small that shifting to a different character set was quite burdensome. The early PC also had no communications capability. Its designers disregarded each eighth bit for error checking and instead used all eight bits of each character for information bits, permitting 256 combinations of characters called extended characters or extended alphabet. Obviously, they could not be transmitted meaningfully (or even successfully) down mainframe links that permitted only seven information bits per ASCII character.

This leads to a schism in personal computer networks and a main division in this report—eight-bit versus seven-bit protocols. It also justifies the two major PC communications protocol types: xmodem and its descendants (eight bits) and Kermit (designed specifically to connect PCs into the seven-bit environment).

Eight-Bit Protocols: Xmodem and its Descendants

There are at least six identifiable forms of xmodem, the first PC message protocol. Considering the origin of xmodem and its development, today's widespread use of xmodem is nothing short of astounding. Also noteworthy is the fact that xmodem is not documented by any standards body. Users must know with which variant they are working.

The history of xmodem is very short and quite well documented. Until February 12, 1978, it can be said that microcomputers were not used for communications in any more than the most rudimentary sense of keyboard text transmission. Essentially, a microcomputer was only a substitute teleprinter terminal. Microcomputers themselves were extremely limited; 4K bytes of RAM was a typical maximum. For the most part, they were the toys of computing hobbyists. Even matters such as the much-maligned eight-character filenames of today's IBM PC-DOS hark back to those severe limitations.

Xmodem-Checksum: The First Widespread PC Protocol

In 1978, a Chicago programmer named Ward Christensen set up a communications program for his bulletin board so programmers could transmit binary files with some degree of error control. Calling his communications program "Modem," the file transfer rules it used became known as xmodem. The Modem program still lives in the public domain, with purists of PC programming augmenting and improving it. Even some early commercial PC communications programs such as the smodem used with Hayes products can be recognized as variants of Christensen's Modem.

The file transfer protocol of Modem, however, has spread, spawning a number of variants of its own. Choosing which, if any, of the variants of xmodem to use for PC file transfer is one of the critical questions newcomers to PC communications must answer.

Christensen openly called his new protocol "a quick hack" and really did not maintain interest in it. Subsequent changes to xmodem have been made by others, who did not always identify themselves; however, the first xmodem is still used for PC file transfers. The original xmodem is a simple protocol—it merely sends a block with an error check appended and waits for the receiver to acknowledge or reject the block. If acknowledged, the sender proceeds to the next block. If rejected, the sender repeats the current block and waits again for an acceptance.

The block length of original xmodem is a fixed 128 bytes (or characters) and sends a checksum for its error checking. While ASCII parity checking is about 95 percent effective in identifying errors, the checksum form of xmodem is about 99.5 percent effective. While this is still less than common carriers demand, it works more often and better than many would expect.

When offered on communications lines or in PC communications packages, it is usually called xmodem-Checksum to differentiate it from the several other forms of xmodem.

The factors that caused Christensen to make his xmodem so simple still have value today. First, it is a very basic and compact program, easily realizable in high-level languages. Its short blocks require only a 256-byte communications buffer and run in a very small environment, making it quite useful in small, economical laptops.

Xmodem-Checksum has some very simple rules. The transfer always begins when the receiving station is ready to receive. There is usually a PC user at an outlying end waiting to receive a data file, so the automated main library waits to meet the user's convenience. This receiver-driven operational mode pervades most PC transfer protocols.

When an xmodem receiving station is ready, it sends one character, the ASCII NAK (Control U). Receiving that one character signals the sender to ship its first 128-byte block of information. That outbound information is packetized with a single "Start of Header" (Control A). Next, two characters follow for progress checking. The first is the

Table 2. Xmodem (CRC) Operations

SENDER		RECEIVER
(..first character sent...)		
	←	[C]
(..first file block sent...)		
[SOH][001][254][..128 bytes[crc-hi][crc-lo]→		
(..normal response from file receiver..)		
	←	[ACK]
(..error detection response from receiver..)		
	←	[NAK]
All other operations identical to XMODEM-Checksum		

sequential number of the current block. The second is the "ones complement" of the next block to follow. The ones complement is the remainder after subtracting the succeeding block number from 255. The 128 bytes of data follow, with the checksum, which is the result of summing the ASCII values of the 128 bytes being sent, dividing them by 255, and sending the arithmetic remainder.

