M17 Protocol Specification

DRAFT

M17 Working Group:

Wojciech	SP5WWP
Juhani	OH1CAU
Elms	KM6VMZ
Nikoloz	SO3ALG
Mark	KR6ZY
Steve	KC1AWV
Rob	WX9O
Tom	N7TAE
Mike	W2FBI

Feb 28, 2021

Published February 28, 2021

Part I - Air and IP Interface:

CHAPTER 1

M17 RF Protocol: Summary

M17 is an RF protocol that is:

- Completely open: open specification, open source code, open source hardware, open algorithms. Anyone must be able to build an M17 radio and interoperate with other M17 radios without having to pay anyone else for the right to do so.
- Optimized for amateur radio use.
- Simple to understand and implement.
- Capable of doing the things hams expect their digital protocols to do:
  - Voice (eg: DMR, D-Star, etc)
  - Point to point data (eg: Packet, D-Star, etc)
  - Broadcast telemetry (eg: APRS, etc)
  - Extensible, so more capabilities can be added over time.

To do this, the M17 protocol is broken down into three protocol layers, like a network:

1. Physical Layer: How to encode 1s and 0s into RF. Specifies RF modulation, symbol rates, bits per symbol, etc.
2. Data Link Layer: How to packetize those 1s and 0s into usable data. Packet vs Stream modes, headers, addressing, etc.
3. Application Layer: Accomplishing activities. Voice and data streams, control packets, beacons, etc.

This document attempts to document these layers.

CHAPTER 2

Glossary

ECC Error Correcting Code

FEC Forward Error Correction

Frame The individual components of a stream, each of which contains payload data interleaved with frame signalling.

Link Information Frame The first frame of any transmission. It contains full LICH data.

LICH Link Information Channel. The LICH contains all information needed to establish an M17 link. The first frame of a transmission contains full LICH data, and subsequent frames each contain one sixth of the LICH data so that late-joiners can obtain the LICH.

Packet A single burst of transmitted data containing 100s to 1000s of bytes, after which the physical layer stops sending data.

Superframe A set of six consecutive frames which collectively contain full LICH data are grouped into a superframe.

CHAPTER 3

Physical Layer

3.1 4FSK generation

M17 standard uses 4FSK modulation running at 4800 symbols/s (9600 bits/s) with a deviation index h=0.33 for transmission in 9 kHz channel bandwidth. Channel spacing is 12.5 kHz. The symbol stream is converted to a series of impulses which pass through a root-raised-cosine (alpha=0.5) shaping filter before frequency modulation at the transmitter and again after frequency demodulation at the receiver.

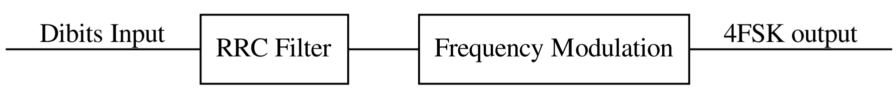

Fig. 3.1: 4FSK modulator dataflow

The bit-to-symbol mapping is shown in the table below.

Table 3.1: Dibit symbol mapping to 4FSK deviation

Information bits		Symbol	4FSK deviation
Bit 1	Bit 0		
0	1	+3	+2.4 kHz
0	0	+1	+0.8 kHz
1	0	-1	-0.8 kHz
1	1	-3	-2.4 kHz

The most significant bits are sent first, meaning that the byte 0xB4 in type 4 bits (see *Bit types*) would be sent as the symbols -1 -3 +3 +1.

3.2 Preamble

Every transmission starts with a preamble, which shall consist of at least 40ms of alternating -3, +3… symbols. This is equivalent to 40 milliseconds of a 2400 Hz tone

3.3 Bit types

The bits at different stages of the error correction coding are referred to with bit types, given in Table 3.2.

Table 3.2: Bit types

Type 1	Data link layer bits
Type 2	Bits after appropriate encoding
Type 3	Bits after puncturing (only for convolutionally coded data, for other ECC schemes type 3 bits are the same as type 2 bits)
Type 4	Decorrelated and interleaved (re-ordered) type 3 bits

Type 4 bits are used for transmission over the RF. Incoming type 4 bits shall be decoded to type 1 bits, which are then used to extract all the frame fields.

