The ZMODEM Inter Application File Transfer Protocol

Chuck Forsberg

Omen Technology Inc

A overview of this document is available as ZMODEM.OV

(in ZMDMOV.ARC)

This file may be redistributed without restriction provided the text is

not altered.

Please distribute as widely as possible.

Omen Technology Incorporated

The High Reliability Software

17505-V Northwest Sauvie Island Road

Portland Oregon 97231

VOICE: 503-621-3406 :VOICE

Modem: 503-621-3746 Speed 1200,2400,19200(Telebit PEP)

Compuserve:70007,2304 GEnie:CAF

UUCP: ...!tektronix!reed!omen!caf

Chapter 0 Rev Oct-14-88 Typeset 10-14-88 1

Chapter 0 ZMODEM Protocol 2

1. INTENDED AUDIENCE

This document is intended for telecommunications managers, systems

programmers, and others who choose and implement asynchronous file

transfer protocols over dial-up networks and related environments.

2. WHY DEVELOP ZMODEM?

Since its development half a decade ago, the Ward Christensen MODEM

protocol has enabled a wide variety of computer systems to interchange

data. There is hardly a communications program that doesn't at least

claim to support this protocol, now called XMODEM.

Advances in computing, modems and networking have spread the XMODEM

protocol far beyond the micro to micro environment for which it was

designed. These application have exposed some weaknesses:

+ The awkward user interface is suitable for computer hobbyists.

Multiple commands must be keyboarded to transfer each file.

+ Since commands must be given to both programs, simple menu selections

are not possible.

+ The short block length causes throughput to suffer when used with

timesharing systems, packet switched networks, satellite circuits,

and buffered (error correcting) modems.

+ The 8 bit checksum and unprotected supervison allow undetected errors

and disrupted file transfers.

+ Only one file can be sent per command. The file name has to be given

twice, first to the sending program and then again to the receiving

program.

+ The transmitted file accumulates as many as 127 bytes of garbage.

+ The modification date and other file attributes are lost.

+ XMODEM requires complete 8 bit transparency, all 256 codes. XMODEM

will not operate over some networks that use ASCII flow control or

escape codes. Setting network transparency disables important

control functions for the duration of the call.

A number of other protocols have been developed over the years, but none

have proven satisfactory.

+ Lack of public domain documentation and example programs have kept

proprietary protocols such as Relay, Blast, DART, and others tightly

bound to the fortunes of their suppliers. These protocols have not

benefited from public scrutiny of their design features.

Chapter 2 Rev Oct-14-88 Typeset 10-14-88 2

Chapter 2 ZMODEM Protocol 3

+ Link level protocols such as X.25, X.PC, and MNP do not manage

application to application file transfers.

+ Link Level protocols do not eliminate end-to-end errors. Interfaces

between error-free networks are not necessarily error-free.

Sometimes, error-free networks aren't.

+ The Kermit protocol was developed to allow file transfers in

environments hostile to XMODEM. The performance compromises

necessary to accommodate traditional mainframe environments limit

Kermit's efficiency. Even with completely transparent channels,

Kermit control character quoting limits the efficiency of binary file

transfers to about 75 per cent.[1]

A number of submodes are used in various Kermit programs, including

different methods of transferring binary files. Two Kermit programs

will mysteriously fail to operate with each other if the user has not

correctly specified these submodes.

Kermit Sliding Windows ("SuperKermit") improves throughput over

networks at the cost of increased complexity. SuperKermit requires

full duplex communications and the ability to check for the presence

of characters in the input queue, precluding its implementation on

some operating systems.

SuperKermit state transitions are encoded in a special language

"wart" which requires a C compiler.

SuperKermit sends an ACK packet for each data packet of 96 bytes

(fewer if control characters are present). This reduces throughput

on high speed modems, from 1350 to 177 characters per second in one

test.

A number of extensions to the XMODEM protocol have been made to improve

performance and (in some cases) the user interface. They provide useful

improvements in some applications but not in others. XMODEM's unprotected

control messages compromise their reliability. Complex proprietary

techniques such as Cybernetic Data Recovery(TM)[2] improve reliability,

but are not universally available. Some of the XMODEM mutant protocols

have significant design flaws of their own.

+ XMODEM-k uses 1024 byte blocks to reduce the overhead from transmission

delays by 87 per cent compared to XMODEM, but network delays still

\_\_\_\_\_\_\_\_\_\_

1. Some Kermit programs support run length encoding.

2. Unique to DSZ, ZCOMM, Professional-YAM and PowerCom

Chapter 2 Rev Oct-14-88 Typeset 10-14-88 3

Chapter 2 ZMODEM Protocol 4

degrade performance. Some networks cannot transmit 1024 byte packets

without flow control, which is difficult to apply without impairing the

perfect transparency required by XMODEM. XMODEM-k adds garbage to

received files.

+ YMODEM sends the file name, file length, and creation date at the

beginning of each file, and allows optional 1024 byte blocks for

improved throughput. The handling of files that are not a multiple of

1024 or 128 bytes is awkward, especially if the file length is not

known in advance, or changes during transmission. The large number of

non conforming and substandard programs claiming to support YMODEM

further complicates its use.

+ YMODEM-g provides efficient batch file transfers, preserving exact file

length and file modification date. YMODEM-g is a modification to

YMODEM wherein ACKs for data blocks are not used. YMODEM-g is

essentially insensitive to network delays. Because it does not support

error recovery, YMODEM-g must be used hard wired or with a reliable

link level protocol. Successful application at high speed requires

cafeful attention to transparent flow control. When YMODEM-g detects a

CRC error, data transfers are aborted. YMODEM-g is easy to implement

because it closely resembles standard YMODEM-1k.

+ WXMODEM, SEAlink, and MEGAlink have applied a subset of ZMODEM's

techniques to "Classic XMODEM" to improve upon their suppliers'

previous offerings. They provide good performance under ideal

conditions.

Another XMODEM "extension" is protocol cheating, such as Omen Technology's

OverThruster(TM) and OverThruster II(TM). These improve XMODEM throughput

under some conditions by compromising error recovery.

The ZMODEM Protocol corrects the weaknesses described above while

maintaining as much of XMODEM/CRC's simplicity and prior art as possible.

3. ZMODEM Protocol Design Criteria

The design of a file transfer protocol is an engineering compromise

between conflicting requirements:

3.1 Ease of Use

+ ZMODEM allows either program to initiate file transfers.

+ The sender can pass commands and/or modifiers to the receiving program.

+ File names need be entered only once.

Chapter 3 Rev Oct-14-88 Typeset 10-14-88 4

Chapter 3 ZMODEM Protocol 5

+ Menu selections are supported.

+ Wild Card names may be used with batch transfers.

+ Minimum keystrokes required to initiate transfers.

+ ZRQINIT frame sent by sending program can trigger automatic downloads.

+ ZMODEM can optionally step down to YMODEM if the other end does not

support ZMODEM.[1]

3.2 Throughput

All file transfer protocols make tradeoffs between throughput,

reliability, universality, and complexity according to the technology and

knowledge base available to their designers.

In the design of ZMODEM, three applications deserve special attention.

+ Network applications with significant delays (relative to character

transmission time) and low error rate

+ Timesharing and buffered modem applications with significant delays

and throughput that is quickly degraded by reverse channel traffic.

ZMODEM's economy of reverse channel bandwidth allows modems that

dynamically partition bandwidth between the two directions to operate

at optimal speeds. Special ZMODEM features allow simple, efficient

implementation on a wide variety of timesharing hosts.

+ Traditional direct modem to modem communications with high error rate

Unlike Sliding Windows Kermit, ZMODEM is not optimized for optimum

throughput when error rate and delays are both high. This tradeoff

markedly reduces code complexity and memory requirements. ZMODEM

generally provides faster error recovery than network compatible XMODEM

implementations.

In the absence of network delays, rapid error recovery is possible, much

faster than MEGAlink and network compatible versions of YMODEM and XMODEM.

File transfers begin immediately regardless of which program is started

first, without the 10 second delay associated with XMODEM.

\_\_\_\_\_\_\_\_\_\_

1. Provided the transmission medium accommodates X/YMODEM.

Chapter 3 Rev Oct-14-88 Typeset 10-14-88 5

Chapter 3 ZMODEM Protocol 6

3.3 Integrity and Robustness

Once a ZMODEM session is begun, all transactions are protected with 16 or

32 bit CRC.[2] Complex proprietary techniques such as Omen Technology's

Cybernetic Data Recovery(TM)[3] are not needed for reliable transfers.

This complete protection of data and supervisory information accounts for

most of ZMODEM's high reliability compared to XMODEM derived protocols

including WXMODEM, SEAlink, MEGAlink, etc.

An optional 32-bit CRC used as the frame check sequence in ADCCP (ANSI

X3.66, also known as FIPS PUB 71 and FED-STD-1003, the U.S. versions of

CCITT's X.25) is available. The 32 bit CRC reduces undetected errors by

at least five orders of magnitude when properly applied (-1 preset,

inversion).

A security challenge mechanism guards against "Trojan Horse" messages

written to mimic legitimate command or file downloads.

3.4 Ease of Implementation

ZMODEM accommodates a wide variety of systems:

+ Microcomputers that cannot overlap disk and serial i/o

+ Microcomputers that cannot overlap serial send and receive

+ Computers and/or networks requiring XON/XOFF flow control

+ Computers that cannot check the serial input queue for the presence of

data without having to wait for the data to arrive.

Although ZMODEM provides "hooks" for multiple "threads", ZMODEM is not

intended to replace link level protocols such as X.25.

ZMODEM accommodates network and timesharing system delays by continuously

transmitting data unless the receiver interrupts the sender to request

retransmission of garbled data. ZMODEM in effect uses the entire file as

a window.[4] Using the entire file as a window simplifies buffer

management, avoiding the window overrun failure modes that affect

MEGAlink, SuperKermit, and others.

\_\_\_\_\_\_\_\_\_\_

2. Except for the CAN-CAN-CAN-CAN-CAN abort sequence which requires five

successive CAN characters.

3. Unique to Professional-YAM, ZCOMM, and PowerCom

4. Streaming strategies are discussed in coming chapters.

Chapter 3 Rev Oct-14-88 Typeset 10-14-88 6

Chapter 3 ZMODEM Protocol 7

ZMODEM provides a general purpose application to application file transfer

protocol which may be used directly or with with reliable link level

protocols such as X.25, MNP, Fastlink, etc. When used with X.25, MNP,

Fastlink, etc., ZMODEM detects and corrects errors in the interfaces

between error controlled media and the remainder of the communications

link.

ZMODEM was developed for the public domain under a Telenet contract. The

ZMODEM protocol descriptions and the Unix rz/sz program source code are

public domain. No licensing, trademark, or copyright restrictions apply

to the use of the protocol, the Unix rz/sz source code and the ZMODEM

name.

4. EVOLUTION OF ZMODEM

In early 1986, Telenet funded a project to develop an improved public

domain application to application file transfer protocol. This protocol

would alleviate the throughput problems network customers were

experiencing with XMODEM and Kermit file transfers.

In the beginning, we thought a few modifications to XMODEM would allow

high performance over packet switched networks while preserving XMODEM's

simplicity.

The initial concept would add a block number to the ACK and NAK characters

used by XMODEM. The resultant protocol would allow the sender to send

more than one block before waiting for a response.

But how to add the block number to XMODEM's ACK and NAK? WXMODEM,

SEAlink, MEGAlink and some other protocols add binary byte(s) to indicate

the block number.

