Higher Protocol Information Transfer

by

Brian P. Makare

Dept. of Computer Science

Submitted in Partial Fulfillment of the Requirements of the University Undergraduate Fellows Program

1985-1986

Approved by:

Bart Childs

Dr. Bart Childs

Abstract

Most computer terminals only allow the transfer of text to the computer. Often there is a need to send various forms of nontextual data, such as binary (compiled programs) or graphic, from a terminal to a larger computer. This can be done through the implementation of a system using a higher protocol than normally used for terminal programs. Such a system is to be implemented at Texas A&M. The system will allow students to access more programs within the Texas A&M system, and allow for more flexible communication between the student's terminals (microcomputers) and the larger Texas A&M computers.

Acknowledgment

I would like to thank Dr. Bart Childs for all his time and patience through the past year. I would also like to extend my appreciation to Marcus Montalvo and Scott Boyd for their help and support in getting this project accomplished. But most of all, I would like to thank Jennifer Mitcham for her dedication and loving support during the past year.

List of Figures

Table of Contents

Introduction

With the continuing growth of networks through the 1980's, the need for file transfer protocols seems to be slowly dwindling. Networks, such as Arpanet and Bitnet, extend their control across the United States and beyond. But networks are expensive to install and operate and may not be readily available at all times. For these reasons, the availibility of point to point communication between two computers is still a needed facility. This is especially true in communications between microcomputers and main–frame or mini computers at a remote site. Most people who own personal computers could not afford to have a network line installed at their home. So the need to develop file transfer protocols still exists.

This need to develop file transfer protocols is especially prevalent in an academic atmosphere. Large numbers of university students interface with computers on a daily basis, demonstrating the need for quick, easy access to information. Uploading and downloading files from the university mainframes to personal computers for updating and storage would free the larger computers for other activities.

Kermit[1] is a higher protocol information transfer system which was developed in an academic atmosphere to meet the needs of the academic community. It was originally developed in 1981 at the Columbia University Center for Computing Activity by Frank da Cruz and Bill Catchings. Since that time, it has been implemented on a large number of computer systems across the nation. In the fall of 1985 and spring of 1986, I undertook the task of implementing the Kermit protocol on the Data General MV10000 computer at Texas A&M University to allow students more flexibility in communicating with other computer systems.

[1]**The name comes from Kermit the Frog, MC of THE MUPPET SHOW, and is used by permission from Henson Associates, Inc.**

Body

"The drive to develop computer communications protocols has as its goal nothing less than a new breed of cooperatively developed, standardized protocols that will allow every vendor's computer to communicate easily with every others"[7, page 64]

According to Random House Dictionary, the definition of protocol is *" the customs and regulations dealing with the ceremonies and etiquette of ..."* a system. In this case, protocol refers to *"a set of rules for forming and transmitting information carried out by programs which embody those rules"*. Designing and implementing a protocol for communications is a difficult task. Standardization of communications protocols aids in the design and implementation process. One set of communications standards has gained a lot of recognition in the United States in the past few years, Open Systems Interconnection (Created under the direction of the International Standards Organization). This standard outlines a layered approach to development of communication software. Seven layers are described; the physical layer, data link layer, network layer, transport layer, session layer, presentation layer, and application layer. Layers are grouped so that similar levels on different machines can communicate. Kermit's file transfer protocol can be defined in terms of these seven layers.

Physical Layer :

The physical layer is the hardware on which the software is operates.[7] Kermit is supported on a multitude of computer systems.[1]

Main–Frame and Mini Computers

- Cyber Systems
- Digital Equiptment Systems
- Data General
- Burroughs
- Vax
- IBM

- Cray
- Honeywell
- Hewlett–Packard
- Prime
- Tandem
- Sperry

Microcomputers

- Apple
- NEC
- Apollo
- Atari
- Commodore
- TRS
- Fujitsu
- Honeywell
- Hewlett–Packard
- Digital Equiptment Corp.
- IBM
- Tandy

Portability, the quality of being easily implemented or transported between computer systems, is an obvious characteristic of Kermit. A small kernal of system calls needs to be changed to meet the needs of a new computer. Also, the protocol is not oriented towards features of any one language to aid in maintaining compatibility across systems.

Several distinctions between systems need to be noted as considerations when developing a communications protocol. One of the biggest is the difference between mainframe and micro computers. Besides the size and expense differences, speed of the central processor, availability of multiple user operating systems, size of buffers, commands, character sets and control characteristics can be different. The overall differences in power between the two types of computers needs to be accounted for.

Characteristic problems occur in flow control. A computer can utilize full or half duplex transmission. Full duplex implies that information can be transferred both to and from the host computer simultaneously during a communication session. Half duplex communication takes place in one direction, with the host computer sending out a message which is acknowledged by a remote computer or terminal, and then a response is sent to the host computer from the remote site.

Another problem area involves buffering the influx of data during a communication session. If data is transmitted from a site faster then the receiving computer can accept it, the buffer of the receiving computer may overflow, causing the remainder of the transmission to be lost. This can happen when mainframes with large buffers communicate with, slower microcomputers with smaller buffers. It can also occur when a remote computer is sending information to a host computer which is overloaded. The high activity level may slow the response time of the host to such a degree that it is unable to clear its buffers fast enough. Several methods can be used to handle this problem. The data stream can be regulated using *xon/xoff* flow control. Modem control signals can also be utilized to slow the inflow of information. A final method would be to minimize the data bursts so they cannot overflow the buffer, such as to the length of a single line (approximately 80 characters).