3.4 Error correction coding schemes and bit type conversion

Two distinct *ECC*/*FEC* schemes are used for different parts of the transmission.

3.4.1 Link setup frame

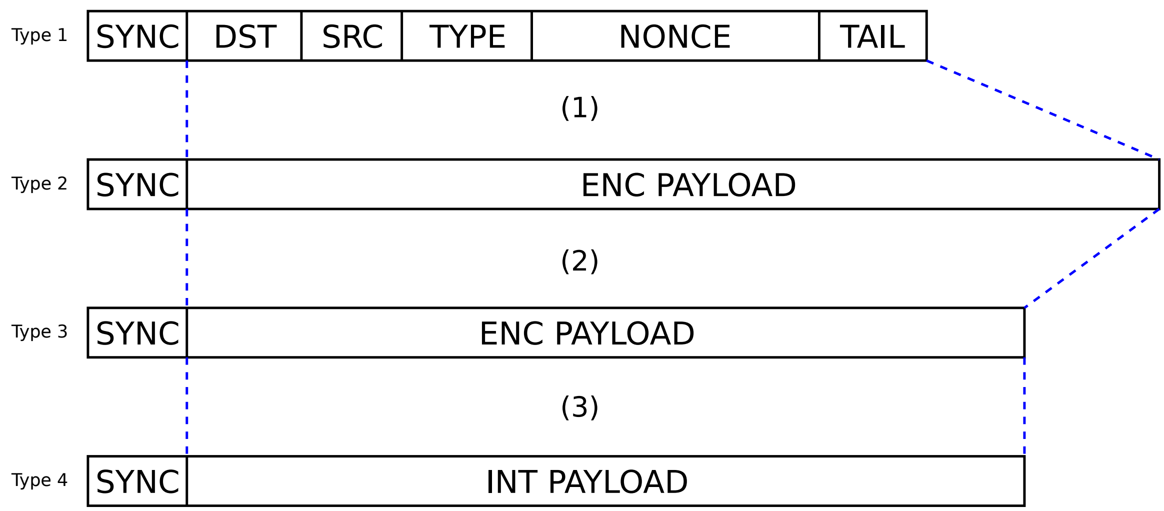

Fig. 3.2: ECC stages for the link setup frame

240 DST, SRC, TYPE, NONCE and CRC type 1 bits are convolutionally coded using rate 1/2 coder with constraint K=5. 4 tail bits are used to flush the encoder's state register, giving a total of 244 bits being encoded. Resulting 488 type 2 bits are retained for type 3 bits computation. Type 3 bits are computed by puncturing type 2 bits using a scheme shown in chapter 4.4. This results in 368 bits, which in conjunction with the synchronization burst gives 384 bits (384 bits / 9600bps = 40 ms).

Interleaving type 3 bits produce type 4 bits that are ready to be transmitted. Interleaving is used to combat error bursts.