Buffering 128 bytes, recomputing a local checksum, and comparing it against the received checksum is a relatively simple and rapid processing task, for which the receiving computer sends an ACK (Control F) to tell the sender to proceed to the next block in sequence. Where the checksum does not match, the receiver sends a NAK, which tells the sender not to send the next block but to resend the block it is still holding. The transfer proceeds with either ACKs or NAKs from the receiver pacing and prompting the sender until the file is completed, at which point the sender responds to an ACK with an EOT (Control D), which signals the receiver to revert to normal keyboard-controlled operation.

The system's simplicity, of course, limits helpful amenities. Xmodem has no means within itself to transmit the filename, for example, so the recipient must direct the incoming file to its destination; or the receiver's communications software must parse the filename from the keyboard setup transactions.

The inability to send filenames down the link also means that xmodem does not automatically cover file management. In complex organizations where multiple versions of the same file might be made and changed within minutes of each other at distances a continent apart, users cannot look to xmodem as a complete answer. This is an area where PC communications gives mainframe network people fits. While trends in the large machine and network world are toward segmenting layers of network operations into tidy, discrete functionalities, PC development and its protocols tend to combine all functions in one complex. Therefore, information such as the filename and date stamp transfer are added later in PC protocols, while mainframe networks are trying to place them in discrete layers.

Of itself, xmodem may be more OSI compliant than the later PC protocols that include added functionality. In fact, handling filename transfer and multiple-file transfers at a session are done in OSI style by the seventh major revision of Christensen's original program. In Modem7, filenames to be transferred are actually sent down the link as separate messages preceding the transfer. As a result, the operators enjoy an automated session (actually an automated series of xmodem sessions), albeit in a mode some programmers call "brain damaged." In Modem7 multifile sessions, the filenames are transmitted one character at a time and wait for a confirmation before sending the next character. This does not help the throughput or on-line holding time and should be repairable by making the filename shipment a separate message.

Experienced users of mainframe protocols will find many serious flaws in xmodem-Checksum; however, they are not all as valid in the PC environment as in the mainframe world. First, the process of sending a raw checksum down a transmission channel is not recommended by communications people. It is only about 99.5 percent successful at trapping errors, which is not desirable for commercial communications. Further, a checksum is particularly prone to being fooled by two successive incorrect bits. We should also consider that in PC communications, the data rates are frequently low and lines are commonly short. The majority of PC modems used now run at 1200 bps.

Adding Better Error Checking: Xmodem (CRC)

In a relatively short time, users of Christensen's original work began to add functions to xmodem to make it more broadly useful for longer distance, more demanding work. Among the first of these was to increase the error-checking capability. John Bryns added the commonly accepted Cyclic Redundancy Check (CRC) error-checking scheme to xmodem, resulting in the variant called xmodem (CRC). Using 16 bits spread across two character times, Bryns added CRC-16 to xmodem with the addition of only one character time per block. At the block size of 128 message characters, CRC-16 effectively traps 100 percent of errors of less than 16 bits, 99.997 percent of 17-bit error strings, and 99.998 percent of all errors of 18 bits or more. CRC-16 is so robust that it is the standardized error-checking method of CCITT Recommendation V.41 and has been adopted for years by many mainframe computer suppliers.

Further, Bryns made a rather clever change to the xmodem operation that permits automatic recognition of the use of CRC or Checksum operation. Bryns changed the receiving end's first action from a NAK character to the letter "C." Many automated file transfer sources (such as computer bulletin boards) can now automatically prepare their outgoing file blocks with either a Checksum or a CRC-16, depending on whether the controlling file receiver primed them with a NAK or a C. Virtually all PC communications software containing internal file transfer protocols now has both forms of xmodem operation available.