Problems arise when one computer system attempts to interpret special characters sent by another computer. Special characters are often handled by system dependent functions. These characters can involve padding, control (*linefeed, carriage return, delete, nul, space*) or printable characters. Although American National Standard for Information Interchange (ASCII) is the most commonly used character set, others, such as EBCDIC, are still used.

Deadlock can occur when a computer system does not provide capabilities for timing out. A process should terminate if it stays in a static state for a certain period of time, as determined by the system or the process itself. A deadlock sitution can occur if a communications program is waiting for a response on one end of the connection, but for some reason, its signal was never received, so it's twin process on the other end of the connection is also waiting for a signal. If no time–out capability is provided, both ends of the transmission will wait indefinitely for the other to signal the continuation of communication.

Deadlock, buffering, flow control and the differences between mainframe and micro computers are problems which communication protocols must be designed around. The software must be able to initiate action above the level of the hardware, or it must find a way to disable special characteristics on the hardware level to meet the standards of the physical layer.

Data Link Layer :

According to the International Standards Organization, a connection between the hardware and the software should be established in the data link layer.[7] The hardware should recognize messages passed to it by communications software. An example is terminal input and output, where the communications package, normally through systems calls, configures terminal characteristics like the speed of data transmission (baud rate). Opening and closing modem communication is another example of a software to hardware link. Although these actions are not activated in the data link layer of the protocol, the communication necessary to undertake the operations is implemented. Designers of the Kermit protocol took an interesting approach. To insure that the information is correctly transferred, they chose to describe the format used to transfer information in the data link layer.[1]

Kermit uses packets to transfer information, as recommended by a European standards organization, CCITT.[4] A packet consists of a string of characters which represents both data and control and includes a description of the data. Certain characteristics must be taken into account when designing a packet. What fields will be used to delimit the packet externally? Which fields, internal to the packet, will be used to delimit the data? What is the best arrangement for all of these fields? Finally, how should the data itself be represented?

The smallest unit of information in the Kermit protocol is an ASCII character.[2] In fact, Kermit only transfers information using the ASCII character set. Versions of Kermit operating on systems using other character sets must provide functions for translation of the information. In an effort to address the buffering problem described earlier, packets are limited in length to a range of 40 — 94 characters. This range allows kermit to fit into the standard line length of most machine and therefore conform to the appropriate buffer size.

Kermit allows information to be represented by the ninety–five printable ASCII characters. These characters are, in turn, translated to integer values 0–94 for transmission. ASCII characters have numeric values ranging from 0 to 127. Values from 32 to 126 are printable, while values from 0–31 and 127 represent control characters. Two simple functions are defined by Kermit to translate printable characters to integer values and back.[2]

$$char(x) = x + 32$$

converts an integer value into a printable ASCII character.

$$unchar(x) = x - 32$$

converts a printable character into an integer value.

In the *unchar* function, x is the numeric representation of the character. The result of the *char* function is the ASCII character determined by the numeric value.

A Kermit packet is divided into six sections. Five of these sections are control fields, the other is data. In an effort to minimize the packet size, control fields are limited to single characters. The control fields which delimit the packet beginning and end, and the control character use in error checking are defined in the data link layer.[1]

Two of the five fields used for control indicate the beginning and end of a packet. (Figure 1.) The mark field is the first field in the packet, and indicates that a new packet is about to start. It is commonly represented by the ASCII start of header *(SOH)*. Immediately following the mark field is a length field which designates how many of the following characters are part of the packet. By using a fixed length, Kermit helps to insure all file information will be transferred correctly. Extraneous characters between the end of one packet and the mark which begins the next are ignored.

The largest cause of errors in packet transmission, indeed in communications, is simply termed *noise*. Noise refers to any form of electrical interference. Causes of interference include electrical storms, static electricity, poor communication lines or other machines operating in the vicinity of a computer system. Another error causing problem is synchronization between communication devices. Systems operate at different speeds, as do communication devices and even communication lines. Errors may be caused by general line outages, where a connection is dropped prematurely due to mechanical failure, or a line is cut. These errors are not immediately correctable.

An error detection character is the final field in the packet. Several error detection and correction methods with differing degrees of implementation difficulty are available for use. Vertical Redundancy Checks[3], such as Hamming codes or parity checks, detect errors on a per character basis. These codes are expensive in terms of resources because every character is checked. They may also be difficult to implement because not all systems allow free manipulation of the seventh data bit. The bit may already be used for system parity, or may be necessary as a portion of the data, as in binary files.

Cyclic Redundancy Checks (CRC)[3], pass each character through a shift register. A block check character, or several check characters, are formed in which each bit has been affected in many ways by the data. CRC's form some of the most sophisticated error detection methods, as well as some of the more difficult to implement. They center around sixteen bit quantities, rather than eight bit, which are broken into two or three characters and summed through extensive high order polynomials. A three bit CRC is capable of detecting double bit errors, messages with an odd number of bits in error and all errors in information bursts shorter then sixteen characters.

