A Proposal For An ARQ Protocol For Use With MT63 And Similar Modes

Revision 0.2

Paul L Schmidt, K9PS
December, 2004

Introduction

This is a partial first draft for an ARQ protocol for MT63 or similar 7-bit data stream. Multiple streams (such as are supported by WA8DED's host mode protocol) are supported. This description has, as much as possible, been written in straightforward English rather than in technical jargon common to communications and data protocols. It is hoped that this format will be sufficient to be understandable by the average ham, but descriptive enough to allow implementation of the protocol described. Some technical terms (e.g. CRC, ARQ) are included, but their definitions should be readily obtainable from generic sources.

The intent of the protocol is to provide a link layer wrapper to encapsulate the physical layer of MT63 or similar streamed protocol. The use of ASCII-readable data has been preferred over attempting to get the maximum utility from each bit; this is to make the transmitted signal as readable as possible to stations not having ARQ capability, and to permit easy debugging and diagnosis of problems. In order to identify complete frames of data, it is necessary only to locate start of header <SOH> and end of transmission <EOT> ASCII characters within a stream of data. These two characters are prohibited from appearing elsewhere in the transmitted signal: a frame can then be easily identified as a block of characters beginning with an <SOH> character and ending with the last character received before either the next <SOH> or an <EOT>:

Frame	Frame	Frame	Frame	
<SOH>....bytes…	<SOH>….bytes……	<SOH>…..bytes……	<SOH>….bytes……	<EOT>

A logical stream identifier is assigned by the station at each end of the link, specifying a connection that has been set up in a manner similar to Internet Protocol: a source and destination host and service identifier (corresponding to a TCP or UDP port number) are used, as well as a protocol type. Numbers corresponding to IP services are recommended: SMTP as port 25, a generic user source socket being 1025, etc. Recommended additional standard service numbers to be implemented would be BBS interface on service 24 (i.e. private mail protocol), and keyboard-to-keyboard connections on service 87.

The mapping of streams to services allows multiple services to exist at the same station without requiring the overhead of fully specifying everything in every packet. (It also allows a framework by which internet connectivity can be realized in the future without having to deal with it at this time.)

A transmission from a station is composed of one or more frames, each containing a header, an optional payload, and a CRC. The format of a frame is:

Header (4 bytes)	Payload (0 to 512 bytes)	CRC (4 bytes)
<SOH> byte2 byte3 byte4	byte1 byte2 byte3 ….. byte(n-1) byte(n)	byte1 byte2 byte3 byte4

The header is of known length (which may be dependent on the protocol identifier – for the version described in this draft, that length is 4 bytes). The CRC will be the final 4 bytes in the frame, with any bytes remaining between the groups being payload bytes.

The CRC will be a CRC-16 covering the bytes from <SOH> to the last byte of the payload, inclusive, and will be expressed as a 4-byte uppercase hexadecimal string (e.g. “EF12”) in “network order” - high byte first.

Frames should be sent in the following order:
1 - Identification
2..N-1 - Data frames (if needed)
N - Ack / Nak / Retransmission Request or Poll, as required

Extra identification frames may be inserted at any point, and the addition of at least one (probably at the end of the transmission) is particularly recommended if several data frames are sent in a single transmission (doing so will assure that the receiving station gets at least one ID frame so he knows the data is coming to his station!).

Multiple <SOH> characters are to be considered as a single character, as are multiple <EOT> characters.

Block counting and re-transmission.

This protocol uses a 6-bit counter, which counts from (0 to 63) to number frames transmitted between stations. Each station can configure the maximum number of frames that can be sent at one time, but the sending station cannot ever let the counter be more than 62 ahead of the oldest unacknowledged frame's count. For example, if Frame #4 is not acknowledged, transmission can continue with frames up to frame 63 (an increase of 59), then wrap to 0 (an increase of 60), and continue to 1 and 2 (an increase of 62). The new frame 3 cannot be sent until the old frame 4 has been acknowledged. (This should be an extreme case and should never be seen in practice!)