3.4.2 Subsequent frames

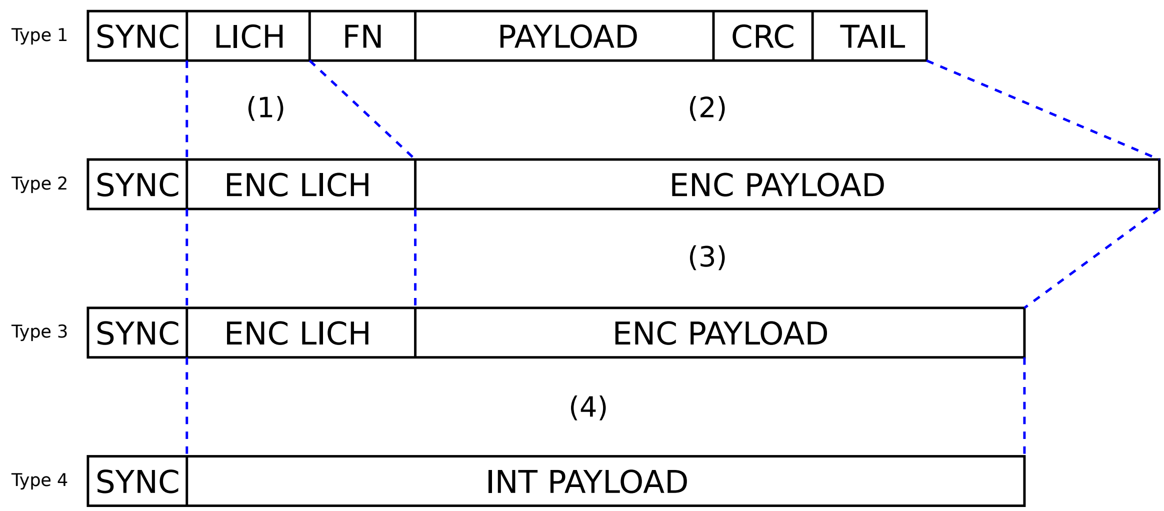

Fig. 3.3: ECC stages of subsequent frames

A 48-bit (type 1) chunk of LICH is partitioned into 4 12-bit parts and encoded using Golay (24, 12) code. This produces 96 encoded LICH bits of type 2.

FN, payload and CRC is 160 bits which are convolutionally encoded in a manner analogous to that of the link setup frame. A total of 164 bits is being encoded resulting in 328 type 2 bits. These bits are punctured to generate 272 type 3 bits.

96 type 2 bits of LICH are concatenated with 272 type 3 bits and re-ordered to form type 4 bits for transmission. This, along with 16-bit sync in the beginning of frame, gives a total of 384 bits

The LICH chunks allow for late listening and indepedent decoding to check destination address. The goal is to require less complexity to decode just the LICH and check if the full message should be decoded.

3.4.3 Golay (24,12)

The Golay (24,12) encoder uses the polynomial 0xC75 to generate the 11 check bits. The check bits and an overall parity bit are appended to the 12 bit data, resulting in a 24 bit encoded chunk.

$$G = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1 \tag{3.1}$$

The output of the Golay encoder looks like:

Data	Check bits	Parity
23-12 (12 bits)	11-1 (11 bits)	0 (1 bit)

Four of these 24-bit blocks are used to encode the LICH.

3.4.4 Convolutional encoder

The convolutional code shall encode the input bit sequence after appending 4 tail bits at the end of the sequence. Rate of the coder is R=½ with constraint length K=5 [NXDN]. The encoder diagram and generating polynomials are shown below

$$G_1(D) = 1 + D^3 + D^4 \tag{3.2}$$
$$G_2(D) = 1 + D + D^2 + D^4 \tag{3.3}$$

The output from the encoder must be read alternately.

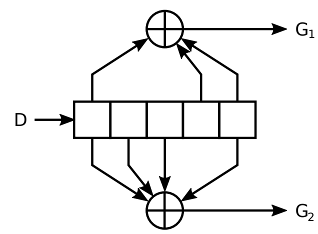

Fig. 3.4: Convolutional coder diagram

3.4.5 Code puncturing

Removing some of the bits from the convolutional coder's output is called code puncturing. The nominal coding rate of the encoder used in M17 is ½. This means the encoder outputs two bits for every bit of the input data stream. To get other (higher) coding rates, a puncturing scheme has to be used.

Two different puncturing schemes are used in M17 stream mode:

1. P_1 leaving 46 from 61 encoded bits
2. P_2 leaving 34 from 41 encoded bits

Scheme P_1 is used for the initial LICH link setup info, taking 488 bits of encoded data and selecting 368 bits. The $gcd(368, 488)$ is 8 which, when used to divide, leaves 46 and 61. A full puncture pattern requires the output be divisible by the number of encoding polynomials. For this case the full puncture matrix should have 122 entries with 92 of them being 1.

Scheme P_2 is for frames (excluding LICH chunks, which are coded differently). This takes 328 encoded bits and selects 272 of the bits. The $gcd(272, 328)$ is 8 which results in the 34 and 41 reduced ratio. The full matrix will have 82 entries with 68 being 1.