A final method of error detection is the Longitudinal Redundancy Check (LRC)[3]. Block check characters are formed by a combination of a sequence of

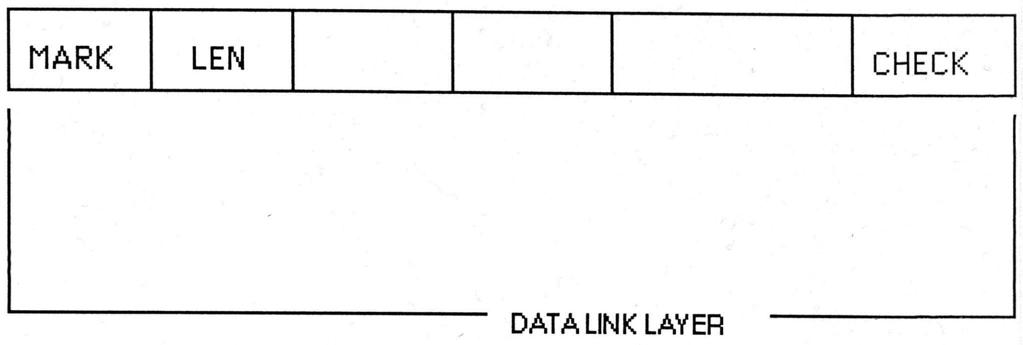


Figure 1. Packet fields defined in the data link layer.

characters. One example of an LRC, column parity checking, involves taking the exclusive or of the correspnding bits of a group of data characters.

Another type of logitudinal redundancy check, recommended for use with the Kermit protocol, is single character check summing[1]. The arithmetic sum of numeric representations for all characters in a packet makes up the check. This sum excludes the *mark* character, *SOH*, and the checksum itself. In the worst case, the result approached 12,000. Binary representation of 12000 takes 14 bits, more than the 8 bit character allowance. Correction for this problem is simple if it is possible to discard the high order bit (every character in the sum contributes the low order 7 bits, 0 — 6) and still maintain the integrity of the check sum. Some systems add information to the eighth bit, such as parity, and binary files require maintenance of all 8 bits in a byte. In cases where it is impossible to ignore the high order bit, the seventh and eighth bit can still be manipulated into the block check character by adding them back to the low order bits. The sum of all the characters thus computed is pushed through the equation $(s + ((s \textbf{ and } 300)/100)) \textbf{ and } 77$ to find the value of the single character check sum reflecting every bit in the packet. The value of the character computed when the packet is sent is compared to the value computed upon receipt to determine if an error has occured.

Single character checksums are single error detecting. If an error is detected, the Kermit protocol will request that the packet containing the error be resent. The probability of an undetected error is the ratio of the number of possible errors which cancel each other out (leaving the checksum with a correct value) to the total number of possible errors. This ratio approximates $1/2^n$, where n is the number of bits in the checksum.[1] A single character checksum leaves about 1.56% of the errors undetected. Thus single character checksumming is a fairly effective method of ensuring information in Kermit packets is correct.

Network Layer :

The network layer represents interconnection between computer systems, the level at which actual communication is said to take place. Two modes are established by the International Standards Organization, connection and connectionless.[7]

Connection mode allows data to be transmitted between two end systems without worrying about how networking of these systems is achieved. Data packets are sent sequentially and reassembled at the receiving end. Handshaking is allowed. Handshaking, loosely defined, is when the sending computer receives acknowledgement that the data packet was received. A network operating in connection mode is incapable of routing and relaying information between nodes for internetworking purposes.

Connectionless mode is sometimes termed *Internetworking Protocol.* Single, unsequenced data packages are passed. Unlike connection mode transmission, messages can be routed within the network. The network is capable of making its own routing decisions for a particular communication.

Frank da Cruz argues that the Kermit file transfer protocol does not contain a network layer because a point to point connection is established between two computers, not intercommunication between multiple systems.[1]

Transport Layer :

Reliable data transfer and actual data movement are normally accounted for in the transport layer. The transport layer was only recently established as a portion of the ISO Open Systems Interconnection standards. It came about because of differing needs for communication reliablility. Five classes of reliable transfer were established ranging from 0 to 4. Class 0 is straight transmission of information, no buffering, no modification. Succeeding levels include some amount of error detection of correction. Level four is single or better error detection and a definite form of error correction.[7]

The transport layer was established after the Kermit protocol had already been designed and implemented[2] and is therefore not considered to be a part of the protocol. Kermit, however, would be classified at level four because the single character checksum established in the data link layer is single error detecting and the protocol can request that information be resent when errors are detected. Reliable data transfer is established in the data link layer rather than the transport layer.

Session Layer :

The ability to request that information be resent when errors are detected is one part of the session layer. Generally, the session is responsible for the establishment and the termination of a communication session and the design or structuring of a communications dialog. The sessions layer consists of a kernal responsible for three operations, connect, data transfer and disconnect.[7]

In the session layer, another control character of the Kermit packet is defined, the sequence number.[1] (Figure 2.) This number is checked by Kermit for duplicates or for a missing number in the packet sequence. Duplicate packets are simply ignored. If a packet is missing, Kermit can request that the packet be retransmitted of that file transfer be dropped if the error proves to be unrecoverable.