Detail of the CRC characters of xmodem (CRC) is as simple as the rest of xmodem. The first 8 bits following the 128 data bytes are the "high" digits of CRC, while the second 8 bits are the "low" digits. Except for the use of C as the first priming character by the file receiver, and the change and addition of a byte for CRC, xmodem (CRC) uses all the simple operations we explained earlier.

CRC-16 for transmission of message blocks has been in use for more than 40 years, dating to the era of relay logic, when its computation was an arcane art. Today, microprocessors make short, simple work of the computation.

To compute a CCITT V.41 CRC-16, the number of ones bits in the block to be protected is manipulated in the polynomial $x^{16} = +x^{12} + x^5 + 1$. After a division process to reduce the number, a remainder results. This is a number in the order of x^{1023}, which requires 16 binary places to describe. That number is sent down the link, where the file receiver recomputes it and compares it to the forwarded CRC. A match indicates no errors, for which xmodem will return an ACK character to the sender, and the sender will advance to transmitting the next block. A mismatch indicates an error, and xmodem returns a NAK, which will cause the sender to retransmit the same block. The computation process sounds daunting; however, in the C programming language of PCs, it comprises 16 lines of code and is performed in less than an eye blink.

Increasing Throughput: Xmodem (1K)

In similarly short order, the improved effectiveness of CRC-16 error checking was found to be so good that xmodem's block length could be increased. In fact, the overhead of 6 added characters to the 128 traffic characters of an xmodem block should allow for about 95.5 percent throughput, but the delays of line response time and wait for ACKs reduce this percent considerably.

A typical trial of sending moderate-length PC files across the country showed that xmodem with 128-byte blocks offers throughput efficiencies in the range of 55 to 65 percent at modem speeds of 1200 and 2400 bps, respectively, on dial-up lines with no errors. On a packet net, 128-byte xmodem suffered even worse, running only 35 to 45 percent at 1200 bps, and even slightly less at 2400 bps. The added delay of repacketizing data into the frame blocks of a packet network debilitate a 128-byte protocol. Using a protocol with 1,024-byte blocks in the same trials showed an improvement of 84 to 91 percent on regular DDD, and 79 to 84 percent on packet nets.

Increasing block length provides for a major improvement in throughput. Increasing xmodem's block length to 1,024 bytes helps considerably. It essentially reduces the number of stop-and-wait times by a factor of eight.

By the time of the increase to 1K packets, a number of varying camps were already using xmodem. Fortunately the various camps settled on a single way of identifying a 1,024-byte block. Also, it is another simple change. Instead of starting the message block from the sender with a SOH character, the 1K versions of xmodem use the STX character. Simple variations such as these make it easy for the writers of communications packages to provide many automatic functions in their programs. Conversely, many communications packages do not take advantage of these simple tricks because their writers do not understand the simple logic behind the variations in the protocols.

Overcoming Line Turnaround Delays: Windowed Xmodem

One interesting point about the development of PC communications is that its lack of standards simultaneously permits rapid moves to operation—with both good and bad ideas. One good idea for xmodem has been the addition of sliding windows. In the windowed implementation, xmodem's rigid stop-and-wait-for-an-ACK link is broken. Here, once started, the file sender runs ahead and sends several blocks without waiting for an ACK. If a NAK does come back, Windowed xmodem backs up that many blocks and retransmits them. Adding windowing to the 128-byte blocks of xmodem helps throughput considerably, placing it close to that of xmodem (1K).

For whatever the reason, there seems not to have been any merger of Windowed xmodem with xmodem (1K). This should have resulted in a very efficient protocol. Rather, it seems the proliferation of PC protocols that broke the OSI rules by packing the filenames and date stamps into the protocol block has overtaken such a development. Several proprietary variations of Windowed xmodem may be doing so, such as CompuServe's CIS-B and Quick CIS-B protocols, both variations of Windowed xmodem.

Cousins of Xmodem, Not Descendants

The main thread of xmodem-like protocols shifted to the Pacific Coast when Chuck Forsberg produced ymodem. Forsberg was an early worker in the UNIX domain and produced a protocol he called "Yet Another Modem," or YAM for the UNIX environment. Recompiled for CP/M and PC-DOS, he named it ymodem. More recently, Forsberg was commissioned by Telenet to produce what should be the best protocol for use on packet networks, so he christened that improvement zmodem.