Pure binary was unsuitable for ZMODEM because binary code combinations

won't pass bidirectionally through some modems, networks and operating

systems. Other operating systems may not be able to recognize something

coming back[1] unless a break signal or a system dependent code or

sequence is present. By the time all this and other problems with the

simple ACK/NAK sequences mentioned above were corrected, XMODEM's simple

ACK and NACK characters had evolved into a real packet. The Frog was

riveting.

Managing the window[2] was another problem. Experience gained in

\_\_\_\_\_\_\_\_\_\_

1. Without stopping for a response

2. The WINDOW is the data in transit between sender and receiver.

Chapter 4 Rev Oct-14-88 Typeset 10-14-88 7

Chapter 4 ZMODEM Protocol 8

debugging The Source's SuperKermit protocol indicated a window size of

about 1000 characters was needed at 1200 bps. High speed modems require a

window of 20000 or more characters for full throughput. Much of the

SuperKermit's inefficiency, complexity and debugging time centered around

its ring buffering and window management. There had to be a better way to

get the job done.

A sore point with XMODEM and its progeny is error recovery. More to the

point, how can the receiver determine whether the sender has responded, or

is ready to respond, to a retransmission request? XMODEM attacks the

problem by throwing away characters until a certain period of silence.

Too short a time allows a spurious pause in output (network or timesharing

congestion) to masquerade as error recovery. Too long a timeout

devastates throughput, and allows a noisy line to lock up the protocol.

SuperKermit solves the problem with a distinct start of packet character

(SOH). WXMODEM and ZMODEM use unique character sequences to delineate the

start of frames. SEAlink and MEGAlink do not address this problem.

A further error recovery problem arises in streaming protocols. How does

the receiver know when (or if) the sender has recognized its error signal?

Is the next packet the correct response to the error signal? Is it

something left over "in the queue"? Or is this new subpacket one of many

that will have to be discarded because the sender did not receive the

error signal? How long should this continue before sending another error

signal? How can the protocol prevent this from degenerating into an

argument about mixed signals?

SuperKermit uses selective retransmission, so it can accept any good

packet it receives. Each time the SuperKermit receiver gets a data

packet, it must decide which outstanding packet (if any) it "wants most"

to receive, and asks for that one. In practice, complex software "hacks"

are needed to attain acceptable robustness.[3]

For ZMODEM, we decided to forgo the complexity of SuperKermit's packet

assembly scheme and its associated buffer management logic and memory

requirements.

Another sore point with XMODEM and WXMODEM is the garbage added to files.

This was acceptable with the old CP/M files which had no exact length, but

not with newer systems such as PC-DOS and Unix. YMODEM uses file length

information transmitted in the header block to trim the output file, but

this causes data loss when transferring files that grow during a transfer.

\_\_\_\_\_\_\_\_\_\_

3. For example, when SuperKermit encounters certain errors, the wndesr

function is called to determine the next block to request. A burst of

errors generates several wasteful requests to retransmit the same

block.

Chapter 4 Rev Oct-14-88 Typeset 10-14-88 8

Chapter 4 ZMODEM Protocol 9

In some cases, the file length may be unknown, as when data is obtained

from a process. Variable length data subpackets solve both of these

problems.

Since some characters had to be escaped anyway, there wasn't any point

wasting bytes to fill out a fixed packet length or to specify a variable

packet length. In ZMODEM, the length of data subpackets is denoted by

ending each subpacket with an escape sequence similar to BISYNC and HDLC.

The end result is a ZMOEM header containing a "frame type", four bytes of

supervisory information, and its own CRC. Data frames consist of a header

followed by 1 or more data subpackets. In the absence of transmission

errors, an entire file can be sent in one data frame.

Since the sending system may be sensitive to numerous control characters

or strip parity in the reverse data path, all of the headers sent by the

receiver are sent in hex. A common lower level routine receives all

headers, allowing the main program logic to deal with headers and data

subpackets as objects.

With equivalent binary (efficient) and hex (application friendly) frames,

the sending program can send an "invitation to receive" sequence to

activate the receiver without crashing the remote application with

unexpected control characters.

Going "back to scratch" in the protocol design presents an opportunity to

steal good ideas from many sources and to add a few new ones.

From Kermit and UUCP comes the concept of an initial dialog to exchange

system parameters.

ZMODEM generalizes Compuserve B Protocol's host controlled transfers to

single command AutoDownload and command downloading. A Security Challenge

discourages password hackers and Trojan Horse authors from abusing

ZMODEM's power.

We were also keen to the pain and $uffering of legions of

telecommunicators whose file transfers have been ruined by communications

and timesharing faults. ZMODEM's file transfer recovery and advanced file

management are dedicated to these kindred comrades.

After ZMODEM had been operational a short time, Earl Hall pointed out the

obvious: ZMODEM's user friendly AutoDownload was almost useless if the

user must assign transfer options to each of the sending and receiving

programs. Now, transfer options may be specified to/by the sending

program, which passes them to the receiving program in the ZFILE header.

Chapter 5 Rev Oct-14-88 Typeset 10-14-88 9

Chapter 5 ZMODEM Protocol 10

5. ROSETTA STONE

Here are some definitions which reflect current vernacular in the computer

media. The attempt here is identify the file transfer protocol rather

than specific programs.

FRAME A ZMODEM frame consists of a header and 0 or more data subpackets.

XMODEM refers to the original 1977 file transfer etiquette introduced by

Ward Christensen's MODEM2 program. It's also called the MODEM or

MODEM2 protocol. Some who are unaware of MODEM7's unusual batch

file mode call it MODEM7. Other aliases include "CP/M Users's

Group" and "TERM II FTP 3". This protocol is supported by most

communications programs because it is easy to implement.

XMODEM/CRC replaces XMODEM's 1 byte checksum with a two byte Cyclical

Redundancy Check (CRC-16), improving error detection.

XMODEM-1k Refers to XMODEM-CRC with optional 1024 byte blocks.

YMODEM refers to the XMODEM/CRC protocol with batch transmission and

optional 1024 byte blocks as described in YMODEM.DOC.[1]

6. ZMODEM REQUIREMENTS

ZMODEM requires an 8 bit transfer medium.[1] ZMODEM escapes network

control characters to allow operation with packet switched networks. In

general, ZMODEM operates over any path that supports XMODEM, and over many

that don't.

To support full streaming,[2] the transmission path should either assert

flow control or pass full speed transmission without loss of data.

Otherwise the ZMODEM sender must manage the window size.

6.1 File Contents

6.1.1 Binary Files

ZMODEM places no constraints on the information content of binary files,

except that the number of bits in the file must be a multiple of 8.

\_\_\_\_\_\_\_\_\_\_

1. Available on TeleGodzilla as part of YZMODEM.ZOO

1. The ZMODEM design allows encoded packets for less transparent media.

2. With XOFF and XON, or out of band flow control such as X.25 or CTS

Chapter 6 Rev Oct-14-88 Typeset 10-14-88 10

Chapter 6 ZMODEM Protocol 11

6.1.2 Text Files

Since ZMODEM is used to transfer files between different types of computer

systems, text files must meet minimum requirements if they are to be

readable on a wide variety of systems and environments.

Text lines consist of printing ASCII characters, spaces, tabs, and

backspaces.

6.1.2.1 ASCII End of Line

The ASCII code definition allows text lines terminated by a CR/LF (015,

012) sequence, or by a NL (012) character. Lines logically terminated by

a lone CR (013) are not ASCII text.

A CR (013) without a linefeed implies overprinting, and is not acceptable

as a logical line separator. Overprinted lines should print all important

characters in the last pass to allow CRT displays to display meaningful

text. Overstruck characters may be generated by backspacing or by

overprinting the line with CR (015) not followed by LF.

Overstruck characters generated with backspaces should be sent with the

most important character last to accommodate CRT displays that cannot

overstrike. The sending program may use the ZCNL bit to force the

receiving program to convert the received end of line to its local end of

line convention.[3]

\_\_\_\_\_\_\_\_\_\_

3. Files that have been translated in such a way as to modify their

length cannot be updated with the ZCRECOV Conversion Option.

Chapter 6 Rev Oct-14-88 Typeset 10-14-88 11

Chapter 6 ZMODEM Protocol 12

7. ZMODEM BASICS

7.1 Packetization

ZMODEM frames differ somewhat from XMODEM blocks. XMODEM blocks are not

used for the following reasons:

+ Block numbers are limited to 256

+ No provision for variable length blocks

+ Line hits corrupt protocol signals, causing failed file transfers. In

particular, modem errors sometimes generate false block numbers, false

EOTs and false ACKs. False ACKs are the most troublesome as they cause

the sender to lose synchronization with the receiver.

State of the art programs such as Professional-YAM and ZCOMM overcome

some of these weaknesses with clever proprietary code, but a stronger

protocol is desired.

+ It is difficult to determine the beginning and ends of XMODEM blocks

when line hits cause a loss of synchronization. This precludes rapid

error recovery.

7.2 Link Escape Encoding

ZMODEM achieves data transparency by extending the 8 bit character set

(256 codes) with escape sequences based on the ZMODEM data link escape

character ZDLE.[1]

Link Escape coding permits variable length data subpackets without the

overhead of a separate byte count. It allows the beginning of frames to

be detected without special timing techniques, facilitating rapid error

recovery.

Link Escape coding does add some overhead. The worst case, a file

consisting entirely of escaped characters, would incur a 50% overhead.

The ZDLE character is special. ZDLE represents a control sequence of some

sort. If a ZDLE character appears in binary data, it is prefixed with

ZDLE, then sent as ZDLEE.

The value for ZDLE is octal 030 (ASCII CAN). This particular value was

chosen to allow a string of 5 consecutive CAN characters to abort a ZMODEM

\_\_\_\_\_\_\_\_\_\_

1. This and other constants are defined in the zmodem.h include file.

Please note that constants with a leading 0 are octal constants in C.

Chapter 7 Rev Oct-14-88 Typeset 10-14-88 12

Chapter 7 ZMODEM Protocol 13

session, compatible with YMODEM session abort.

Since CAN is not used in normal terminal operations, interactive

applications and communications programs can monitor the data flow for

ZDLE. The following characters can be scanned to detect the ZRQINIT

header, the invitation to automatically download commands or files.

Receipt of five successive CAN characters will abort a ZMODEM session.

Eight CAN characters are sent.

The receiving program decodes any sequence of ZDLE followed by a byte with

bit 6 set and bit 5 reset (upper case letter, either parity) to the

equivalent control character by inverting bit 6. This allows the

transmitter to escape any control character that cannot be sent by the

communications medium. In addition, the receiver recognizes escapes for

0177 and 0377 should these characters need to be escaped.

ZMODEM software escapes ZDLE, 020, 0220, 021, 0221, 023, and 0223. If

preceded by 0100 or 0300 (@), 015 and 0215 are also escaped to protect the

Telenet command escape CR-@-CR. The receiver ignores 021, 0221, 023, and

0223 characters in the data stream.

The ZMODEM routines in zm.c accept an option to escape all control

characters, to allow operation with less transparent networks. This

option can be given to either the sending or receiving program.

7.3 Header

All ZMODEM frames begin with a header which may be sent in binary or HEX

form. ZMODEM uses a single routine to recognize binary and hex headers.