The most important operation in the session layer is the establishment of how communication is to take place; determine who is to talk and for how long. Kermit is designed with communication between micro and mainframe comput-

[2]**Recall that the protocol was initially designed in 1981**

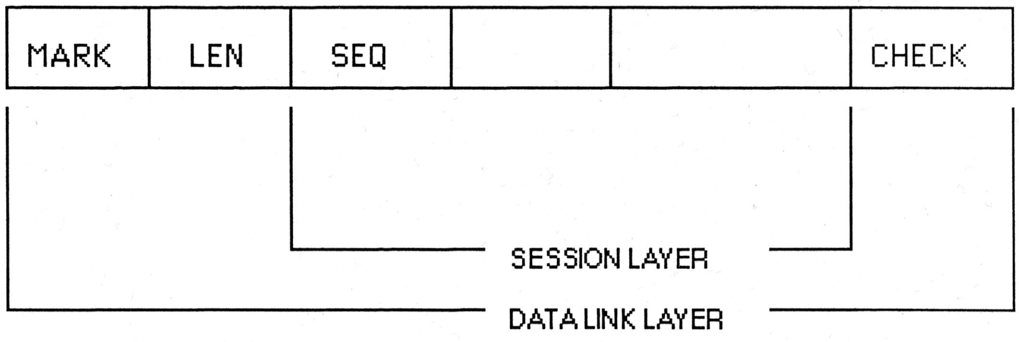


Figure 2. Packet fields defined in the data link and session layers.

ers in mind, indirectly implying asynchronous communication. Asynchronous communication is event driven. Actions are taken in response to signals, commands or an expected series of operations. The most common asynchronous port is the RS232. It allows sending of information serially in digital form upon a software defined event.

Asynchronous communication between micro and mainframe computers commonly takes place over a telephone line with the aid of a modem. Modem stand for modulator–demodulator. It's function is to translate digital signals from the computer into analog signals recognizable by the phone system (and back from analog to digital).[5] Digital signals are represented by a precise series of 1's and 0's, where one stands for *on* and 0 stands for *off*. They reflect binary representations (representation to the machine) of data. The telephone system understands analog information, series of fluctuations in electrical voltage. The farther these signals have to travel, the more distorted they become, so the telephone company uses filters and amplifiers to maintain the integrity of all signals between 300 and 3300 hertz. All other signals are discarded. The modem is responsible, then, for translating series of *on/off* pulses into electrical voltages in the range of 300 to 3300 hz.

The most recent technique used in modem design is Four Phase Differential Phase Shift Keying.[5] Two sets of tones are used to distinguish the transmitted signal from the received signal. The initial connection end is the transmitted signal, the originator, and is assigned a tone of 1200 hz. The answering end, received signal, is assigned a tone of 2400 hz. Communication starts at a particular point on the analog sound wave (sine wave). A marker is set at a point on each wave representing two bits of data. The four combinations possible for two binary digits determine where the marker is set on the wave:

- top of wave
- bottom of wave
- midway between top and bottom on *up* side of wave
- midway between top and bottom on *down* side of wave

The receiving modem checks for the marker at each of these four position to check the value of the incoming two bits of data.

Four Phase Differential Phase Shift Keying has proven to pass information effectively at rates up to 19,200 baud, approximately 1,920 characters per second. The highest commonly used modem communication speed is 2400 baud, or approximately 240 characters per second.

Asynchronous communication over a phone line can take place at two levels of flow control, full duplex or half duplex. Using full duplex lines, both computers can send and receive information simultaneously. Half duplex lines only carry one communicator at a time, so a method of determining which end of the commection is active is needed. As discussed in the physical layer, some machines only support half duplex communication. Systems which support full duplex lines can be used at half duplex by waiting for one end of the connection to quit sending information before allowing its mate to start.

Kermit's protocol always runs at half duplex in an effort not to exclude any system from its realm of possible implementations.[2] In the data link layer, a beginning mark and a packet length were defined to determine the initiation and end of a communication phase. Once a packet has been sent, the receiving end is able to transfer information. The information transferred is a special packet called an acknowledgement. If the information transferred was correct, a positive acknowledgement is sent, otherwise a negative acknowledgement is sent and the information sending computer is prompted to resend the same packet, determined by its sequence number. On a positive acknowledgement, communication continues with the next packet in the sequence being sent.

Presentation Layer :

> For communicating computers and applications to understand the information transferred by computers with different data representations, they must use common syntax, which represents information such as character codes, alpha numerics, data types and file formats.[7, page 78]

The presentation layer is responsible for negotiating the syntax used when two applications exchange information.[7] This includes file formats, data structures, control characters and the context in which these are valid. As this information passes through a protocol hierarchy, it must be found, identified and interpreted at the appropriate level.