The matrices can be represented more concisely by duplicating a smaller matrix with a *flattening*.

$$S = \begin{bmatrix} a & \vec{r_1} & c \\ b & \vec{r_2} & X \end{bmatrix} \tag{3.4}$$

$$S_{full} = \begin{bmatrix} a & \vec{r_1} & c & b & \vec{r_2} \\ b & \vec{r_2} & a & \vec{r_1} & c \end{bmatrix} \tag{3.5}$$

The puncturing schemes are defined by their partial puncturing matrices: .. only:: latex

$$P_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & X \end{bmatrix} \tag{3.6}$$

$$P_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & X \end{bmatrix} \tag{3.7}$$

The complete linearized representations are:

Listing 3.1: linearized puncture patterns

```
P1 = [1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1]

P2 = [1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1]
```

3.4.6 Interleaving

For interleaving a Quadratic Permutation Polynomial (QPP) is used. The polynomial $\pi(x) = (45x + 92x^2) \mod 368$ is used for a 368 bit interleaving pattern [QPP]. See appendix Table 3.1 for pattern.

3.4.7 Data decorrelator

To avoid transmitting long sequences of constant symbols (e.g. 010101…), a simple algorithm is used. All 46 bytes of type 4 bits shall be XORed with a pseudorandom, predefined stream. The same algorithm has to be used for incoming bits at the receiver to get the original data stream. See Table 2.1 for sequence.

CHAPTER 4

Data Link Layer

The Data Link layer is split into two modes:

#. Packet mode: data are sent in small bursts, on the order of 100s to 1000s of bytes at a time, after which the physical layer stops sending data. eg: messages, beacons, etc. #. Stream mode: data are sent in a continuous stream for an indefinite amount of time, with no break in physical layer output, until the stream ends. eg: voice data, bulk data transfers, etc.

When the physical layer is idle (no RF being transmitted or received), the data link defaults to packet mode. ~~To switch to stream mode, a start stream packet (detailed later) is sent, immediately followed by the switch to stream mode; the Stream of data immediately follows the Start Stream packet without disabling the Physical layer. To switch out of Stream mode, the stream simply ends and returns the Physical layer to the idle state, and the Data Link defaults back to Packet mode.~~

As is the convention with networking protocols, all quantities larger than 8 bits are encoded in bigendian.

4.1 Stream Mode

In Stream Mode, an *indefinite* amount of payload data is sent continuously without breaks in the physical layer. The *stream* is broken up into parts, called *frames* to not confuse them with *packets* sent in packet mode. Frames contain payload data interleaved with frame signalling (similar to packets). Frame signalling is contained within the **Link Information Channel (LICH)**.

4.1.1 Sync Burst

All frames are preceded by a 16-bit *synchronization burst*.

- Link setup frames shall be preceded with 0x55F7.
- Stream frames shall be preceeded with 0xFF5D.
- Packet frames shall be preceeded with 0x75FF.

All syncwords are type 4 bits.

These sync words are based on Barker codes. The sequence 0xDF55 (symbols -3 +3 -3 -3 +3 +3 +3 +3) is reserved.

4.1.2 Link setup frame

First frame of the transmission contains full LSF data. It's called the **Link Setup Frame (LSF)**, and is not part of any superframes.

Table 4.1: Link setup frame fields

DST	48 bits	Destination address - Encoded callsign or a special number (eg. a group)
SRC	48 bits	Source address - Encoded callsign of the originator or a special number (eg. a group)
TYPE	16 bits	Information about the incoming data stream
NONCE	112 bits	Nonce for encryption
CRC	16 bits	CRC for the link setup data
TAIL	4 bits	Flushing bits for the convolutional encoder that do not carry any information

Table 4.2: Bitfields of type field

Bits	Meaning
0	Packet/stream indicator, 0=packet, 1=stream
1-2	Data type indicator, 01_2 =data (D), 10_2 =voice (V), 11_2 =V+D, 00_2 =reserved
3-4	Encryption type, 00_2 =none, 01_2 =AES, 10_2 =scrambling, 11_2 =other/reserved
5-6	Encryption subtype (meaning of values depends on encryption type)
7-15	Reserved (don't care)

The fields in Table 3 (except tail) form initial LSF. It contains all information needed to establish M17 link. Later in the transmission, the initial LSF is divided into 6 "chunks" and transmitted interleaved with data. The purpose of that is to allow late-joiners to receive the LICH at any point of the transmission. The process of collecting full LSF takes 6 frames or 6*40 ms = 240 ms. Four TAIL bits are needed for the convolutional coder to go back to state 0, so also the ending trellis position is known.