Either form of the header contains the same raw information:

+ A type byte[2] [3]

+ Four bytes of data indicating flags and/or numeric quantities depending

on the frame type

\_\_\_\_\_\_\_\_\_\_

2. The frame types are cardinal numbers beginning with 0 to minimize

state transition table memory requirements.

3. Future extensions to ZMODEM may use the high order bits of the type

byte to indicate thread selection.

Chapter 7 Rev Oct-14-88 Typeset 10-14-88 13

Chapter 7 ZMODEM Protocol 14

Figure 1. Order of Bytes in Header

TYPE: frame type

F0: Flags least significant byte

P0: file Position least significant

P3: file Position most significant

TYPE F3 F2 F1 F0

-------------------

TYPE P0 P1 P2 P3

7.3.1 16 Bit CRC Binary Header

A binary header is sent by the sending program to the receiving program.

ZDLE encoding accommodates XON/XOFF flow control.

A binary header begins with the sequence ZPAD, ZDLE, ZBIN.

The frame type byte is ZDLE encoded.

The four position/flags bytes are ZDLE encoded.

A two byte CRC of the frame type and position/flag bytes is ZDLE encoded.

0 or more binary data subpackets with 16 bit CRC will follow depending on

the frame type.

The function zsbhdr transmits a binary header. The function zgethdr

receives a binary or hex header.

Figure 2. 16 Bit CRC Binary Header

\* ZDLE A TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2

7.3.2 32 Bit CRC Binary Header

A "32 bit CRC" Binary header is similar to a Binary Header, except the

ZBIN (A) character is replaced by a ZBIN32 (C) character, and four

characters of CRC are sent. 0 or more binary data subpackets with 32 bit

CRC will follow depending on the frame type.

The common variable Txfcs32 may be set TRUE for 32 bit CRC iff the

receiver indicates the capability with the CANFC32 bit. The zgethdr,

zsdata and zrdata functions automatically adjust to the type of Frame

Check Sequence being used.

Figure 3. 32 Bit CRC Binary Header

\* ZDLE C TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2 CRC-3 CRC-4

7.3.3 HEX Header

The receiver sends responses in hex headers. The sender also uses hex

headers when they are not followed by binary data subpackets.

Chapter 7 Rev Oct-14-88 Typeset 10-14-88 14

Chapter 7 ZMODEM Protocol 15

Hex encoding protects the reverse channel from random control characters.

The hex header receiving routine ignores parity.

Use of Kermit style encoding for control and paritied characters was

considered and rejected because of increased possibility of interacting

with some timesharing systems' line edit functions. Use of HEX headers

from the receiving program allows control characters to be used to

interrupt the sender when errors are detected. A HEX header may be used

in place of a binary header wherever convenient. If a data packet follows

a HEX header, it is protected with CRC-16.

A hex header begins with the sequence ZPAD, ZPAD, ZDLE, ZHEX. The zgethdr

routine synchronizes with the ZPAD-ZDLE sequence. The extra ZPAD

character allows the sending program to detect an asynchronous header

(indicating an error condition) and then call zgethdr to receive the

header.

The type byte, the four position/flag bytes, and the 16 bit CRC thereof

are sent in hex using the character set 01234567890abcdef. Upper case hex

digits are not allowed; they false trigger XMODEM and YMODEM programs.

Since this form of hex encoding detects many patterns of errors,

especially missing characters, a hex header with 32 bit CRC has not been

defined.

A carriage return and line feed are sent with HEX headers. The receive

routine expects to see at least one of these characters, two if the first

is CR. The CR/LF aids debugging from printouts, and helps overcome

certain operating system related problems.

An XON character is appended to all HEX packets except ZACK and ZFIN. The

XON releases the sender from spurious XOFF flow control characters

generated by line noise, a common occurrence. XON is not sent after ZACK

headers to protect flow control in streaming situations. XON is not sent

after a ZFIN header to allow clean session cleanup.

0 or more data subpackets will follow depending on the frame type.

The function zshhdr sends a hex header.

Figure 4. HEX Header

\* \* ZDLE B TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2 CR LF XON

(TYPE, F3...F0, CRC-1, and CRC-2 are each sent as two hex digits.)

Chapter 7 Rev Oct-14-88 Typeset 10-14-88 15

Chapter 7 ZMODEM Protocol 16

7.4 Binary Data Subpackets

Binary data subpackets immediately follow the associated binary header

packet. A binary data packet contains 0 to 1024 bytes of data.

Recommended length values are 256 bytes below 2400 bps, 512 at 2400 bps,

and 1024 above 4800 bps or when the data link is known to be relatively

error free.[4]

No padding is used with binary data subpackets. The data bytes are ZDLE

encoded and transmitted. A ZDLE and frameend are then sent, followed by

two or four ZDLE encoded CRC bytes. The CRC accumulates the data bytes

and frameend.

The function zsdata sends a data subpacket. The function zrdata receives

a data subpacket.

7.5 ASCII Encoded Data Subpacket

The format of ASCII Encoded data subpackets is not currently specified.

These could be used for server commands, or main transfers in 7 bit

environments.

8. PROTOCOL TRANSACTION OVERVIEW

As with the XMODEM recommendation, ZMODEM timing is receiver driven. The

transmitter should not time out at all, except to abort the program if no

headers are received for an extended period of time, say one minute.[1]

8.1 Session Startup

To start a ZMODEM file transfer session, the sending program is called

with the names of the desired file(s) and option(s).

The sending program may send the string "rz\r" to invoke the receiving

program from a possible command mode. The "rz" followed by carriage

return activates a ZMODEM receive program or command if it were not

already active.

The sender may then display a message intended for human consumption, such

\_\_\_\_\_\_\_\_\_\_

4. Strategies for adjusting the subpacket length for optimal results

based on real time error rates are still evolving. Shorter subpackets

speed error detection but increase protocol overhead slightly.

1. Special considerations apply when sending commands.

Chapter 8 Rev Oct-14-88 Typeset 10-14-88 16

Chapter 8 ZMODEM Protocol 17

as a list of the files requested, etc.

Then the sender may send a ZRQINIT header. The ZRQINIT header causes a

previously started receive program to send its ZRINIT header without

delay.

In an interactive or conversational mode, the receiving application may

monitor the data stream for ZDLE. The following characters may be scanned

for B00 indicating a ZRQINIT header, a command to download a command or

data.

The sending program awaits a command from the receiving program to start

file transfers. If a "C", "G", or NAK is received, an XMODEM or YMODEM

file transfer is indicated, and file transfer(s) use the YMODEM protocol.

Note: With ZMODEM and YMODEM, the sending program provides the file name,

but not with XMODEM.

In case of garbled data, the sending program can repeat the invitation to

receive a number of times until a session starts.

When the ZMODEM receive program starts, it immediately sends a ZRINIT

header to initiate ZMODEM file transfers, or a ZCHALLENGE header to verify

the sending program. The receive program resends its header at response

time (default 10 second) intervals for a suitable period of time (40

seconds total) before falling back to YMODEM protocol.

If the receiving program receives a ZRQINIT header, it resends the ZRINIT

header. If the sending program receives the ZCHALLENGE header, it places

the data in ZP0...ZP3 in an answering ZACK header.

If the receiving program receives a ZRINIT header, it is an echo

indicating that the sending program is not operational.

Eventually the sending program correctly receives the ZRINIT header.

The sender may then send an optional ZSINIT frame to define the receiving

program's Attn sequence, or to specify complete control character

escaping.[2]

If the ZSINIT header specifies ESCCTL or ESC8, a HEX header is used, and

the receiver activates the specified ESC modes before reading the

following data subpacket.

The receiver sends a ZACK header in response, containing either the serial

\_\_\_\_\_\_\_\_\_\_

2. If the receiver specifies the same or higher level of escaping, the

ZSINIT frame need not be sent unless an Attn sequence is needed.

Chapter 8 Rev Oct-14-88 Typeset 10-14-88 17

Chapter 8 ZMODEM Protocol 18

number of the receiving program, or 0.

8.2 File Transmission

The sender then sends a ZFILE header with ZMODEM Conversion, Management,

and Transport options[3] followed by a ZCRCW data subpacket containing the

file name, file length, modification date, and other information identical

to that used by YMODEM Batch.

The receiver examines the file name, length, and date information provided

by the sender in the context of the specified transfer options, the

current state of its file system(s), and local security requirements. The

receiving program should insure the pathname and options are compatible

with its operating environment and local security requirements.

The receiver may respond with a ZSKIP header, which makes the sender

proceed to the next file (if any) in the batch.

The receiver has a file with the same name and length, may

respond with a ZCRC header with a byte count, which

requires the sender to perform a 32 bit CRC on the

specified number of bytes in the file and transmit the

complement of the CRC in an answering ZCRC header.[4] The

receiver uses this information to determine whether to

accept the file or skip it. This sequence may be triggered

by the ZMCRC Management Option.

A ZRPOS header from the receiver initiates transmission of the file data

starting at the offset in the file specified in the ZRPOS header.

Normally the receiver specifies the data transfer to begin begin at

offset 0 in the file.

The receiver may start the transfer further down in the

file. This allows a file transfer interrupted by a loss

or carrier or system crash to be completed on the next

connection without requiring the entire file to be

retransmitted.[5] If downloading a file from a timesharing

system that becomes sluggish, the transfer can be

interrupted and resumed later with no loss of data.

The sender sends a ZDATA binary header (with file position) followed by

\_\_\_\_\_\_\_\_\_\_

3. See below, under ZFILE header type.

4. The crc is initialized to 0xFFFFFFFF. A byte count of 0 implies the

entire file.

5. This does not apply to files that have been translated.

Chapter 8 Rev Oct-14-88 Typeset 10-14-88 18

Chapter 8 ZMODEM Protocol 19

one or more data subpackets.

The receiver compares the file position in the ZDATA header with the

number of characters successfully received to the file. If they do not

agree, a ZRPOS error response is generated to force the sender to the

right position within the file.[6]

A data subpacket terminated by ZCRCG and CRC does not elicit a response

unless an error is detected; more data subpacket(s) follow immediately.

ZCRCQ data subpackets expect a ZACK response with the

receiver's file offset if no error, otherwise a ZRPOS

response with the last good file offset. Another data

subpacket continues immediately. ZCRCQ subpackets are

not used if the receiver does not indicate FDX ability

with the CANFDX bit.

ZCRCW data subpackets expect a response before the next frame is sent.

If the receiver does not indicate overlapped I/O capability with the

CANOVIO bit, or sets a buffer size, the sender uses the ZCRCW to allow

the receiver to write its buffer before sending more data.

A zero length data frame may be used as an idle

subpacket to prevent the receiver from timing out in

case data is not immediately available to the sender.

In the absence of fatal error, the sender eventually encounters end of

file. If the end of file is encountered within a frame, the frame is

closed with a ZCRCE data subpacket which does not elicit a response

except in case of error.

The sender sends a ZEOF header with the file ending offset equal to

the number of characters in the file. The receiver compares this

number with the number of characters received. If the receiver has

received all of the file, it closes the file. If the file close was

satisfactory, the receiver responds with ZRINIT. If the receiver has

not received all the bytes of the file, the receiver ignores the ZEOF

because a new ZDATA is coming. If the receiver cannot properly close

the file, a ZFERR header is sent.

\_\_\_\_\_\_\_\_\_\_

6. If the ZMSPARS option is used, the receiver instead seeks to the

position given in the ZDATA header.