The difference between this protocol and other protocols typically used by amateurs is that selected blocks can be retransmitted, without requiring the retransmission of data already received correctly.

The Header

General rules and definition of Bytes 1,2 - Protocol

The first byte of the header will be the ASCII <SOH> (0x01) character; all remaining bytes in the header will be will be “printable ASCII”: characters from 0x20 to 0x7E. Byte 2 of the header will indicate the protocol version: in order to make monitoring of the data to be “not too ugly”, the value of the byte will begin at 0x30 (ASCII “0”) for the initial version. For this protocol version, the header will consist of exactly four bytes, so there is no additional delimiter character needed to mark the boundary between the header and any payload bytes.

The initial protocol will have 2 additional bytes in the header:

Header Byte 3 - Stream ID

Byte 3: Receiving Station Stream ID, beginning at 0x30 (ASCII '0').
Stream ID '0' will indicate “unknown” or “not assigned”; active stream identifiers will begin at 0x30 ('1').

Header Byte 4 - Block Type

Byte 4: Block Type Description.
0x00 - 0x1f – reserved
0x20 - 0x5f – data blocks with block sequences 0x00 - 0x3f (After transmitting block 0x3f, the counter starts again at 0x00)
0x40 - 0x7e -- reserved except for:
'i' - Identification
'c' - Connection Request
'k' - Connection Request Acknowledge
'r' - Connection Refused
'd' - Disconnection Request
's' - Data Ack / Retransmission Request (status)
'p' - Poll status
'f' - Format Failure

Block Types 0x20 to 0x5f - Data Blocks

Data transfer between stations will be accomplished through the use of numbered data blocks (numbered 0 to 63, or in hexadecimal 0x00 to 0x3f).
In order to avoid the use of bytes 0x00 - 0x1f or 0x7f in the header, the block number will be offset by 0x20. Bytes

Block Type 'i' - Identification

The identification payload will be a traditional station ID string: receiving station's callsign , followed by a space, “DE”, a space, and the sending station's callsign.

Example: “OA2VR/VE3 DE 5Y3GTB”

Block Type 'c' - Connection Request

The payload of a Connect request shall be a text string containing both host callsigns, a stream identifier for the requesting station's receive stream, port identifiers for both stations, type of service requested, and maximum block size for data blocks (log2: 7=128, 8=256, etc.). An optional extension will permit negotiation of data types supported: if only 7-bit ASCII is supported, there will be no further data. Additional frame types will be indicated by the addition of the <STX> character and the characters indicating supported types. Note that no specific length fields are allocated for callsigns, so that long callsigns can be accommodated. For example, if OA2VR/VE3 wants to open up a connection to an SMTP server at 5Y3GTB's station, assign it to stream 1 with 256-byte blocks at his (OA2VR's) station, and support optional escaped ASCII or binary frames, the string would be: “OA2VR/VE3:1023 5Y3GTB:25 1 8<STX>ab”. While not necessarily the shortest implementation, it is readable and understandable by a generic (non-ARQ) receiver. The connection request will be sent to the “undefined” stream of the receiving station (i.e. '0'). The Connect Request is by definition block #0.

The block size parameter should not be less than 4 (16 byte data blocks), and with current technology should probably never exceed 9 (512 bytes). A nominal default value of 6 (64 bytes) is recommended.

Block Type 'k' - Connection Request Acknowledge

The message is the receiving station's version of a connection request; the callsign and port orders are transposed, the stream identifier is the stream assigned by the station accepting the request. The maximum block size is the higher of the number requested or the accepting station's maximum. An optional extension will permit negotiation of data types supported: if only 7-bit ASCII is supported, there will be no further data. Additional frame types will be indicated by the addition of the <STX> character and the characters indicating supported types. The connection request acknowledge will be sent to the data stream indicated by the connecting station. For example, if the request used as an example in the connection request description: “OA2VR/VE3:1023 5Y3GTB:25 1 8 <STX>b” is received at 5Y3GTB, and 5Y3GTB will accept it as stream 2 with a maximum block size of 64 bytes, and is only able to handle optional binary frames, the reply of “5Y3GTB:25 OA2VR/VE3 2 6<STX>b” will be sent to OA2VR/VE3's stream 1. The Connect Request Acknowledge is by definition block #0.