Ymodem

At its outset, ymodem used 1,024-byte blocks and CRC-16 error checking, making it as efficient and effective as the best xmodem. Also, in the best OSI-shattering tradition of PC work, Forsberg included filename and date-stamping into ymodem. For xmodem, Christensen had started counting file blocks with 1, not 0. Forsberg added a block 0 to ymodem's rules. In ymodem, block 0 is the bearer of the filename information for a batch transfer.

Table 3. Structure of a "Generic" Kermit Frame

```
"Length byte"
          "Type byte"
                "Sequence # byte"
                           "Data field"
                                 "Error Check byte(s)

[SOH] [ x ]   [ y ]     [ z ] [1 to 857,374 bytes] [1 to 3 bytes]
```

Since ymodem was, by original definition, a 1,024-byte protocol, Forsberg had his first revisions use the SOH character to lead blocks. This utterly confused xmodem (1K) receivers that expected SOH to indicate a 128-byte block following. Later revisions corrected the problem, and properly compatible ymodem operations can even switch between long and short blocks when error conditions require. Many erroneous implementations exist on bulletin boards around the country, however, no small number of which are actually xmodems (1K) that get knocked down by receiving a packet 0.

Another variant of ymodem, called ymodem-g, is meant for exclusive use with error-correcting modems of the MNP or X.PC variety, or on packet channels that are intrinsically error correcting. When such channels are used, the Physical Layer checks errors and corrects them; doing so in the file transfer layers of a system would be redundant. Ymodem-g sends no error checks until the end of the file. This makes throughput on unprotected modems seem blisteringly fast. If one error gets into a file, however, the whole transfer session is wasted. Ymodem-g should be used only with error-protected channels or on very local hard-wire links totally free of errors.

Zmodem: The Ultimate Xmodem?

The packet net problem for PC protocols has had several solutions. As previously mentioned, CompuServe addresses it with windowed proprietary protocols derived from xmodem. GEnie uses ordinary xmodem but does so with interfaces physically located at its local dial-in ports, where its packets can be stuffed optimally to overcome problems. Telenet, attempting to avoid the need for hardware around the nation, commissioned Chuck Forsberg to develop a suitable protocol in 1986. The result was zmodem, a public domain protocol.

Forsberg simultaneously produced a shareware-distributed program called DSZ.COM for DOS PCs, however, and commercial programs for the UNIX and VMS environments. Thus, considerable availability of this very advanced protocol for PC-to-mainframe or PC-to-mini links is provided.

Zmodem combines all the technological advances previously mentioned and adds a few of its own. Perhaps most significant is zmodem's streaming operation. Datastreaming permits nonstop data transmission in the absence of errors. With streaming, there is no stop-and-wait until the file receiver detects an error. It answers quite well the limits of throughput on satellite or packet links. When the transmission is in error, a recovery means is needed, and zmodem's is as advanced as any. If the zmodem file receiver detects an error, it initiates a reverse message telling the sender how many blocks to back up and resend. Zmodem also can recover from a failed transfer, with the file receiver telling the sender the block number to restart.

Zmodem provides full directory listings for transmitted files along with time/date identity for those files. The time/date system of zmodem surpasses that of the DOS provided for IBM PCs and compatibles. The zmodem time/date system is set in global Universal Time Coordinated (UTC, for practical purposes the same as GMT) and relates back to January 1, 1970.

Other Eight-Bit Protocols and the Future

Less widespread use is made of numerous other variants of xmodem. Each has its own salient features and, in some cases, very vocal adherents. Having names such as Sealink, jmodem, and bimodem, each has its combination of error-checking schemes, full-duplex operation, or windowed operation, to advance beyond those first steps of xmodem a little more than a decade ago.