Chapter 8 Rev Oct-14-88 Typeset 10-14-88 19

Chapter 8 ZMODEM Protocol 20

After all files are processed, any further protocol

errors should not prevent the sending program from

returning with a success status.

8.3 Session Cleanup

The sender closes the session with a ZFIN header. The receiver

acknowledges this with its own ZFIN header.

When the sender receives the acknowledging header, it sends two

characters, "OO" (Over and Out) and exits to the operating system or

application that invoked it. The receiver waits briefly for the "O"

characters, then exits whether they were received or not.

8.4 Session Abort Sequence

If the receiver is receiving data in streaming mode, the Attn

sequence is executed to interrupt data transmission before the Cancel

sequence is sent. The Cancel sequence consists of eight CAN

characters and ten backspace characters. ZMODEM only requires five

Cancel characters, the other three are "insurance".

The trailing backspace characters attempt to erase the effects of the

CAN characters if they are received by a command interpreter.

static char canistr[] = {

24,24,24,24,24,24,24,24,8,8,8,8,8,8,8,8,8,8,0

};

Chapter 8 Rev Oct-14-88 Typeset 10-14-88 20

Chapter 8 ZMODEM Protocol 21

9. STREAMING TECHNIQUES / ERROR RECOVERY

It is a fact of life that no single method of streaming is applicable

to a majority of today's computing and telecommunications

environments. ZMODEM provides several data streaming methods

selected according to the limitations of the sending environment,

receiving environment, and transmission channel(s).

9.1 Full Streaming with Sampling

If the receiver can overlap serial I/O with disk I/O, and if the

sender can sample the reverse channel for the presence of data

without having to wait, full streaming can be used with no Attn

sequence required. The sender begins data transmission with a ZDATA

header and continuous ZCRCG data subpackets. When the receiver

detects an error, it executes the Attn sequence and then sends a

ZRPOS header with the correct position within the file.

At the end of each transmitted data subpacket, the sender checks for

the presence of an error header from the receiver. To do this, the

sender samples the reverse data stream for the presence of either a

ZPAD or CAN character.[1] Flow control characters (if present) are

acted upon.

Other characters (indicating line noise) increment a counter which is

reset whenever the sender waits for a header from the receiver. If

the counter overflows, the sender sends the next data subpacket as

ZCRCW, and waits for a response.

ZPAD indicates some sort of error header from the receiver. A CAN

suggests the user is attempting to "stop the bubble machine" by

keyboarding CAN characters. If one of these characters is seen, an

empty ZCRCE data subpacket is sent. Normally, the receiver will have

sent an ZRPOS or other error header, which will force the sender to

resume transmission at a different address, or take other action. In

the unlikely event the ZPAD or CAN character was spurious, the

receiver will time out and send a ZRPOS header.[2]

Then the receiver's response header is read and acted upon.[3]

\_\_\_\_\_\_\_\_\_\_

1. The call to rdchk() in sz.c performs this function.

2. The obvious choice of ZCRCW packet, which would trigger an ZACK from

the receiver, is not used because multiple in transit frames could

result if the channel has a long propagation delay.

3. The call to getinsync() in sz.c performs this function.

Chapter 9 Rev Oct-14-88 Typeset 10-14-88 21

Chapter 9 ZMODEM Protocol 22

A ZRPOS header resets the sender's file offset to the correct

position. If possible, the sender should purge its output buffers

and/or networks of all unprocessed output data, to minimize the

amount of unwanted data the receiver must discard before receiving

data starting at the correct file offset. The next transmitted data

frame should be a ZCRCW frame followed by a wait to guarantee

complete flushing of the network's memory.

If the receiver gets a ZACK header with an address that disagrees

with the sender address, it is ignored, and the sender waits for

another header. A ZFIN, ZABORT, or TIMEOUT terminates the session; a

ZSKIP terminates the processing of this file.

The reverse channel is then sampled for the presence of another

header from the receiver.[4] if one is detected, the getinsync()

function is again called to read another error header. Otherwise,

transmission resumes at the (possibly reset) file offset with a ZDATA

header followed by data subpackets.

9.1.1 Window Management

When sending data through a network, some nodes of the network store

data while it is transferred to the receiver. 7000 bytes and more of

transient storage have been observed. Such a large amount of storage

causes the transmitter to "get ahead" of the reciever. This can be

fatal with MEGAlink and other protocols that depend on timely

notification of errors from the receiver. This condition is not

fatal with ZMODEM, but it does slow error recovery.

To manage the window size, the sending program uses ZCRCQ data

subpackets to trigger ZACK headers from the receiver. The returning

ZACK headers inform the sender of the receiver's progress. When the

window size (current transmitter file offset - last reported receiver

file offset) exceeds a specified value, the sender waits for a

ZACK[5] packet with a receiver file offset that reduces the window

size.

Unix sz versions beginning with May 9 1987 control the window size

with the "-w N" option, where N is the maximum window size. Pro-YAM,

ZCOMM and DSZ versions beginning with May 9 1987 control the window

size with "zmodem pwN". This is compatible with previous versions of

these programs.[6]

\_\_\_\_\_\_\_\_\_\_

4. If sampling is possible.

5. ZRPOS and other error packets are handled normally.

6. When used with modems or networks that simultaneously assert flow

Chapter 9 Rev Oct-14-88 Typeset 10-14-88 22

Chapter 9 ZMODEM Protocol 23

9.2 Full Streaming with Reverse Interrupt

The above method cannot be used if the reverse data stream cannot be

sampled without entering an I/O wait. An alternate method is to

instruct the receiver to interrupt the sending program when an error

is detected.

The receiver can interrupt the sender with a control character, break

signal, or combination thereof, as specified in the Attn sequence.

After executing the Attn sequence, the receiver sends a hex ZRPOS

header to force the sender to resend the lost data.

When the sending program responds to this interrupt, it reads a HEX

header (normally ZRPOS) from the receiver and takes the action

described in the previous section. The Unix sz.c program uses a

setjmp/longjmp call to catch the interrupt generated by the Attn

sequence. Catching the interrupt activates the getinsync() function

to read the receiver's error header and take appropriate action.

When compiled for standard SYSTEM III/V Unix, sz.c uses an Attn

sequence of Ctrl-C followed by a 1 second pause to interrupt the

sender, then give the sender (Unix) time to prepare for the

receiver's error header.

9.3 Full Streaming with Sliding Window

If none of the above methods is applicable, hope is not yet lost. If

the sender can buffer responses from the receiver, the sender can use

ZCRCQ data subpackets to get ACKs from the receiver without

interrupting the transmission of data. After a sufficient number of

ZCRCQ data subpackets have been sent, the sender can read one of the

headers that should have arrived in its receive interrupt buffer.

A problem with this method is the possibility of wasting an excessive

amount of time responding to the receiver's error header. It may be

possible to program the receiver's Attn sequence to flush the

sender's interrupt buffer before sending the ZRPOS header.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

control with XON and XOFF characters and pass XON characters that

violate flow control, the receiving program should have a revision

date of May 9 or later.

Chapter 9 Rev Oct-14-88 Typeset 10-14-88 23

Chapter 9 ZMODEM Protocol 24

9.4 Full Streaming over Error Free Channels

File transfer protocols predicated on the existence of an error free

end to end communications channel have been proposed from time to

time. Such channels have proven to be more readily available in

theory than in actuality. The frequency of undetected errors

increases when modem scramblers have more bits than the error

detecting CRC.

A ZMODEM sender assuming an error free channel with end to end flow

control can send the entire file in one frame without any checking of

the reverse stream. If this channel is completely transparent, only

ZDLE need be escaped. The resulting protocol overhead for average

long files is less than one per cent.[7]

9.5 Segmented Streaming

If the receiver cannot overlap serial and disk I/O, it uses the

ZRINIT frame to specify a buffer length which the sender will not

overflow. The sending program sends a ZCRCW data subpacket and waits

for a ZACK header before sending the next segment of the file.

If the sending program supports reverse data stream sampling or

interrupt, error recovery will be faster (on average) than a protocol

(such as YMODEM) that sends large blocks.

A sufficiently large receiving buffer allows throughput to closely

approach that of full streaming. For example, 16kb segmented

streaming adds about 3 per cent to full streaming ZMODEM file

transfer times when the round trip delay is five seconds.

10. ATTENTION SEQUENCE

The receiving program sends the Attn sequence whenever it detects an

error and needs to interrupt the sending program.

The default Attn string value is empty (no Attn sequence). The

receiving program resets Attn to the empty default before each

transfer session.

The sender specifies the Attn sequence in its optional ZSINIT frame.

The Attn string is terminated with a null.

\_\_\_\_\_\_\_\_\_\_

7. One in 256 for escaping ZDLE, about two (four if 32 bit CRC is used)

in 1024 for data subpacket CRC's

Chapter 10 Rev Oct-14-88 Typeset 10-14-88 24

Chapter 10 ZMODEM Protocol 25

Two meta-characters perform special functions:

+ \335 (octal) Send a break signal

+ \336 (octal) Pause one second

11. FRAME TYPES

The numeric values for the values shown in boldface are given in

zmodem.h. Unused bits and unused bytes in the header (ZP0...ZP3) are

set to 0.

11.1 ZRQINIT

Sent by the sending program, to trigger the receiving program to send

its ZRINIT header. This avoids the aggravating startup delay

associated with XMODEM and Kermit transfers. The sending program may

repeat the receive invitation (including ZRQINIT) if a response is

not obtained at first.

ZF0 contains ZCOMMAND if the program is attempting to send a command,

0 otherwise.

11.2 ZRINIT

Sent by the receiving program. ZF0 and ZF1 contain the bitwise or

of the receiver capability flags:

#define CANCRY 8 /\* Receiver can decrypt \*/

#define CANFDX 01 /\* Rx can send and receive true FDX \*/

#define CANOVIO 02 /\* Rx can receive data during disk I/O \*/

#define CANBRK 04 /\* Rx can send a break signal \*/

#define CANCRY 010 /\* Receiver can decrypt \*/

#define CANLZW 020 /\* Receiver can uncompress \*/

#define CANFC32 040 /\* Receiver can use 32 bit Frame Check \*/

#define ESCCTL 0100 /\* Receiver expects ctl chars to be escaped

\*/

#define ESC8 0200 /\* Receiver expects 8th bit to be escaped \*/

ZP0 and ZP1 contain the size of the receiver's buffer in bytes, or 0

if nonstop I/O is allowed.

11.3 ZSINIT

The Sender sends flags followed by a binary data subpacket terminated

with ZCRCW.

/\* Bit Masks for ZSINIT flags byte ZF0 \*/

#define TESCCTL 0100 /\* Transmitter expects ctl chars to be escaped

\*/

#define TESC8 0200 /\* Transmitter expects 8th bit to be escaped

**Chapter 11 Rev Oct-14-88 Typeset 10-14-88 25**

Chapter 11 ZMODEM Protocol 26

\*/

The data subpacket contains the null terminated Attn sequence,

maximum length 32 bytes including the terminating null.

11.4 ZACK

Acknowledgment to a ZSINIT frame, ZCHALLENGE header, ZCRCQ or ZCRCW

data subpacket. ZP0 to ZP3 contain file offset. The response to

ZCHALLENGE contains the same 32 bit number received in the ZCHALLENGE

header.

11.5 ZFILE

This frame denotes the beginning of a file transmission attempt.