Block Type 'r' - Connection Refused

This message is sent if a connection is refused -- i.e. the station receiving the connection request cannot honor it. Future enhancements to the protocol should allow for numeric descriptions explaining the reason for the refusal (e.g. too many streams already in use, etc.), but for now, responses in the text field should be “00” (with no text explanation), or “01 “ (generic failure message) followed by a brief text string.
Examples:
“00”
“01 NO AVAILABLE STREAMS”
“01 NO CLIENT FOR THAT SERVICE”
“01 YOUR STATION LOCKED OUT - CONTACT SYSOP BY OTHER MEANS”

Block Type 'd' - Disconnection Request

There is no data associated with this frame; just a sequence number and the frame type. Since there is a sequence number, the receiving station will not process the request until it has successfully received all data transmitted before the disconnection request. An acknowledgement from the receiving station is required.

Block Type 's' - Data Ack / Retransmission Request

The Data Ack / Retransmission Request frame is sent in response to reception of any data frame or a poll request. It is a string of 2 or more characters:
First character - the last data block sequence number transmitted, offset by 0x20.
Second character - the last data block sequence number received correctly with no gaps, offset by 0x20.

Third character - the last data block sequence number received, offset by 0x20.
Following characters - data block sequence numbers that were received with errors.

Example: The transmitting station sends frames 0x26 through 0x35, and the receiving station has errors in receiving 0x2D, 0x30, and 0x32. The last frame transmitted by the receiving station was 0x29 which has been acknowledged, and has no data that currently needs sent. After the sending station finishes, the receiver transmits an identification frame followed by a Data Ack / Retransmission Request with the string of characters ILUMPR which indicate:

Character 1: 'I' I The last frame sent was frame 0x29 (0x29+0x20=0x49, an ASCII 'I')
Character 2: 'L' everything up to and including frame 0x2C has been received correctly (0x2C+0x20=0x4C, an ASCII 'L')
Character 3: 'U' the latest frame received was frame 0x35 (indicated by 'U')
Characters 4-6: 'MPR' retransmission of 3 frames from your last transmission is needed: 0x2D (the M), 0x30 (the P), and 0x32 (the R)

Assuming the sending station has been set up to have up to 12 frames outstanding (his previous transmission was 16 frames), he will continue on by retransmitting frames 0x2D, 0x30, and 0x32, and sending frames 0x36, 0x37, 0x38, 0x38, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x00, 0x01, and 0x02.

Block Type 'q' - Poll

The Poll frame is identical to a Data Ack / Retransmission request, except that reception of a poll REQUIRES a the receiving station reply with a Data Ack / Retransmission Request.

Data Format

All data within a block will be interpreted as ASCII data, unless the first byte of data within the block is the ASCII <STX> (start of text) character. If the first byte within the block *is* <STX>, the second byte will indicate how the data is to be interpreted.
Byte 2: data format:
'a' - ascii data stream (alternate method)
'b' - binary data stream
'c' - Huffman compressed (to be defined later)
'z' - zlib compressed (to be defined later)
other values - reserved

Note that since this byte exists in all frames, there is no need to specify if the stream needs to support 8-bit data when opening it. The sending station can assemble the packet “on the fly” using the most appropriate mode available.

The frame's data: the Payload

First Byte NOT <STX>

If the first byte of the payload is *not* <STX>, the frame's payload is to be interpreted as 7-bit ASCII characters. The payload may contain any 7-bit characters except <SOH>, <EOT>, or <STX>. If these characters exist in the data stream, they will be deleted before transmission. ALL implementations must support this mode – support of additional frames (binary, compressed, etc.) is optional.