ASCII characters make up the basic building blocks of all the information in a Kermit packet. Several encoding methods manipulate the characters to reduce the physical amount of information transferred and to take into account special control characters passed in the data.[1]

- Logical records are terminated by addition of a carriage return/line feed.
- Non–printable control characters are transformed into a printable character prefaced by a "quoting" character. The most common quoting character used is $\#$. The transformation is performed by complementing bit 6[3] and then take the result modulo 64. Control–A would become $\#A$, for example.
- Required line terminators for delimiting packets to the system, such as when a command is transferred, are not included as part of the packet. Terminators are invisible to the Kermit protocol, as are any characters which are passed for system control and occur between specified packets.
- Limited data compression is used to reduce physical information passed. Data may contain long strings of redundant characters. These strings can be compressed into 3 characters. The first character indicates a redundant string, the next character's numeric value determines how many characters were in the string, and finally the redundant character itself is given.

There are other options which are available for manipulation of data but are not used in the Kermit protocol.

- Huffman encoding, a form of increased data compression. If the data transferred is known to represent English text, mathematical models can be formulated based on the known frequency of character use. The most frequently used characters can be represented by short bit strings, while more unusual characters use long bit strings (up to 8 bits). This method of compression backfires if unusual character sequences are sent.
- Nibble encoding attempts to circumvent problems with control characters and 8 bit characters by dividing every byte into 4 bit *nibbles*. Each nibble is then sent as a printable character. Because translations have to be made on each character and 8 bit characters must be reconstructed from the 4 bit nibbles, there is a 100% overhead associated with this manipulation.

Besides defining data structures, the presentation layer is also responsible for describing file structures. File organizations include linear of record oriented, their contents and name must be verified and external attributes must be specified.

[3]**A simple mathematical formula represents this transformation x XOR 64 (add or subtract 64 from the numeric representation)**

MAXL	TIME	NPAD	PADC	EOL	QCTL	QBIN	CHKT	REPT	CAPAS

Fig. 3 Send-init Packet data

The configuration of the system for file transfer is tranmitted by Kermit in a send–init (send initial configuration) packet.[1] (Figure 3.) The fields of this packet are defined as:

1. *Maxl* — Specifies the maximum packet length the protocol can receive or send. The Kermit default is 94 characters. The minimum feasible packet length is 40 characters.
2. *Time* — The Kermit protocol will wait this long to receive information before time out occurs and file transfer is discontinued.
3. *Npad* — Number of padding characters preceeding each packet. The Padding characters give the receiving system an opprotunity to clear its buffers and act on the information received. These charaters are ignored by the protocol (they are not considered part of the information packet).
4. *Padc* — The control character used to pad between packets.
5. *Eol* — End of line character. This can actually be any character used to signify the end of a packet. Again, this is a delimiting character and not really considered part of a packet.
6. *Qctl* — The quoting character used to denote control characters.
7. *Qbin* — The quoting character used to denote a set eigth bit in binary files.
8. *Chkt* — Check character type. This field denotes what type of block error check is used. The default is a single character check sum.
9. *Rept* — Prefix for repeated characters. This character is used to indicate a compression of a series of redundant characters follows.
10. *Capas* — The capas field is a bit mask which tells what file attributes are supported.

The send–init packet is processed by the computer on the receiving end of file transfer, which turns around and sends a receive–init packet using the same format. The receive–init packet tells the values the receiving protocol uses for the fields. Where possible, it matches the values of the sending computer. Where it cannot match values, it sends its own characteristics so that the sending computer can attempt to match the values. The Kermit protocol is flexible enough for a compromise to be reached. Either the send–init or the receive–init packet can be left empty, signifying that the defaults of the other end system will be accepted.

It is in the last field, the bit mask of file attributes, where kermit is ineffective. File attributes are often incompatible accross systems. Kermit allows files to be transfered with incompatible attributes. In itself, this is not a problem, because one computer may merely be a temporary layover for a file which will be passed on to another computer with file attributes matching the originating system. Problems occur because Kermit does not store the file attributes. When a file is transfered from a layover computer, the attributes of that file on the layover computer are sent, not the attributes associated with the creation of the file. A good example of this involves the transfer of binary files from the Apple Macintosh.[4] The Macintosh divides binary files into two sections, control information and data. Once this file is sent to another computer (besides a Macintosh) the control section is virtually lost. Only a sophisticated version of Kermit on another Macintosh can recreate the original binary file. But this is not standard to the Kermit protocol. The problem is one of *invertability.* Files can be transfered in one direction, but not back to the original system.

Aside from the loss of file attributes, binary files are difficult to transfer in general. The characters of a normal text file can be made sense of on unlike terminals (so long as the same character set is used). But binary files, such as executable program modules, are not simply transferred. They need to be sent as a sequence of characters while preserving the status of the eigth bit. Systems whose byte size is not 8 bits must make provisions for *image mode* transfer; files are translated into long strings of bytes (using the appropriate number of bits) with no record terminators or delimiting characters. The exact image of the file is transfered.

All 8 bits in a byte of a binary file are normally important. At the very least, 7 bits will be data and one bit will be parity needed by the hardware or software system. Kermit has to map binary bytes to ASCII characters for transmission. To do this, an ampersand (&) is appended to the front of a character of binary information to denote a set eigth bit. The ampersand is only used as necessary, when the eigth bit cannot be neglected or parity can not be ignored. By limiting the use of the ampersand, Kermit ensures the integrity of binary transmission while still attempting to minimize the amount of information passed.[2]

Application Layer :

Implementation of file transfer actually takes place in the application layer.[7] A user interface is built for operation of the application. All underlying layers of the protocol are brought together into a running program.