ZF0, ZF1, and ZF2 may contain options. A value of 0 in each of these

bytes implies no special treatment. Options specified to the

receiver override options specified to the sender with the exception

of ZCBIN. A ZCBIN from the sender overrides any other Conversion

Option given to the receiver except ZCRESUM. A ZCBIN from the

receiver overrides any other Conversion Option sent by the sender.

11.5.1 ZF0: Conversion Option

If the receiver does not recognize the Conversion Option, an

application dependent default conversion may apply.

ZCBIN "Binary" transfer - inhibit conversion unconditionally

ZCNL Convert received end of line to local end of line

convention. The supported end of line conventions are

CR/LF (most ASCII based operating systems except Unix

and Macintosh), and NL (Unix). Either of these two end

of line conventions meet the permissible ASCII

definitions for Carriage Return and Line Feed/New Line.

Neither the ASCII code nor ZMODEM ZCNL encompass lines

separated only by carriage returns. Other processing

appropriate to ASCII text files and the local operating

system may also be applied by the receiver.[1]

ZCRECOV Recover/Resume interrupted file transfer. ZCREVOV is

also useful for updating a remote copy of a file that

grows without resending of old data. If the destination

file exists and is no longer than the source, append to

the destination file and start transfer at the offset

\_\_\_\_\_\_\_\_\_\_

1. Filtering RUBOUT, NULL, Ctrl-Z, etc.

Chapter 11 Rev Oct-14-88 Typeset 10-14-88 26

Chapter 11 ZMODEM Protocol 27

corresponding to the receiver's end of file. This

option does not apply if the source file is shorter.

Files that have been converted (e.g., ZCNL) or subject

to a single ended Transport Option cannot have their

transfers recovered.

11.5.2 ZF1: Management Option

If the receiver does not recognize the Management Option, the

file should be transferred normally.

The ZMSKNOLOC bit instructs the receiver to bypass the

current file if the receiver does not have a file with the

same name.

Five bits (defined by ZMMASK) define the following set of

mutually exclusive management options.

ZMNEWL Transfer file if destination file absent. Otherwise,

transfer file overwriting destination if the source file

is newer or longer.

ZMCRC Compare the source and destination files. Transfer if

file lengths or file polynomials differ.

ZMAPND Append source file contents to the end of the existing

destination file (if any).

ZMCLOB Replace existing destination file (if any).

ZMDIFF Transfer file if destination file absent. Otherwise,

transfer file overwriting destination if files have

different lengths or dates.

ZMPROT Protect destination file by transferring file only if

the destination file is absent.

ZMNEW Transfer file if destination file absent. Otherwise,

transfer file overwriting destination if the source file

is newer.

11.5.3 ZF2: Transport Option

If the receiver does not implement the particular transport

option, the file is copied without conversion for later

processing.

ZTLZW Lempel-Ziv compression. Transmitted data will be

identical to that produced by compress 4.0 operating on

a computer with VAX byte ordering, using 12 bit

encoding.

Chapter 11 Rev Oct-14-88 Typeset 10-14-88 27

Chapter 11 ZMODEM Protocol 28

ZTCRYPT Encryption. An initial null terminated string

identifies the key. Details to be determined.

ZTRLE Run Length encoding, Details to be determined.

A ZCRCW data subpacket follows with file name, file length,

modification date, and other information described in a later

chapter.

11.5.4 ZF3: Extended Options

The Extended Options are bit encoded.

ZTSPARS Special processing for sparse files, or sender managed

selective retransmission. Each file segment is transmitted as

a separate frame, where the frames are not necessarily

contiguous. The sender should end each segment with a ZCRCW

data subpacket and process the expected ZACK to insure no data

is lost. ZTSPARS cannot be used with ZCNL.

11.6 ZSKIP

Sent by the receiver in response to ZFILE, makes the sender skip to

the next file.

11.7 ZNAK

Indicates last header was garbled. (See also ZRPOS).

11.8 ZABORT

Sent by receiver to terminate batch file transfers when requested by

the user. Sender responds with a ZFIN sequence.[2]

11.9 ZFIN

Sent by sending program to terminate a ZMODEM session. Receiver

responds with its own ZFIN.

11.10 ZRPOS

Sent by receiver to force file transfer to resume at file offset

given in ZP0...ZP3.

\_\_\_\_\_\_\_\_\_\_

2. Or ZCOMPL in case of server mode.

Chapter 11 Rev Oct-14-88 Typeset 10-14-88 28

Chapter 11 ZMODEM Protocol 29

11.11 ZDATA

ZP0...ZP3 contain file offset. One or more data subpackets follow.

11.12 ZEOF

Sender reports End of File. ZP0...ZP3 contain the ending file

offset.

11.13 ZFERR

Error in reading or writing file, protocol equivalent to ZABORT.

11.14 ZCRC

Request (receiver) and response (sender) for file polynomial.

ZP0...ZP3 contain file polynomial.

11.15 ZCHALLENGE

Request sender to echo a random number in ZP0...ZP3 in a ZACK frame.

Sent by the receiving program to the sending program to verify that

it is connected to an operating program, and was not activated by

spurious data or a Trojan Horse message.

11.16 ZCOMPL

Request now completed.

11.17 ZCAN

This is a pseudo frame type returned by gethdr() in response to a

Session Abort sequence.

11.18 ZFREECNT

Sending program requests a ZACK frame with ZP0...ZP3 containing the

number of free bytes on the current file system. A value of 0

represents an indefinite amount of free space.

11.19 ZCOMMAND

ZCOMMAND is sent in a binary frame. ZF0 contains 0 or ZCACK1 (see

below).

A ZCRCW data subpacket follows, with the ASCII text command string

terminated with a NULL character. If the command is intended to be

executed by the operating system hosting the receiving program

(e.g., "shell escape"), it must have "!" as the first character.

Otherwise the command is meant to be executed by the application

program which receives the command.

Chapter 11 Rev Oct-14-88 Typeset 10-14-88 29

Chapter 11 ZMODEM Protocol 30

If the receiver detects an illegal or badly formed command, the

receiver immediately responds with a ZCOMPL header with an error

code in ZP0...ZP3.

If ZF0 contained ZCACK1, the receiver immediately responds with a

ZCOMPL header with 0 status.

Otherwise, the receiver responds with a ZCOMPL header when the

operation is completed. The exit status of the completed command is

stored in ZP0...ZP3. A 0 exit status implies nominal completion of

the command.

If the command causes a file to be transmitted, the command sender

will see a ZRQINIT frame from the other computer attempting to send

data.

The sender examines ZF0 of the received ZRQINIT header to verify it

is not an echo of its own ZRQINIT header. It is illegal for the

sending program to command the receiving program to send a command.

If the receiver program does not implement command downloading, it

may display the command to the standard error output, then return a

ZCOMPL header.

12. SESSION TRANSACTION EXAMPLES

12.1 A simple file transfer

A simple transaction, one file, no errors, no CHALLENGE, overlapped

I/O:

Sender Receiver

"rz\r"

ZRQINIT(0)

ZRINIT

ZFILE

ZRPOS

ZDATA data ...

ZEOF

ZRINIT

ZFIN

ZFIN

OO

Chapter 12 Rev Oct-14-88 Typeset 10-14-88 30

Chapter 12 ZMODEM Protocol 31

12.2 Challenge and Command Download

Sender Receiver

"rz\r"

ZRQINIT(ZCOMMAND)

ZCHALLENGE(random-number)

ZACK(same-number)

ZRINIT

ZCOMMAND, ZDATA

(Performs Command)

ZCOMPL

ZFIN

ZFIN

OO

13. ZFILE FRAME FILE INFORMATION

ZMODEM sends the same file information with the ZFILE frame data

that YMODEM Batch sends in its block 0.

N.B.: The pathname (file name) field is mandatory.

Pathname The pathname (conventionally, the file name) is sent as a

null terminated ASCII string. This is the filename format used

by the handle oriented MSDOS(TM) functions and C library fopen

functions. An assembly language example follows:

DB 'foo.bar',0

No spaces are included in the pathname. Normally only the file

name stem (no directory prefix) is transmitted unless the

sender has selected YAM's f option to send the full absolute or

relative pathname. The source drive designator (A:, B:, etc.)

usually is not sent.

Filename Considerations

+ File names should be translated to lower case unless the

sending system supports upper/lower case file names. This

is a convenience for users of systems (such as Unix) which

store filenames in upper and lower case.

+ The receiver should accommodate file names in lower and

upper case.

+ When transmitting files between different operating

systems, file names must be acceptable to both the sender

and receiving operating systems. If not, transformations

should be applied to make the file names acceptable. If

the transformations are unsuccessful, a new file name may

Chapter 13 Rev Oct-14-88 Typeset 10-14-88 31

Chapter 13 ZMODEM Protocol 32

be invented be the receiving program.

If directories are included, they are delimited by /; i.e.,

"subdir/foo" is acceptable, "subdir\foo" is not.

Length The file length and each of the succeeding fields are

optional.[1] The length field is stored as a decimal string

counting the number of data bytes in the file.

The ZMODEM receiver uses the file length as an estimate only.

It may be used to display an estimate of the transmission time,

and may be compared with the amount of free disk space. The

actual length of the received file is determined by the data

transfer. A file may grow after transmission commences, and

all the data will be sent.

Modification Date A single space separates the modification date

from the file length.

The mod date is optional, and the filename and length may be

sent without requiring the mod date to be sent.

The mod date is sent as an octal number giving the time the

contents of the file were last changed measured in seconds from

Jan 1 1970 Universal Coordinated Time (GMT). A date of 0

implies the modification date is unknown and should be left as

the date the file is received.

This standard format was chosen to eliminate ambiguities

arising from transfers between different time zones.

File Mode A single space separates the file mode from the

modification date. The file mode is stored as an octal string.

Unless the file originated from a Unix system, the file mode is

set to 0. rz(1) checks the file mode for the 0x8000 bit which

indicates a Unix type regular file. Files with the 0x8000 bit

set are assumed to have been sent from another Unix (or

similar) system which uses the same file conventions. Such

files are not translated in any way.

Serial Number A single space separates the serial number from the

file mode. The serial number of the transmitting program is

stored as an octal string. Programs which do not have a serial

\_\_\_\_\_\_\_\_\_\_

1. Fields may not be skipped.

Chapter 13 Rev Oct-14-88 Typeset 10-14-88 32

Chapter 13 ZMODEM Protocol 33

number should omit this field, or set it to 0. The receiver's

use of this field is optional.

Number of Files Remaining Iff the number of files remaining is sent,

a single space separates this field from the previous field.

This field is coded as a decimal number, and includes the

current file. This field is an estimate, and incorrect values

must not be allowed to cause loss of data. The receiver's use

of this field is optional.

Number of Bytes Remaining Iff the number of bytes remaining is sent,

a single space separates this field from the previous field.

This field is coded as a decimal number, and includes the

current file. This field is an estimate, and incorrect values

must not be allowed to cause loss of data. The receiver's use

of this field is optional.

File Type Iff the file type is sent, a single space separates this

field from the previous field. This field is coded as a

decimal number. Currently defined values are:

0 Sequential file - no special type

1 Other types to be defined.

The receiver's use of this field is optional.

The file information is terminated by a null. If only the pathname

is sent, the pathname is terminated with two nulls. The length of

the file information subpacket, including the trailing null, must

not exceed 1024 bytes; a typical length is less than 64 bytes.