First Byte IS <STX>

If the first byte of the payload *is* <STX>, the frame's payload is defined by the second byte. Implementations supporting ANY of these modes must support <STX>b – the binary mode. Additional modes will be implementation dependent, but the use of definitions must be coordinated between implementations, and fully defined.

First Two Bytes <STX> a – 'escaped' ASCII data

If the first two bytes of the payload are <STX>a, the frame's payload is to be interpreted as primarily 7-bit ASCII characters. If a character with the 8th bit set must be transmitted, the <SUB> (substitute, 0x1A) ASCII character is transmitted, and the next character will be the 8-bit character with the most-significant bit forced to zero. For example, the character 0xFA would be sent as 0x1A 0x7A, or <SUB> z.

If a string of characters with the 8th bit set are required, the <SO> (ASCII shift out, 0x0E) may be used to toggle the stream into a mode where bit 8 is inverted on all characters until the <SI> (ASCII shift-in, 0x0F) character is received. (If 'shifted out', the <SUB> character sequence can be used to set the bit back to zero for a single character in the string of 'shifted' characters.)

In summary:

<SUB> is used to set the 8th bit the character immediately following, for data streams which are predominantly 7-bit with only occasional 8-bit data. (e.g. the extended ASCII characters: <SUB>A = 0xC1).

<SO> toggles the stream into a mode where bit 8 is set until <SI> is received.

It is also necessary to prevent the <SOH>, <STX>, <SUB>, <EOT>, <SO>, and <SI> characters from appearing in the frame's payload, since these characters have other specific meanings. The <DLE> will be used to 'escape' and pass reserved characters: upon receipt of the <DLE> character, the receiving station will removed it from the stream, and modify the character immediately after it by either:
  subtracting 0x40 from it, if the character is between 0x40 and 0x5f
  adding 0x20 to it, if the character is between 0x60 and 0x7f. (This allows passing the characters 0x80 to 0x9f without having to escape twice - once to get the reserved 7-bit character, and another time to set its eighth bit)

<DLE>@	= <DLE>	(0x10)
<DLE>D	= <EOT>	(0x04)
<DLE>N	= <SO>	(0x0e)
<DLE>O	= <SI>	(0x0f)
<DLE>Z	= <SUB>	(0x1a)
<DLE>z	=	(0x9a)

Support of this option is optional, but recommended.

First Two Bytes <STX> b - Binary data

Binary Mode: Same as for text, except <DLE> is only needed to escape the sending of itself, <SOH>, and <EOT>. (We don't want to allow the <SOH> or <EOT> bytes to appear in the transmitted data, to avoid confusion on the receiving end; and we need to protect <DLE> for use as the escaping character in the binary frame payload.) The lower seven bits of each byte are used to form a continuous bit stream, which is then split into 8-bit bytes. The final 4 bytes received in the frame (marked either by the receipt of an <EOT> or another <SOH>) are the CRC, and any remaining bits in the last byte before the CRC are discarded.
Example: the data stream 0xDF 0x2C 0x26 0xEF becomes o K <DLE> D n z.

The data to be transmitted:
0xDF   0x2C   0x26   0xEF

Converted to binary representation:
1101 1111 0010 1100 0010 0110 1110 1111

Bytes run together:
11011111001011000010011011101111

Re-grouped as 7-bit characters:
110 1111   100 1011   000 0100   110 1110   111 1(000)

Expressed as Hexadecimal:
0x6f   0x4b   0x04   0x6e   0x78

Expressed as ASCII (with the 0x04 <EOT> character “escaped”)
o   K   <DLE> D   n   z

If ANY options are supported, support of this option is REQUIRED.

Data Format 'c' - Huffman compressed

(support is optional - to be defined later)

Data Format 'z' - zlib compressed

(support is optional - to be defined later)