As to future developments, the problem eight-bit PC protocols seem to have is a rather flagrant ignorance of the OSI system layers and interfaces. Rather than conform with OSI, they seem to ignore it. Users may have to deliberately depart from OSI planning in order to service PCs. Some users are following that path. Not the least of these is AT&T with its Electronic Mail service, ATT-Mail, using its own proprietary ymodem251 protocol to batch upload and download E-Mail messages to IBM, Apple, and UNIX PCs on that network. Some seven-bit protocols fit OSI more closely, but at considerable throughput expense.

Table 4. "Generic" UUCP-g Frame Using Control Byte

```
"data length bytes"
                   CRC-16
                          "control byte"
                                 "x-or byte"
                                        data field

[DLE] [L1] [L2]    [lo] [hi] [ttxxxyy] [x-or] [. . 0 to 4096]
```

Seven-Bit PC Protocols: The More Conventional Way

In 1981, some establishments began to realize the need to establish file transfers for microcomputers in the office. Protocol work in the establishment area has paid more attention to the Open Systems Interconnection model of the ISO; however, it has also been largely optimized for the standard seven-bit characters of ASCII (International Telegraph Alphabet #5). This presents some real throughput problems for the simplified "PC way" of using the eighth bit for extending the code to 256 characters. The largest proportion of PC files are in those eight-bit characters. Integrators of PCs into large systems networks, therefore, must make some hard choices, at least in the PC-DOS environment—compliance with OSI or reduced throughput. The following examines some establishment answers.

Kermit: The Seven-Bit Standard

While working on a solution to connect student and departmental PCs to minicomputers at Columbia University, Frank da Cruz and William Catchings formulated an evolving standard. Its name typifies the whimsy of computer educators, coming directly from the redoubtable frog of television fame, Kermit. Its character evokes that of its namesake, constantly trying to do all things that everyone wants, succeeding at some, suffering false starts and retries at others, until it finds the best way. The result has been many revisions and iterations of Kermit, with many implementers of the protocol running generations behind.

For instance, Kermit originally was intended for file sharing with minicomputers and was structured around seven-bit information character slots.

It had only one way to handle eight-bit characters: divide them and ship seven bits of a character in one slot and the eighth bit in the next slot. The effect on throughput of PC binary files is obvious. It was (and is) a penalty users of earlier versions of Kermit had to pay.

Many users of earlier versions maintain negative opinions about Kermit. More recent versions use some method to signal that a following sequence is in the extended part of the character set of PCs. Used in various ways at various revision levels of Kermit, a "quote byte" contains a character (most typically the ASCII # for control characters and the & for high-order bit set "true") to identify nontext characters requiring retranslation for the receiver. So many revisions of Kermit exist that not everyone knows there are variations on "quoting" and that even the most capable versions automatically set up and agree on what the "quote" character will be.

Many of the earlier versions of Kermit are a result of its wide use in many environments. Implementations of Kermit exist for the widest variety of computers imaginable in languages as diverse as APL for Unisys machines and PL/I for Prime minicomputers. Some of the better mainframe implementations provide a server mode that allows all commands to be given from the remote end. The server mode is similar to what many PC users find on their PC bulletin boards.

The earliest Kermit versions were built to run in what was then the very small memory environment of a PC. As a result, those early versions ran a block size of only 91 bytes, even smaller than the original xmodem and less transfer efficient. Since 1986, longer blocks have been used in newer Kermit versions that permit options of long packets of 9,024 bytes, extra-long packets of 857,374 bytes, sliding window techniques, detailed file information, encrypting of data on the link, and other functions. Each of these is complicated by variable error-checking methods. These newer improvements, particularly the sliding window technique, have increased Kermit's complexity. As a result, many of the mainframe implementations have yet to include it.

Kermit is, however, well kept by its custodians at Columbia University, who distribute revisions in both disk and source code form for nominal fees. Those of a dial-up mind can download recent revisions from BITNET or ARPANET.

Perhaps the greatest problem in using Kermit is the tremendous variety of implementations, particularly those provided in various PC communications programs. They can often be some different vintage (usually earlier) than a particular host machine. A later vintage in both ends of the link should result in a version of Kermit as capable as any PC protocol.