Voice coder rate is inferred from TYPE field, bits 1 and 2.

Table 4.3: Voice coder rates for different data type indicators

Data type indicator	Voice coder rate
00_2	none/reserved
01_2	no voice
10_2	3200 bps
11_2	1600 bps

4.1.3 Subsequent frames

Table 4.4: Fields for frames other than the link setup frame

LICH	48 bits	LSF chunk, one of 6
FN	16 bits	Frame number, starts from 0 and increments every frame to a max of 0x7fff where it will then wrap back to 0. High bit set indicates this frame is the last of the stream.
PAY-LOAD	128 bits	Payload/data, can contain arbitrary data
CRC	16 bits	This field contains 16-bit value used to check data integrity, see section 2.4 for details
TAIL	4 bits	Flushing bits for the convolutional encoder that don't carry any information

The most significant bit in the FN counter is used for transmission end signalling. When transmitting the last frame, it shall be set to 1 (one).

The payload is used so that earlier data in the voice stream is sent first. For mixed voice and data payloads, the voice data is stored first, then the data.

Table 4.5: LSF chunk structure

Bits	Content
0..39	40 bits of full LSF
40..42	A modulo 6 counter (LICH_CNT) for LICH re-assembly
43..47	5-bit Color Code (CC)

Table 4.6: Payload example 1

Codec2 encoded frame t + 0	Codec2 encoded frame t + 1

Table 4.7: Payload Example 2

Codec2 encoded frame t + 0	Mixed data t + 0

4.1.4 Superframes

Each frame contains a chunk of the LSF frame that was used to establish the stream. Frames are grouped into superframes, which is the group of 6 frames that contain everything needed to rebuild the original LSF packet, so that the user who starts listening in the middle of a stream (late-joiner) is eventually able to reconstruct the LSF message and understand how to receive the in-progress stream.