14. PERFORMANCE RESULTS

14.1 Compatibility

Extensive testing has demonstrated ZMODEM to be compatible with

satellite links, packet switched networks, microcomputers,

minicomputers, regular and error correcting buffered modems at 75 to

19200 bps. ZMODEM's economy of reverse channel bandwidth allows

modems that dynamically partition bandwidth between the two

directions to operate at optimal speeds.

Chapter 14 Rev Oct-14-88 Typeset 10-14-88 33

Chapter 14 ZMODEM Protocol 34

14.2 Throughput

Between two single task PC-XT computers sending a program image on

an in house Telenet link, SuperKermit provided 72 ch/sec throughput

at 1200 baud. YMODEM-k yielded 85 chars/sec, and ZMODEM provided

113 chars/sec. XMODEM was not measured, but would have been much

slower based on observed network propagation delays.

Recent tests downloading large binary files to an IBM PC (4.7 mHz

V20) running YAMK 16.30 with table driven 32 bit CRC calculation

yielded a throughput of 1870 cps on a 19200 bps direct connection.

Tests with TELEBIT TrailBlazer modems have shown transfer rates

approaching 1400 characters per second for long files. When files

are compressed, effective transfer rates of 2000 characters per

second are possible.

14.3 Error Recovery

Some tests of ZMODEM protocol error recovery performance have been

made. A PC-AT with SCO SYS V Xenix or DOS 3.1 was connected to a PC

with DOS 2.1 either directly at 9600 bps or with unbuffered dial-up

1200 bps modems. The ZMODEM software was configured to use 1024

byte data subpacket lengths above 2400 bps, 256 otherwise.

Because no time delays are necessary in normal file transfers, per

file negotiations are much faster than with YMODEM, the only

observed delay being the time required by the program(s) to update

logging files.

During a file transfer, a short line hit seen by the receiver

usually induces a CRC error. The interrupt sequence is usually seen

by the sender before the next data subpacket is completely sent, and

the resultant loss of data throughput averages about half a data

subpacket per line hit. At 1200 bps this is would be about .75

second lost per hit. At 10-5 error rate, this would degrade

throughput by about 9 per cent.

The throughput degradation increases with increasing channel delay,

as more data subpackets in transit through the channel are discarded

when an error is detected.

A longer noise burst that affects both the receiver and the sender's

reception of the interrupt sequence usually causes the sender to

remain silent until the receiver times out in 10 seconds. If the

round trip channel delay exceeds the receiver's 10 second timeout,

recovery from this type of error may become difficult.

Noise affecting only the sender is usually ignored, with one common

exception. Spurious XOFF characters generated by noise stop the

Chapter 14 Rev Oct-14-88 Typeset 10-14-88 34

Chapter 14 ZMODEM Protocol 35

sender until the receiver times out and sends an interrupt sequence

which concludes with an XON.

In summation, ZMODEM performance in the presence of errors resembles

that of X.PC and SuperKermit. Short bursts cause minimal data

retransmission. Long bursts (such as pulse dialing noises) often

require a timeout error to restore the flow of data.

15. PACKET SWITCHED NETWORK CONSIDERATIONS

Flow control is necessary for printing messages and directories, and

for streaming file transfer protocols. A non transparent flow

control is incompatible with XMODEM and YMODEM transfers. XMODEM

and YMODEM protocols require complete transparency of all 256 8 bit

codes to operate properly.

The "best" flow control (when X.25 or hardware CTS is unavailable)

would not "eat" any characters at all. When the PAD's buffer almost

fills up, an XOFF should be emitted. When the buffer is no longer

nearly full, send an XON. Otherwise, the network should neither

generate nor eat XON or XOFF control characters.

On Telenet, this can be met by setting CCIT X3 5:1 and 12:0 at both

ends of the network. For best throughput, parameter 64 (advance

ACK) should be set to something like 4. Packets should be forwarded

when the packet is a full 128 bytes, or after a moderate delay

(3:0,4:10,6:0).

With PC-Pursuit, it is sufficient to set parameter 5 to 1 at both

ends after one is connected to the remote modem.

<ENTER>@<ENTER>

set 5:1<ENTER>

rst? 5:1<ENTER>

cont<ENTER>

Unfortunately, many PADs do not accept the "rst?" command.

For YMODEM, PAD buffering should guarantee that a minimum of 1040

characters can be sent in a burst without loss of data or generation

of flow control characters. Failure to provide this buffering will

generate excessive retries with YMODEM.

Chapter 15 Rev Oct-14-88 Typeset 10-14-88 35

Chapter 15 ZMODEM Protocol 36

TABLE 1. Network and Flow Control Compatibility

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Connectivity | Interactive| XMODEM| WXMODEM| SUPERKERMIT| ZMODEM |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Direct Connect | YES | YES | YES | YES | YES |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Network, no FC | no | YES | (4) | (6) | YES (1)|

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Net, transparent FC| YES | YES | YES | YES | YES |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Net, non-trans. FC | YES | no | no (5) | YES | YES |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Network, 7 bit | YES | no | no | YES (2) | YES (3)|

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

(1) ZMODEM can optimize window size or burst length for fastest transfers.

(2) Parity bits must be encoded, slowing binary transfers.

(3) Natural protocol extension possible for encoding data to 7 bits.

(4) Small WXMODEM window size may may allow operation.

(5) Some flow control codes are not escaped in WXMODEM.

(6) Kermit window size must be reduced to avoid buffer overrun.

16. PERFORMANCE COMPARISON TABLES

"Round Trip Delay Time" includes the time for the last byte in a packet to propagate through the operating systems and network to the receiver, plus the time for the receiver's response to that packet to propagate back to the sender.

The figures shown below are calculated for round trip delay times of 40 milliseconds and 5 seconds. Shift registers in the two computers and a pair of 212 modems generate a round trip delay time on the order of 40 milliseconds. Operation with busy timesharing computers and networks can easily generate round trip delays of five seconds. Because the round trip delays cause visible interruptions of data transfer when using XMODEM protocol, the subjective effect of these delays is greatly exaggerated, especially when the user is paying for connect time.

A 102400 byte binary file with randomly distributed codes is sent at1200 bps 8 data bits, 1 stop bit. The calculations assume no transmission errors. For each of the protocols, only the per file functions are considered. Processor and I/O overhead are not included. YM-k refers to YMODEM with 1024 byte data packets. YM-g refers to the YMODEM "g" option. ZMODEM uses 256 byte data subpackets for this example. SuperKermit uses maximum standard

Chapter 16 Rev Oct-14-88 Typeset 10-14-88 36

Chapter 16 ZMODEM Protocol 37

packet size, 8 bit transparent transmission, no run length

compression. The 4 block WXMODEM window is too small to span the 5

second delay in this example; the resulting thoughput degradation is

ignored.

For comparison, a straight "dump" of the file contents with no file

management or error checking takes 853 seconds.

TABLE 2. Protocol Overhead Information

(102400 byte binary file, 5 Second Round Trip)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Protocol | XMODEM| YM-k | YM-g| ZMODEM| SKermit| WXMODEM|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Protocol Round Trips | 804 | 104 | 5 | 5 | 5 | 4 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Trip Time at 40ms | 32s | 4s | 0 | 0 | 0 | 0 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Trip Time at 5s | 4020s | 520s | 25s | 25s | 25 | 20 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Overhead Characters | 4803 | 603 | 503 | 3600 | 38280 | 8000 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Line Turnarounds | 1602 | 204 | 5 | 5 | 2560 | 1602 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Transfer Time at 0s | 893s | 858s | 857s| 883s | 1172s | 916s |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Transfer Time at 40ms| 925s | 862s | 857s| 883s | 1172s | 916s |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

|Transfer Time at 5s | 5766s | 1378s| 882s| 918s | 1197s | 936s |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|

Figure 5. Transmission Time Comparison

(102400 byte binary file, 5 Second Round Trip)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* XMODEM

\*\*\*\*\*\*\*\*\*\*\*\* YMODEM-K

\*\*\*\*\*\*\*\*\*\* SuperKermit (Sliding Windows)

\*\*\*\*\*\*\* ZMODEM 16kb Segmented Streaming

\*\*\*\*\*\*\* ZMODEM Full Streaming

\*\*\*\*\*\*\* YMODEM-G

Chapter 16 Rev Oct-14-88 Typeset 10-14-88 37

Chapter 16 ZMODEM Protocol 38

TABLE 3. Local Timesharing Computer Download Performance

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Command | Protocol| Time/HD| Time/FD| Throughput| Efficiency|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|

|kermit -x | Kermit | 1:49 | 2:03 | 327 | 34% |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|

|sz -Xa phones.t| XMODEM | 1:20 | 1:44 | 343 | 36% |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|

|sz -a phones.t | ZMODEM | :39 | :48 | 915 | 95% |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|

Times were measured downloading a 35721 character text file at 9600

bps, from Santa Cruz SysV 2.1.2 Xenix on a 9 mHz IBM PC-AT to DOS

2.1 on an IBM PC. Xenix was in multiuser mode but otherwise idle.

Transfer times to PC hard disk and floppy disk destinations are

shown.

C-Kermit 4.2(030) used server mode and file compression, sending to

Pro-YAM 15.52 using 0 delay and a "get phones.t" command.

Crosstalk XVI 3.6 used XMODEM 8 bit checksum (CRC not available) and

an "ESC rx phones.t" command. The Crosstalk time does not include

the time needed to enter the extra commands not needed by Kermit and

ZMODEM.

Professional-YAM used ZMODEM AutoDownload. ZMODEM times included a

security challenge to the sending program.

Chapter 16 Rev Oct-14-88 Typeset 10-14-88 38

Chapter 16 ZMODEM Protocol 39

TABLE 4. File Transfer Speeds

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Prot file bytes bps ch/sec Notes |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|X jancol.c 18237 2400 53 Tymnet PTL 5/3/87 |

|X source.xxx 6143 2400 56 Source PTL 5/29/87 |

|X jancol.c 18237 2400 64 Tymnet PTL |

|XN tsrmaker.arc 25088 1200 94 GEnie PTL |

|B/ovth emaibm.arc 51200 1200 101 CIS PTL MNP |

|UUCP 74 files, each >7000 1200 102 (Average) |

|ZM jancol.c 18237 1200 112 DataPac (604-687-7144)|

|X/ovth emaibm.arc 51200 1200 114 CIS PTL MNP |

|ZM emaibm.arc 51200 1200 114 CIS PTL MNP |

|ZM frombyte87.txt 62506 1200 117 BIX |

|SK source.xxx 6143 2400 170 Source PTL 5/29/87 |

|ZM jancol.c 18237 2400 221 Tymnet PTL upl/dl |

|B/ovth destro.gif 33613 2400 223 CIS/PTL 9-12-87 |

|ZM jancol.c 18237 2400 224 Tymnet PTL |

|ZM tp40kerm.arc 112640 2400 224 BIX 6/88 |

|ZM readme.lis 9466 2400 231 BIX 6/88 |

|ZM jancol.c 18237 2400 226/218 TeleGodzilla upl |

|ZM jancol.c 18237 2400 226 Tymnet PTL 5/3/87 |

|ZM zmodem.ov 35855 2400 227 CIS PTL node |

|C jancol.c 18237 2400 229 Tymnet PTL 5/3/87 |

|ZM jancol.c 18237 2400 229/221 TeleGodzilla |

|ZM zmodem.ov 35855 2400 229 CIS PTL node upl |

|ZM jancol.c 18237 2400 232 CIS PTL node |

|QB gifeof.arc 32187 2400 232 CIS PTL node |

|ZM pcpbbs.txt 38423 2400 534 MNP Level 5 |

|ZM mbox 473104 9600 948/942 TeleGodzilla upl |

|ZM zmodem.arc 318826 14k 1357/1345 TeleGodzilla |

|ZM mbox 473104 14k 1367/1356 TeleGodzilla upl |

|ZM c2.doc 218823 38k 3473 Xenix 386 TK upl |

|ZM mbox -a 511893 38k 3860 386 Xenix 2.2 Beta |

|ZM c.doc 218823 57k 5611 AT Clone & 386 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Times are for downloads unless noted. Where two speeds are noted,

the faster speed is reported by the receiver because its transfer

time calculation excludes the security check and transaction log

file processing. The TeleGodzilla computer is a 4.77 mHz IBM PC

with a 10 MB hard disk. The 386 computer uses an Intel motherboard

at 18 mHz 1ws. The AT Clone (QIC) runs at 8 mHz 0ws.