The UNIX Answer: UUCP Packet Protocol-g

The UNIX protocol, UUCP, dates back as far as xmodem (1978). It has also been revised and updated, but one version in particular seems to be fairly widespread, called UUCP-g, or sometimes "PK" for Packet Driver. UUCP-g operates much like IBM's SDLC in that it has in its envelope an eight-bit control byte that carries information about the transfer (instead of the several bytes in fixed positions other protocols use).

For UUCP, Bell Labs developed a unique means of telling the receiver what the data field length is, and an unusual element called the "exclusive-or byte" that functions as a special error check on the header itself.

The length byte code represents the field length as the variable quantity "k," where the field length is expressed as $2k^{-4}$ bytes, up to a maximum of 4,096 bytes. This makes the range of numbers in the two data length bytes from 04 for a data field length of 0 (if ever sent) to 16 if signifying a field length of 4,096, in increments of the powers of 2. Observers state the most common value of "k" seen is 10, for 64-byte frames in UUCP.

UUCP-g's control byte, however, is rather clever and may or may not be why IBM decided to use a similar code. The control byte of UUCP-g has fixed bit positions, called "t", "x", and "y" bits as follows:

Position in control byte	8	7	6	5	4	3	2	1
Bit Name	t	t	x	x	x	y	y	y

Meanings: tt

00 This is a control packet
01 Alternate channel (obsolete)
10 This is a data packet
11 This is a short data packet

xxx (sending signal)	yyy (received result)
001 CLOSE	001 no message
010 RJ (reject)	010 last correctly received packet
011 SRJ (selective RJ)	011 packet # to send
100 RR (receiver ready)	100 last correctly received packet
101 INITC	101 window size to set
110 INITB	110 data segment size (=32 x 2^{yyy})
111 INITA	111 window size to set

To further mystify the control byte, when in data packet operation, the xxx and yyy change to the current outbound packet number and last received valid packet number expressed "modulo 8." Some observers have noticed that older implementations of UUCP-g set the window size at three, which is too small for current higher speed modems and longer channels. If not identified and corrected, this small oversight can result in poor throughput. Obviously, it could be easily fixed, as could an earlier typical packet length of only 64 bytes.

Finally, UUCP-g frames contain another unusual element, the "exclusive-or" byte. It functions as an error check on the preceding header bytes, a rather nice precursor of OSI, dating from times long before OSI. With this, UUCP-g provides a means to strip off the message data for processing at higher layers, with assurance that its handling at lower layers was correct.

Error-Correcting Modems: More Confusion or Less?

Modems are now available that provide error detection and recovery buried within the Physical Layer of data lines, where the OSI Model never intended it to be. That is a very attractive location for error control, as it stays in that hardware layer most avoided by data processing people. If, in fact, it permits the purchase of lines and modems with a vendor guarantee of error-free operation, it is attractive. It removes a degree of network control from the user, however. Now, if operation stops, it requires first a call to the modem supplier. If the modem supplier places the blame on the phone line, another delay ensues while the arguments begin.

Table 5. A Sampling of PC Communications Software Programs

NAME	SOURCE	COMMENTS
Procomm	Shareware	v.2.4 is last and final shareware version, has numerous internal protocols, including earlier versions of KERMIT and SEALINK. Can accept external protocols.
Procomm Plus	Commercial	v.2.4.2 sold widely by software and PC dealers. Features as above.
Bitcom	Commercial	Supplied with many "traveling" and "pocket-sized" modems. Contains only XMODEM public-domain protocols; needs external to get better function.
Smartcom	Commercial	Supplied with Hayes-made modems. Many versions in field; most have only XModem internal.
Telix	Shareware	v.3 and later have many internal protocols including ZMODEM.
QModem	Shareware	v.4 is a programmer's dream, but might be too complex for users to handle.
Boyan	Shareware	v.4.01 has several PD internal protocols; needs DSZ.COM in order to run ZMODEM.
Telemate	Shareware	v.2.0 beta-testing at this writing; has several internal protocols including ZMODEM, plus somewhat clever multitasking for the PC-DOS environment.
PC-Talk	Freeware	Given out freely by its author to all who send in a blank diskette. Current status unknown.
{COMMO}	Shareware	v.3.30 works quite smartly in even the smallest laptops. No internal protocols, but macros for DSZ.COM built-in.
Red Ryder	Commercial	For the MacIntosh environment; well-developed, supports many protocols.