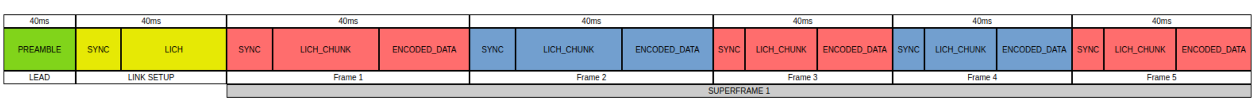

Fig. 4.1: Stream consisting of one superframe

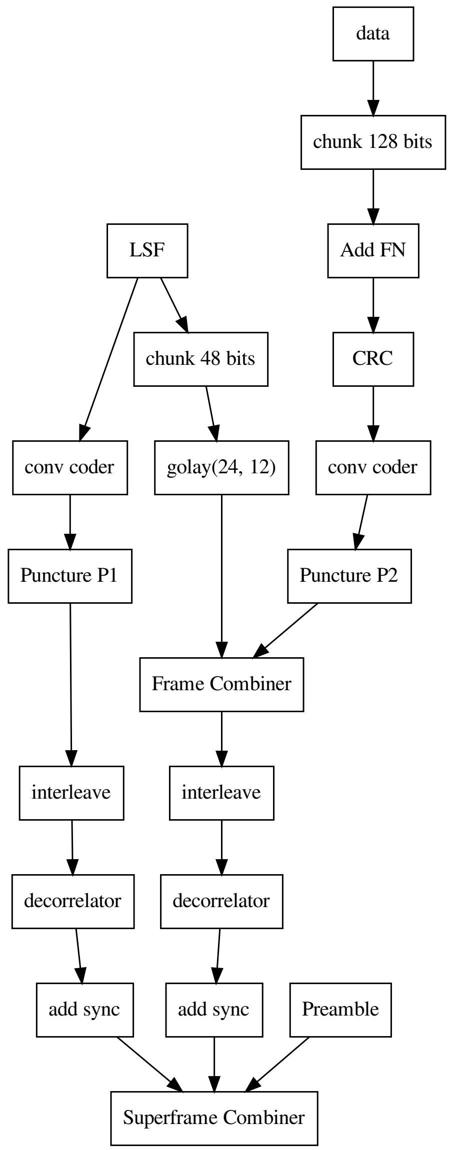

Fig. 4.2: An overview of the forward dataflow

4.1.5 CRC

M17 uses a non-standard version of 16-bit CRC with polynomial $x^{16} + x^{14} + x^{12} + x^{11} + x^8 + x^5 + x^4 + x^2 + 1$ or 0x5935 and initial value of 0xFFFF. This polynomial allows for detecting all errors up to hamming distance of 5 with payloads up to 241 bits[1], which is less than the amount of data in each frame.

As M17's native bit order is most significant bit first, neither the input nor the output of the CRC algorithm gets reflected.

The input to the CRC algorithm consists of the 16 bits of FN and 128 bits of payload, and then depending on whether the CRC is being computed or verified either 16 zero bits or the received CRC.

The test vectors in Table 6 are calculated by feeding the given message and then 16 zero bits to the CRC algorithm.

Table 4.8: CRC test vectors

Message	CRC output
(empty string)	0xFFFF
ASCII string "A"	0x206E
ASCII string "123456789"	0x772B
Bytes from 0x00 to 0xFF	0x1C31

4.2 Packet Mode

In *packet mode*, a finite amount of payload data (for example – text messages or application layer data) is wrapped with a packet, sent over the physical layer, and is completed when done. ~~Any acknowledgement or retransmission is done at the application layer.~~

4.2.1 Link Setup Frame

Packet mode uses the same link setup frame that has been defined for stream mode above. The packet/stream indicator is set to 0 in the type field.

Table 4.9: Bitfields of type field

Bits	Meaning
0	Packet/stream indicator, 0=packet, 1=stream
1-2	Data type indicator, 01_2 =raw (D), 10_2 =encapsulated (V), 11_2 =reserved, 00_2 =reserved
3-4	Encryption type, 00_2 =none, 01_2 =AES, 10_2 =scrambling, 11_2 =other/reserved
5-6	Encryption subtype (meaning of values depends on encryption type)
7-15	Reserved (don't care)

Raw packet frames have no packet type metadata associated with them. Encapsulated packet format is discussed in *Packet Superframes* in the Application Layer section. This provides data type information and is the preferred format for use on M17.

Currently the contents of the source and destination fields are arbitrary as no behavior is defined which depends on the content of these fields. The only requirement is that the content is base-40 encoded.

[1] https://users.ece.cmu.edu/~koopman/crc/ has this listed as 0xAC9A, which is the reversed reciprocal notation

4.2.2 Packet Format

M17 packet mode can transmit up to 798 bytes of payload data. It acheives a base throughput of 5kbps, and a net throughput of about 4.7kbps for the largest data payload, and over 3kbps for 100-byte payloads. (Net throughput takes into account preamble and link setup overhead.)

The packet superframe consists of 798 payload data bytes and a 2-byte CCITT CRC-16 checksum.

Table 4.10: Byte fields of packet superframe

Bytes	Meaning
1-798	Packet payload
2	CCITT CRC-16

Packet data is split into frames of 368 type 4 bits preceded by a packet-specific 16-bit sync word (0xFF5D). This is the same size frame used by stream mode.

The packet frame starts with a 210 byte frame of type 1 data. It is noteworthy that it does not terminate on a byte boundary.

The frame has 200 bits (25 bytes) of payload data, 6 bits of frame metadata, and 4 bits to flush the convolutional coder.

Table 4.11: Bit fields of packet frame

Bits	Meaning
0-199	Packet payload
1	EOF indicator
5	Frame/byte count
4	Flush bits for convolutional coder

The metadata field contains a 1 bit **end of frame** (**EOF**) indicator, and a 5-bit frame/byte counter.

The **EOF** bit is 1 only on the last frame. The **counter** field is used to indicate the frame number when **EOF** is 0, and the number of bytes in the last frame when **EOF** is 1. This encodes the exact packet size, up to 800 bytes, in a 6-bit field.