[4]Macintosh is a trademark licensed to Apple Computers, Inc.

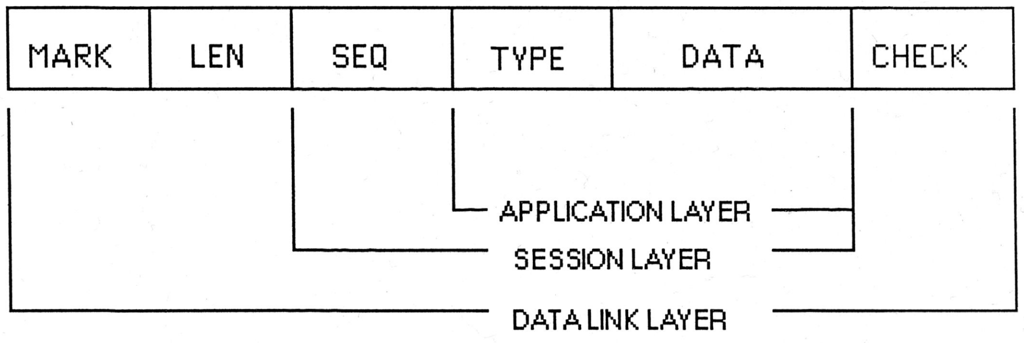


Figure 4. Entire packet layout.

Of the six fields making up the packet structure, four have already been defined. The mark beginning each packet, the packet length and the error detection fields were built into the data link layer. The sequencing character is defined in the session layer. In the application layer, the two remaining fields are defined, the type character and the data itself.[1] (Figure 4.)

The format for data is described in the presentation layer. The application layer places the data into a packet for transmission. The overall packet length, established by a compromise between the send–init and receive–init packets, determines the number of data characters. Subtracting five, for the five control character, from the overall length of the packet gives the number of data characters transferred in the packet.

The type character is the last control character to define. It specifies what type of packet is transmitted. Possible packet types are:

- *Data* — transfer of file information
- *Ack* — A positive acknowledgement
- *Nak* — A negative acknowledgement
- *Send–init* — parameters passed by the originating computer
- *Receive–init* — parameters passed by the receiving computer
- *Break* — signals the end of transmission
- *File header* — contains the name and length of the file to be transferred
- *EOF* — signals the end of a file

- *error* — signals a fatal transmission error, file transfer is cancelled
- *command* — contains a command for the receiving computer to act on

Standards for general communication were established by the International Standards Organization. Another standards organization, the American National Standards Institute (ANSI) established a series of operations which take place during actual data transfer on the application level. Five major phases are described.[3]

1. Connection Establishment Phase

In this phase physical and electronic connections are made on the two ends of the circuit. RS232 communication ports are installed and activated. Modems are plugged in and cables are attatched between the modem and computer. Telephones are dialed, if necessary, to initiate a connection between the end systems.

2. Line Establishment Phase

A line of communication needs to be established between the two end systems. Normally, the modem at the originating end, the end establishing the connection for information transfer, will *poll*, request a verification signal from, its counter part. Once the original signal is sent, line turnaround occurs. This is associated with half duplex communication. The line is given to the receiving end for a return signal. The receiving end send a negative or positive response.

A negative response can be sent for a number of reasons. The communication speeds (baud rate) of the computers may be mismatched. Another possibility is that communication parameters do not coincide. The two systems must match parity, the number of stop bits used to delimit each character of information and the number of bit representing each data character. All four parameters, baud rate, system parity, stop bits and data bits, can be adjusted using commands to the Kermit protocol.[6] A problem may arise if the modem is not capable of operating at alternative baud rates. If a negative response is received in reply to the initial poll, the communication line is disconnected. The line establishment phase is reinitiated after necessary parameters are reset.

If all the necessary parameters are met, a positive response will be received to the initial poll. The third phase can be initiated.

3. Information Transfer Phase

During the information transfer phase, communication and information transfer take place. The general protocol for information transfer is:

- Poll to host from remote
- Acknowledgement sent by host
- Message sent by remote
- Acknowledgement sent by host
- End of text sent by remote
- Poll to remote from host
- Acknowledgement sent by remote
- Message sent by host
- End of text sent by host, transaction is processed

This general scheme can be translated into a Kermit information transfer protocol. (Figure 5.) The remote computer is the computer which initiated communication and will send information to the host computer.[5]

- Remote Kermit polls host Kermit for response
- Host Kermit is activated
- Remote Kermit transfers a send–init packet to establish information transfer parameters.
- Host Kermit acknowledges with a receive–init packet
- Remote Kermit sends a file header packet to notify the host of the file name under which it will store the data on the host.
- Host Kermit acknowledges the receipt of the file header packet.
- Remote Kermit sends data packets
- Host Kermit sends a *nak* or *ack* packet in response to each data packet.
- Remote Kermit responds to *ack* packet by sending the next packet in sequence.

OR

Remote Kermit responds to *nak* packet by retransmitting the last packet sent.