Further, error-correcting modems implement their own protocol on data down at that layer. If the user is blocking up data and error checking it in higher layers by using a protocol similar to the types we have described here, it can cause the same sort of throughput delays that would occur if used on a packet network. The error correction of these modems is generally much like X.25 as used in a packet network. In particular, MNP Class 2 is virtually X.25 run between a pair of modems. In the worst possible scenario, there could be two layers of error correction in the software and modems when a user dials up on a packet network to send a file.

Of course, the user can overbuy and then shut off some of the error-correcting horsepower by using ymodem-g or zmodem with error-correcting modems. But even in this case it is wise to shut off the error correction in the modems when using packet lines.

How to Use Protocols in PCs

The key to sorting out this confusion lies in External Protocols. Rather than selecting a package for its built-in line protocols that may or may not be current with implementations already in place elsewhere, PC software is available that permits recording a track of the necessary protocol in the same directory of the PC as its communications program. Many PC communications programs provide for external protocols. Many of them already have macros built in for the more common external protocol packages, most notably the DOS program DSZ.COM from Forsberg's Omen Technology. With DSZ.COM added to any program, all the variants of xmodem, ymodem, and zmodem are ready in the PC. Omen also has UNIX versions.

Many PC communications software programs come with a variety of communications protocols and capabilities for external protocols. Another problem often encountered when dealing with the proliferation of PCs is the demand for communications with a PC that already has communications software. If there is some knowledge of what the PC needs to be compatible, there is at least a possible answer. (Perhaps a good answer is a diskette of Kermit or the address of Columbia University, *providing the user with the name of the requisite* external protocol software!)

Table 5 lists a sampling of available PC communications software programs.As a general rule, the more "commercial" a PC communications program, the less its range of protocol choices. The shareware authors must make their products attractive by being easier and more flexible than "store-bought" programs. Some contain such a range of internal protocols, however, that their external selections are limited to one or two.

Bibliography

Our research in presenting this report yielded many nontraditional sources of information. In keeping with the informality of the development of PC communications protocols, much of the information is in magazine articles as opposed to hardbound books. Here is a selection of what seemed to us to be the better material.

"Picking the Proper Protocol," *PC Magazine*, June 11, 1985, p. 355 ff. "Modem7 and Downloading," *ONLINE magazine*, November, 1985, p. 57 ff. "Choosing and Using Modems," *PC Magazine*, April 29, 1986, p. 233 ff. "Public Domain Protocols Fill Void for Async File-Transfer Standard," *PC Week*, September 8, 1987, p. C3. "The Downloading Zone," *Personal Computing*, March, 1988, p. 89 ff. "The ABCs of X-, Y-, and Zmodem," *BYTE Magazine*, February, 1989, p. 163 ff. "The Protocol Pack," *BYTE Magazine*, March, 1989, p. 155 ff. "PC Communications and Protocols," *Programmer's Journal*, July/August, 1989, p. 44 ff.

The Complete Handbook of Personal Computer Communications, Glossbrenner, Alfred, New York, St. Martin's Press, 1983, ISBN 0-312-15717-7.

Communications and Networking for the IBM PC, Jordan, Larry E. and Churchill, Bruce, Bowie, MD, Robert J. Brady Company div. of Prentice-Hall, 1983, ISBN 0-89303-385-5.

Computer text file: "Xmodem/Ymodem Protocol Reference . . . A Compendium of documents describing the Xmodem and Ymodem file transfer protocols," edited by Chuck Forsberg, originated from Forsberg's Telegodzilla electronic bulletin board, undated, appearing on many bbs systems as "YMODEM.DOC." Reprints are available from Mordon Services, Box 1776, Safety Harbor, FL for $10 cash for postage and handling. (A selection of the base documents for XMODEM and YMODEM.) ■