Abbreviations:

B Compuserve B Protocol

QB Compuserve Quick-B/B+ Protocol

B/ovth CIS B with Omen Technology OverThruster(TM)

C Capture DC2/DC4 (no protocol)

K Kermit

MNP Microcom MNP error correcting SX/1200 modem

Chapter 16 Rev Oct-14-88 Typeset 10-14-88 39

Chapter 16 ZMODEM Protocol 40

PTL Portland Oregon network node

SK Sliding Window Kermit (SuperKermit) w=15

X XMODEM

XN XMODEM protocol implemented in network modes

X/ovth XMODEM, Omen Technology OverThruster(TM)

ZM ZMODEM

Chapter 16 Rev Oct-14-88 Typeset 10-14-88 40

Chapter 16 ZMODEM Protocol 41

TABLE 5. Protocol Checklist

Etc: Relay, BLAST, DART

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|Item XMODEM YMDM-k YMDM-g ZMODEM SK Etc.|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|IN SERVICE | 1977 | 1982 | 1985 | 1986 | 1985 | ? |

|VENDORS | ?? | ?? | >20 | >20 | ?? | 1 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|HOST AVAILABILITY | | | | | | |

|Compuserve | YES | - | - | YES | - | - |

|BIX | YES | - | - | YES | YES | - |

|Portal | | | YES | - | - | SOON|

|The Source | YES | - | - | - | YES | - |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_|

|USER FEATURES | | | | | | |

|User Friendly | - | - | - | YES | (10) | - |

|Commands/batch | 2\*N | 2 | 2 | 1 | 1(1) | |

|Commands/file | 2 | 0 | 0 | 0 | 0 | |

|Command Download | - | - | - | YES | YES(6)| - |

|Menu Compatible | - | - | - | YES | - | - |

|Crash Recovery | - | - | - | YES | - | ?? |

|File Management | - | - | - | YES | - | some|

|Security Check | - | - | - | YES | - | - |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_|

|COMPATIBILITY | | | | | | |

|Dynamic Files | YES | - | - | YES | YES | ? |

|Packet SW NETS | - | - | - | YES | YES | ? |

|7 bit PS NETS | - | - | - | (3) | YES | ? |

|Old Mainframes | - | - | - | (3) | YES | ? |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_|

|ATTRIBUTES | | | | | | |

|Reliability(5) | fair | fair(5)| none | BEST | good | ? |

|Streaming | - | - | YES | YES | YES | |

|Overhead(2) | 7% | 1% | 1% | 4%(8) | 30% | |

|Faithful Xfers | - | YES(7) | YES(7)| YES | YES | ? |

|Preserve Date | - | YES | YES | YES | - | ? |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_|

|COMPLEXITY | | | | | | |

|No-Wait Sample | - | - | - | opt | REQD | REQD|

|Ring Buffers | - | - | - | opt | REQD | REQD|

|Complexity | LOW(5)| LOW(5) | LOW | MED | HIGH | ? |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_\_\_\_\_|

|EXTENSIONS | | | | | | |

|Server Operation | - | - | - | YES(4)| YES | ? |

|Multiple Threads | - | - | - | future| - | - |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

NOTES:

(1) Server mode or Omen Technology Kermit AutoDownload

(2) Character count, binary file, transparent channel

Chapter 16 Rev Oct-14-88 Typeset 10-14-88 41

Chapter 16 ZMODEM Protocol 42

(3) Future enhancement provided for

(4) AutoDownload operation

(5) Omen Technology's Cybernetic Data Recovery(TM) improves XMODEM

and YMODEM reliability with complex proprietary logic.

(6) Server commands only

(7) Only with True YMODEM(TM)

(8) More then 3% from protected network control characters

(9) With Segmented Streaming

(10) With Pro-YAM extensions

Chapter 16 Rev Oct-14-88 Typeset 10-14-88 42

Chapter 16 ZMODEM Protocol 43

17. FUTURE EXTENSIONS

Future extensions include:

+ Compatibility with 7 bit networks

+ Server/Link Level operation: An END-TO-END error corrected

program to program session is required for financial and other

sensitive applications.

+ Multiple independent threads

+ Bidirectional transfers (STEREO ZMODEM)

+ Encryption

+ Compression

+ File Comparison

+ Selective transfer within a file (e.g., modified segments of a

database file)

+ Selective Retransmission for error correction

18. REVISIONS

10-14-88 Pascal source code now available in Phil Burn's PibTerm

v4.2. 6-24-88 An exception to the previously unconditional ZCBIN

override: a ZCRESUM specified by the receiver need not be overridden

by the sender's ZCBIN.

11-18-87 Editorial improvements

10-27-87 Optional fields added for number of files remaining to be

sent and total number of bytes remaining to be sent.

07-31-1987 The receiver should ignore a ZEOF with an offset that

does not match the current file length. The previous action of

responding with ZRPOS caused transfers to fail if a CRC error

occurred immediately before end of file, because two retransmission

requests were being sent for each error. This has been observed

under exceptional conditions, such as data transmission at speeds

greater than the receiving computer's interrupt response capabilitiy

or gross misapplication of flow control.

Discussion of the Tx backchannel garbage count and ZCRCW after error

ZRPOS was added. Many revisions for clarity.

07-09-87 Corrected XMODEM's development date, incorrectly stated as

Chapter 18 Rev Oct-14-88 Typeset 10-14-88 43

Chapter 18 ZMODEM Protocol 44

1979 instead of the actual August 1977. More performance data was

added.

05-30-87 Added ZMNEW and ZMSKNOLOC

05-14-87 Window management, ZACK zshhdr XON removed, control

character escaping, ZMSPARS changed to ZXPARS, editorial changes.

04-13-87 The ZMODEM file transfer protocol's public domain status

is emphasized.

04-04-87: minor editorial changes, added conditionals for overview

version.

03-15-87: 32 bit CRC added.

12-19-86: 0 Length ZCRCW data subpacket sent in response to ZPAD or

ZDELE detected on reverse channel has been changed to ZCRCE. The

reverse channel is now checked for activity before sending each

ZDATA header.

11-08-86: Minor changes for clarity.

10-2-86: ZCNL definition expanded.

9-11-86: ZMPROT file management option added.

8-20-86: More performance data included.

8-4-86: ASCII DLE (Ctrl-P, 020) now escaped; compatible with

previous versions. More document revisions for clarity.

7-15-86: This document was extensively edited to improve clarity and

correct small errors. The definition of the ZMNEW management option

was modified, and the ZMDIFF management option was added. The

cancel sequence was changed from two to five CAN characters after

spurious two character cancel sequences were detected.

19. MORE INFORMATION

Please contact Omen Technology for troff source files and typeset

copies of this document.

19.1 TeleGodzilla Bulletin Board

More information may be obtained by calling the TeleGodzilla

bulletin board at 503-621-3746. TeleGodzilla supports 19200

(Telebit PEP), 2400 and 1200 bps callers with automatic speed

recognition.

Chapter 19 Rev Oct-14-88 Typeset 10-14-88 44

Chapter 19 ZMODEM Protocol 45

Relevant files include YZMODEM.ZOO, YAMDEMO.ZOO, YAMHELP.ZOO,

ZCOMMEXE.ARC, ZCOMMDOC.ARC, ZCOMMHLP.ARC.

Useful commands for TeleGodzilla include "menu", "dir", "sx file

(XMODEM)", "kermit sb file ...", and "sz file ...".

19.2 Unix UUCP Access

UUCP sites can obtain the current version of this file with

uucp omen!/u/caf/public/zmodem.doc /tmp

A continually updated list of available files is stored in

/usr/spool/uucppublic/FILES.

uucp omen!~uucp/FILES /usr/spool/uucppublic

The following L.sys line allows UUCP to call site "omen" via Omen's

bulletin board system "TeleGodzilla". TeleGodzilla is an instance

of Omen Technology's Professional-YAM in host operation, acting as a

bulletin board and front ending a Xenix system.

In response to TeleGodzilla's "Name Please:" (e:--e:), uucico gives

the Pro-YAM "link" command as a user name. Telegodzilla then asks

for a link password (d:). The password (Giznoid) controls access to

the Xenix system connected to the IBM PC's other serial port.

Communications between Pro-YAM and Xenix use 9600 bps; YAM converts

this to the caller's speed.

Finally, the calling uucico sees the Xenix "Login:" message (n:--

n:), and logs in as "uucp". No password is used for the uucp

account.

omen Any ACU 2400 1-503-621-3746 e:--e: link d: Giznoid n:--n: uucp

20. ZMODEM PROGRAMS

A copy of this document, a demonstration version of

Professional-YAM, a flash-up tree structured help file and

processor, are available in YZMODEM.ZOO on TeleGodzilla and other

bulletin boards. This file must be unpacked with LOOZ.EXE, also

available on TeleGodzilla. YZMODEM.ZOO may be distributed provided

none of the files are deleted or modified without the written

consent of Omen Technology.

TeleGodzilla and other bulletin boards also feature ZCOMM, a

shareware communications program. ZCOMM includes Omen Technology's

TurboLearn(TM) Script Writer, ZMODEM, Omen's highly acclaimed XMODEM

and YMODEM protocol support, Sliding Windows Kermit, several

traditional protocols, a powerful script language, and the most

accurate VT100/102 emulation available in a usr supported program.

The ZCOMM files include:

Chapter 20 Rev Oct-14-88 Typeset 10-14-88 45

Chapter 20 ZMODEM Protocol 46

+ ZCOMMEXE.ARC Executable files and beginner's telephone directory

+ ZCOMMDOC.ARC "Universal Line Printer Edition" Manual

+ ZCOMMHLP.ARC Tree structured Flash-UP help processor and

database

C source code and manual pages for the Unix/Xenix rz and sz programs

are available on TeleGodzilla in RZSZ.ZOO. This ZOO archive may be

unpacked with LOOZ.EXE, also available on TeleGodzilla. Most Unix

like systems are supported, including V7, Sys III, 4.x BSD, SYS V,

Idris, Coherent, and Regulus.

RZSZ.ZOO includes a ZCOMM/Pro-YAM/PowerCom script ZUPL.T to upload

the small (178 lines) YMODEM bootstrap program MINIRB.C without a

file transfer protocol. MINIRB uses the Unix stty(1) program to set

the required raw tty modes, and compiles without special flags on

virtually all Unix and Xenix systems. ZUPL.T directs the Unix

system to compile MINIRB, then uses it as a bootstrap to upload the

rz/sz source and manual files.

Pascal source code for ZMODEM support is available in PibTerm v4.2

written by Phil Burns.