[5]Information can be transferred in either direction between the two computers. The distinction between remote and host versions of Kermit is made to clarify the general description of file transfer.

- Host computer continues to respond with *ack* or *nak* packets
- Remote computer sends *EOT* (end of text) packet
- Information transfer is completed.

At any time while information is being transferred, an error packet may be transmitted signaling a fatal error. Information in a packet may be garbled beyond repair, a system error may occur or the number of attempts at resending a packet containing an error may have exceeded the allowable number of retries. When an error packet is sent, file transfer is immediately halted.

Command packets are sent to the host Kermit from the remote Kermit before and after file transfer takes place. These commands tell the host version of Kermit to perform the specified operations, such as send or receive a file.

4. Line Termination Phase

Line termination occurs when Kermit sends a break packet signifying the end of transmission. This packet is sent by issuing a disconnect command to Kermit. Comminication lines are dropped.

5. Connection Clearing Phase

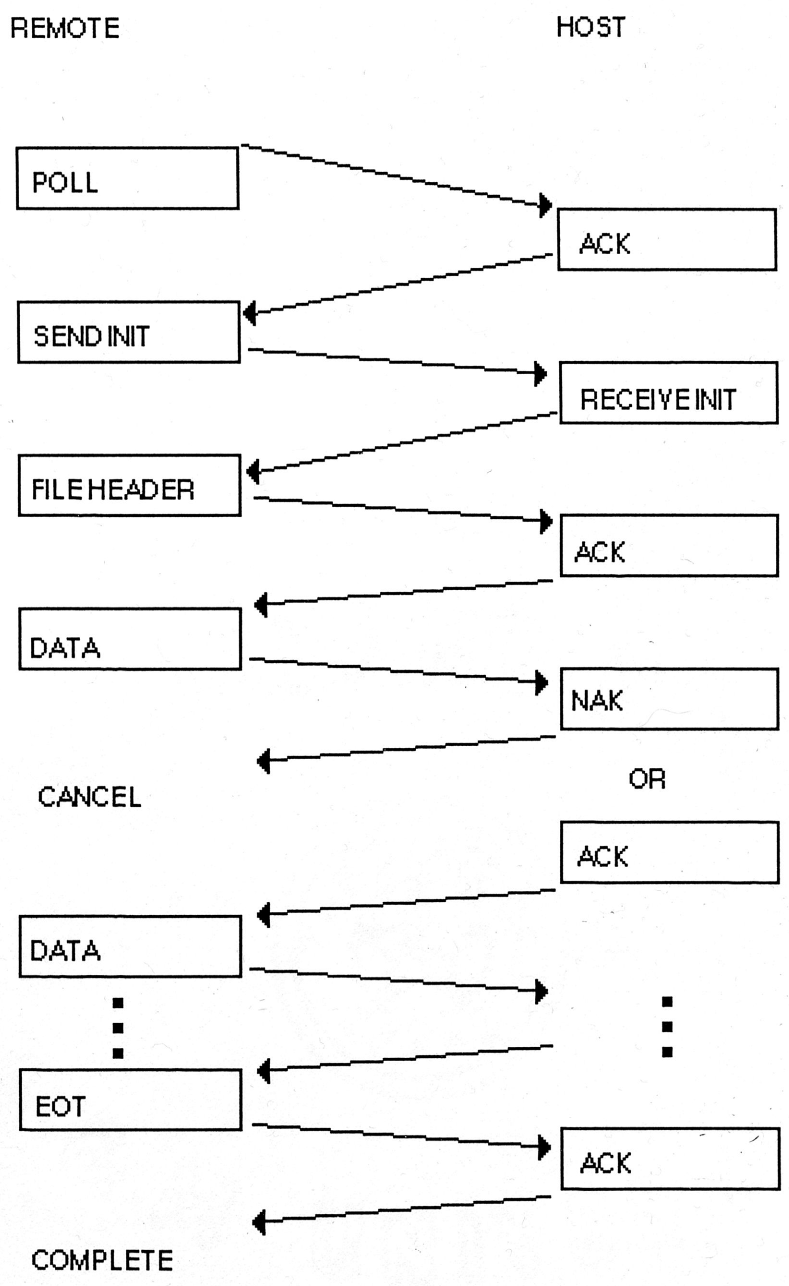
During the connection clearing phase, physical and electronic links are disconnected.

While Kermit is transferring files, it is possible for a temporary deadlock situation to occur. Deadlock is usually the result of corrupt information within a packet. The corruption occurs in such a way that the receiving Kermit is unable to acknowledge the packet. The sending Kermit sits idle waiting for a packet which has already been sent.

Temporary deadlocks are handled by allowing time outs in the protocol. If information has not been received within a certain time period, the idle protocol sends a negative acknowledgement and waits for the packet to be retransmitted. After a predetermined number of unsuccessful retries file transfer is discontinued.

Time out conditions, though they solve the deadlock problem, are not desirable. If Kermit times out, it means several seconds have been wasted when information may have been transferred. The delay caused by time outs during file transfer impair the overall efficiency of the protocol.

Several forms of packet corruption can lead to time out conditions. These corruption problems remain unsolvable by the Kermit protocol and may prove to be universally unsolvable.


REMOTE
HOST
POLL
ACK
SEND INIT
RECEIVE INIT
FILE HEADER
ACK
DATA
NAK
CANCEL
OR
ACK
DATA
EOT
ACK
COMPLETE

- Kermit looks for a marking character to signify the beginning of a packet. If this mark is garbled, Kermit will wait until timing out for a recognizable start of the packet.
- The length field may be too long. If this occurs, Kermit will be stuck waiting for characters which have not been sent.
- Losing characters will cause the same problems as a length field being too long.

Most errors are caught by the checksum, so retransmission of the packet is immediate. This reduces the delay caused by deadlock time outs. Also, by using a length field instead of a distinguishing end of packet character, Kermit reduces the number of possible time out conditions. A garbled end of packet character would always cause a time out condition.

Certain heuristics are followed to help ensure that errors produced by the Kermit protocol are minimal:

- Wait for a response before sending the next packet. This prevents buffer over runs and allows half–duplex systems to participate in file transfer. It also establishes the environment to allow time outs in case of errors not found by comparison of checksum characters.
- A *nak* for the next packet implies an *ack* for the current packet. This helps insure that lost packets do not slow transmission. The implication is that an acknowledgement of packet *n* was lost in transmission. A time out occurred because packet *n+1* was never sent.
- Redundant packets are acknowledged and discarded.
- *Nak* expected commands to make sure the command was never sent rather than lost in transmission.
- Clear input and output buffers at the beginning of transfer and after reading each packet. This aids in keeping buffers from overflowing.
- Discard redundant *ack* packets.

Conclusion

Frank da Cruz feels that the network and transport layers defined by the International Standards Organization's Open Systems Interconnection are not part of the Kermit Information Transfer Protocol. With these exceptions, Kermit conforms well to international standards for file transfer protocols. I believe that Kermit does, in fact represent the Network and Transport layers, and therefore is a usable standard for future protocols.

Kermit acts as a limited network in connection mode. It has handshaking in the form of *ack* and *nak* packets. Data packets are sent sequentially and reassembled by the receiving system. Communication takes place between two end systems without routing accross a network. Though the Kermit protocol is limited to communication only between two computer systems, all the characteristics of a network in connection mode are present. Therefore, establishing a pseudo network layer in Kermit would merely be a matter of building the routines to transfer information between the end systems on top of the data link modules rather than incorporating them into the data link. Really this does not change the format of the present Kermit protocol. All system functions associated with recent versions of Kermit are implemented as seperate modules. These modules form the data link layer. All other protocol layers can be built from these modules, including the necessary parts of the network layer.

Error detection for Kermit has also been established in the form of single character checksumming. Since checksumming operates on the packet once it has been received from the input buffer, it is not necessary to implement it as part of the data link layer. Ensuring proper data transmission in this way leaves the capability for establishment of the transport layer.

Studies have been done to test the effectiveness of the Kermit protocol. Overhead is approximately $10/p + 0.05$ where p is the packet length and single character checksumming is used for error detection. (Using the minimum packet length of 40 characters, overhead approximates $10/40 + 0.05 = 0.30$ or 30%.) This approximation varies slightly with the length of the communication line, the number of control characters transferred in the data and the number of time out delays occur because of transmission errors. The overhead can be reduced by varying the size of the packet to minimize data corruption and meet the needs of specific systems.

Even without the tested overhead information, the usage of the Kermit protocol in a wide variety of communications packages shows its effectiveness. Red Ryder, procom and qmodem are just a few packages which include the Kermit protocol for file transfer. The number of implementations of stand alone versions of Kermit was enumerated earlier in this paper. The only other protocol for file transfer I found in wide spread use in the public domain was Xmodem. The Kermit protocol is also in the public domain and therefore free to users. An undetected error rate calculated at approximately 1.56% and a satisfactory overhead rate of 30% using the minimum allowable packet length added to the desirability of implementing Kermit at Texas A&M University for wide spread general usage among the faculty and students. Kermit also conforms to standards of the International Standards Organization and American National Standards Institute making it a desirable study in information transfer protocols. In general, it is possible to conclude that, since its original implementation in 1981 by Frank da Cruz and Bill Catchings, Kermit has proven itself an efficient and effective higher protocol information transfer system.

Bibliography

[1] "Kermit:A File Transfer Protocol For Universities" Frank da Cruz , Bill Catchings BYTE, June, July 1984

[2] "Kermit Protocol Manual", Frank da Cruz. Columbia University Center for Computing Activities, 1984 Release

[3] U. D. Black. *Data Communication Networks and Distriuted Processing* Restor Publishing Company, 1983

[4] Dogan A. Tugal, Osman Tugal. *Data Transmission: Analysis, Design, Applications*, Mcgraw–Hill Book Company, 1982

[5] "Modem Magic", R. Lockwood. Creative Computing, 11:14–30 May 1985

[6] Generic Version of Kermit written in C. Frank da Cruz, Bill Catchings, Jeff Damens Columbia University Center for Computing Activities, 1985 Release

[7] "Communication Standards: OSI is not a Paper Tiger" Wendy Rauch–Hindin, Systems and Software, 4:64-80 March 1985