The PC-DOS EXEC-PC, QuickBBS, Opus and Nochange bulletin boards

support ZMODEM. Integrated ZMODEM support for the Collie bulletin

board program is planned. Most of the PC-DOS bulletin board

programs that lack integrated ZMODEM support ZMODEM with external

modules (DSZ, etc.).

The BinkleyTerm, Dutchie and D'Bridge email systems support ZMODEM

as their primary protocol.

The IN-SYNCH PC-DOS Teleconferencing system uses ZMODEM.

The LAN modem sharing program Line Plus has announced ZMODEM

support.

Many programs have added direct ZMODEM support, including Crosstalk

Mark IV, and Telix 3.

Most other PC-DOS communications programs support external ZMODEM

via Omen Technology's DSZ, including PibTerm, Qmodem SST and BOYAN.

The ZMDM communications program by Jwahar Bammi runs on Atari ST

machines.

The Online! and A-Talk Gold programs for the Amiga support ZMODEM.

The Byte Information eXchange supports ZMODEM. The Compuserve

Information Service has ported the Unix rz/sz ZMODEM programs to

Chapter 20 Rev Oct-14-88 Typeset 10-14-88 46

Chapter 20 ZMODEM Protocol 47

DECSYSTEM 20 assembler, and has announced future support for ZMODEM.

20.1 Adding ZMODEM to DOS Programs

DSZ is a small shareware program that supports XMODEM, YMODEM, and

ZMODEM file transfers. DSZ is designed to be called from a bulletin

board program or another communications program. It may be called

as

dsz port 2 sz file1 file2

to send files, or as

dsz port 2 rz

to receive zero or more file(s), or as

dsz port 2 rz filea fileb

to receive two files, the first to filea and the second (if sent) to

fileb. This form of dsz may be used to control the pathname of

incoming file(s). In this example, if the sending program attempted

to send a third file, the transfer would be terminated.

Dsz uses DOS stdout for messages (no direct CRT access), acquires

the COMM port vectors with standard DOS calls, and restores the COMM

port's interrupt vector and registers upon exit.

Further information on dsz may be found in dsz.doc and the ZCOMM or

Pro-YAM user manuals.

21. YMODEM PROGRAMS

The Unix rz/sz programs support YMODEM as well as ZMODEM. Most Unix

like systems are supported, including V7, Sys III, 4.2 BSD, SYS V,

Idris, Coherent, and Regulus.

A version for VAX-VMS is available in VRBSB.SHQ, in the same

directory.

Irv Hoff has added 1k packets and YMODEM transfers to the KMD and

IMP series programs, which replace the XMODEM and MODEM7/MDM7xx

series respectively. Overlays are available for a wide variety of

CP/M systems.

Many other programs, including MEX-PLUS and Crosstalk Mark IV also

support some of YMODEM's features.

Chapter 21 Rev Oct-14-88 Typeset 10-14-88 47

Chapter 21 ZMODEM Protocol 48

Questions about YMODEM, the Professional-YAM communications program,

and requests for evaluation copies may be directed to:

Chuck Forsberg

Omen Technology Inc

17505-V Sauvie Island Road

Portland Oregon 97231

VOICE: 503-621-3406 :VOICE

Modem (TeleGodzilla): 503-621-3746

Usenet: ...!tektronix!reed!omen!caf

Compuserve: 70007,2304

Source: TCE022

22. ACKNOWLEDGMENTS

The High Reliability Software(TM), TurboLearn Script Writer(TM),

Cybernetic Data Recovery(TM), AutoDownload(TM), Intelligent Crash

Recovery(TM), Error Containment(TM), Full Time Capture(TM), True

YMODEM(TM), OverThruster(TM), Password Guardian(TM),

CryptoScript(TM), and TurboDial(TM) are Omen Technology trademarks.

ZMODEM was developed for the public domain under a Telenet contract.

The ZMODEM protocol descriptions and the Unix rz/sz program source

code are public domain. No licensing, trademark, or copyright

restrictions apply to the use of the protocol, the Unix rz/sz source

code and the ZMODEM name.

Encouragement and suggestions by Thomas Buck, Ward Christensen, Earl

Hall, Irv Hoff, Stuart Mathison, and John Wales, are gratefully

acknowledged. 32 bit CRC code courtesy Gary S. Brown.

23. RELATED FILES

The following files may be useful while studying this document:

YMODEM.DOC Describes the XMODEM, XMODEM-1k, and YMODEM batch file

transfer protocols. This file is available on TeleGodzilla

as YMODEM.DQC.

zmodem.h Definitions for ZMODEM manifest constants

rz.c, sz.c, rbsb.c Unix source code for operating ZMODEM programs.

rz.1, sz.1 Manual pages for rz and sz (Troff sources).

zm.c Operating system independent low level ZMODEM subroutines.

minirb.c A YMODEM bootstrap program, 178 lines.

Chapter 23 Rev Oct-14-88 Typeset 10-14-88 48

Chapter 23 ZMODEM Protocol 49

RZSZ.ZOO,rzsz.arc Contain the C source code and manual pages listed

above, plus a ZCOMM script to upload minirb.c to a Unix or

Xenix system, compile it, and use the program to upload the

ZMODEM source files with error checking.

DSZ.ZOO,dsz.arc Contains DSZ.COM, a shareware X/Y/ZMODEM subprogram,

DESQview "pif" files for background operation in minimum

memory, and DSZ.DOC.

ZCOMM\*.ARC Archive files for ZCOMM, a powerful shareware

communications program.

Chapter 23 Rev Oct-14-88 Typeset 10-14-88 49

CONTENTS

1. INTENDED AUDIENCE................................................ 2

2. WHY DEVELOP ZMODEM?.............................................. 2

3. ZMODEM Protocol Design Criteria.................................. 4

3.1 Ease of Use............................................... 4

3.2 Throughput................................................ 5

3.3 Integrity and Robustness.................................. 6

3.4 Ease of Implementation.................................... 6

4. EVOLUTION OF ZMODEM.............................................. 7

5. ROSETTA STONE.................................................... 10

6. ZMODEM REQUIREMENTS.............................................. 10

6.1 File Contents............................................. 10

7. ZMODEM BASICS.................................................... 12

7.1 Packetization............................................. 12

7.2 Link Escape Encoding...................................... 12

7.3 Header.................................................... 13

7.4 Binary Data Subpackets.................................... 16

7.5 ASCII Encoded Data Subpacket.............................. 16

8. PROTOCOL TRANSACTION OVERVIEW.................................... 16

8.1 Session Startup........................................... 16

8.2 File Transmission......................................... 18

8.3 Session Cleanup........................................... 20

8.4 Session Abort Sequence.................................... 20

9. STREAMING TECHNIQUES / ERROR RECOVERY............................ 21

9.1 Full Streaming with Sampling.............................. 21

9.2 Full Streaming with Reverse Interrupt..................... 23

9.3 Full Streaming with Sliding Window........................ 23

9.4 Full Streaming over Error Free Channels................... 24

9.5 Segmented Streaming....................................... 24

10. ATTENTION SEQUENCE............................................... 24

11. FRAME TYPES...................................................... 25

11.1 ZRQINIT................................................... 25

11.2 ZRINIT.................................................... 25

11.3 ZSINIT.................................................... 25

11.4 ZACK...................................................... 26

11.5 ZFILE..................................................... 26

11.6 ZSKIP..................................................... 28

11.7 ZNAK...................................................... 28

11.8 ZABORT.................................................... 28

- i -

11.9 ZFIN...................................................... 28

11.10 ZRPOS..................................................... 28

11.11 ZDATA..................................................... 29

11.12 ZEOF...................................................... 29

11.13 ZFERR..................................................... 29

11.14 ZCRC...................................................... 29

11.15 ZCHALLENGE................................................ 29

11.16 ZCOMPL.................................................... 29

11.17 ZCAN...................................................... 29

11.18 ZFREECNT.................................................. 29

11.19 ZCOMMAND.................................................. 29

12. SESSION TRANSACTION EXAMPLES..................................... 30

12.1 A simple file transfer.................................... 30

12.2 Challenge and Command Download............................ 31

13. ZFILE FRAME FILE INFORMATION..................................... 31

14. PERFORMANCE RESULTS.............................................. 33

14.1 Compatibility............................................. 33

14.2 Throughput................................................ 34

14.3 Error Recovery............................................ 34

15. PACKET SWITCHED NETWORK CONSIDERATIONS........................... 35

16. PERFORMANCE COMPARISON TABLES.................................... 36

17. FUTURE EXTENSIONS................................................ 43

18. REVISIONS........................................................ 43

19. MORE INFORMATION................................................. 44

19.1 TeleGodzilla Bulletin Board............................... 44

19.2 Unix UUCP Access.......................................... 45

20. ZMODEM PROGRAMS.................................................. 45

20.1 Adding ZMODEM to DOS Programs............................. 47

21. YMODEM PROGRAMS.................................................. 47

22. ACKNOWLEDGMENTS.................................................. 48

23. RELATED FILES.................................................... 48

LIST OF FIGURES

Figure 1. Order of Bytes in Header................................... 14

Figure 2. 16 Bit CRC Binary Header................................... 14

- ii -

Figure 3. 32 Bit CRC Binary Header................................... 14

Figure 4. HEX Header................................................. 15

Figure 5. Transmission Time Comparison............................... 37

LIST OF TABLES

TABLE 1. Network and Flow Control Compatibility...................... 36

TABLE 2. Protocol Overhead Information............................... 37

TABLE 3. Local Timesharing Computer Download Performance............. 38

TABLE 4. File Transfer Speeds........................................ 39

TABLE 5. Protocol Checklist.......................................... 41

- iii -

The ZMODEM Inter Application File Transfer Protocol

Chuck Forsberg

Omen Technology Inc

ABSTRACT

The ZMODEM file transfer protocol provides reliable file and command

transfers with complete END-TO-END data integrity between application

programs. ZMODEM's 32 bit CRC protects against errors that continue to

sneak into even the most advanced networks.

Unlike traditional and many recently introduced protocols, ZMODEM

safeguards all data and supervisory information with effective error

detection.

ZMODEM rapidly transfers files, particularly with buffered (error

correcting) modems, timesharing systems, satellite relays, and wide area

packet switched networks.

User Friendliness is an important ZMODEM feature. ZMODEM AutoDownload

(Automatic file Download initiated without user intervention) greatly

simplifies file transfers compared to the traditional protocols.

ZMODEM provides advanced file management features including Crash

Recovery, flexible control of selective file transfers, and security

verified command downloading.

ZMODEM protocol features allow implementation on a wide variety of systems

operating in a wide variety of environments. A choice of buffering and

windowing modes allows ZMODEM to operate on systems that cannot support

other streaming protocols. Finely tuned control character escaping allows

operation with real world networks without Kermit's high overhead.

ZMODEM is the only high performance high reliability public protocol that

does not require large buffer allocations for normal file transfers.

Although ZMODEM software is more complex than unreliable XMODEM routines,

a comphrensive protocol description and actual C source code to production

programs have allowed dozens of developers to upgrade their applications

with efficient, reliable ZMODEM file transfers with a minimum of effort.

ZMODEM was developed for the public domain under a Telenet contract. The

ZMODEM protocol descriptions and the Unix rz/sz program source code are

public domain. No licensing, trademark, or copyright restrictions apply

to the use of the protocol, the Unix rz/sz source code and the ZMODEM

name.