Table 4.12: Metadata field with EOF = 0

Bits	Meaning
0	Set to 0, Not end of frame
1-5	Frame number, 0..31

Table 4.13: Metadata field with EOF = 1

Bits	Meaning
0	Set to 1, End of frame
1-5	Number of bytes in frame, 1..25

Note that it is non-conforming to send a last frame with a length of 0 bytes.

4.2.3 Convolutional Coding

The entire frame is convolutionally coded, giving 420 bits of type 2 data. It is then punctured using a 7/8 puncture matrix (1,1,1,1,1,1,1,0) to give 368 type 3 bits. These are then interleaved and decorrelated to give 368 type 4 bits.

Table 4.14: Packet frame

Bits	Meaning
16 bits	Sync word 0xFF5D
368 bits	Payload

4.2.4 Carrier-sense Multiple Access

When sending packets, the sender is reponsible for ensuring the channel is clear before transmitting. CSMA is used to minimize collisions on a shared network. Specifically, P-persistent access is used. Each time slot is 40ms (one packet length) and the probability SHOULD default to 25%. In terms of the values used by the KISS protocol, these equate to a slot time of 4 and a P-persistence value of 63.

The benefit of this method is that it imposes no penalty on uncontested networks.

CHAPTER 5

Application Layer

PARTS 1 AND 2 REMOVED – will add this later.

5.1 Packet Superframes

Packet superframes are composed of a 1..n byte data type specifier, 0..797 bytes of payload data. The data type specifier is encoded in the same way as UTF-8. It provides efficient coding of common data types. And it can be extended to include a very large number of distinct packet data type codes.

The data type specifier can also be used as a protocol specifier. For example, the following protocol identifers are reserved in the M17 packet spec:

Table 5.1: Reserved Protocols

Identifer	Protocol
0x00	RAW
0x01	AX.25
0x02	APRS
0x03	6LoWPAN
0x04	IPv4
0x05	SMS
0x06	WinLink

The data type specifier is used to compute the CRC, along with the payload.

5.2 Encryption Types

Encryption is optional and disabled by default. The use of it is only allowed if local laws allow to doso.

5.2.1 Null Encryption

Encryption type = 00_2

No encryption is performed, payload is sent in clear text.

5.2.2 Scrambler

Encryption type = 01_2

Scrambling is an encryption by bit inversion using a bitwise exclusive-or (XOR) operation between bit sequence of data and pseudorandom bit sequence.

Encrypting bitstream is generated using a Fibonacci-topology Linear-Feedback Shift Register (LFSR). Three different LFSR sizes are available: 8, 16 and 24-bit. Each shift register has an associated polynomial. The polynomials are listed in Table 7. The LFSR is initialised with a seed value of the same length as the shift register. Seed value acts as an encryption key for the scrambler algorithm. Figures 5 to 8 show block diagrams of the algorithm

Table 5.2: LFSR scrambler polynomials

Encryption subtype	LFSR polynomial	Seed length	Sequence period
00_2	$x^8 + x^6 + x^5 + x^4 + 1$	8 bits	255
01_2	$x^{16} + x^{15} + x^{13} + x^4 + 1$	16 bits	65,535
10_2	$x^{24} + x^{23} + x^{22} + x^{17} + 1$	24 bits	16,777,215

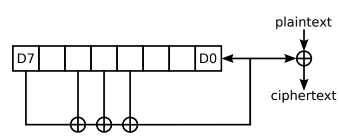

Fig. 5.1: 8-bit LFSR taps

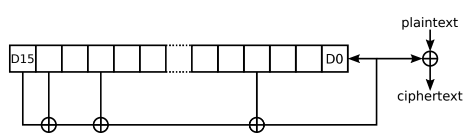

Fig. 5.2: 16-bit LFSR taps

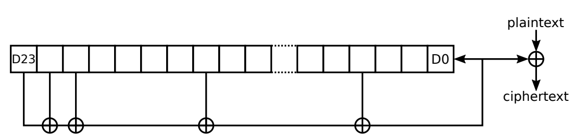

Fig. 5.3: 24-bit LFSR taps

5.2.3 Advanced Encryption Standard (AES)

Encryption type = 10_2

This method uses AES block cipher in counter (CTR) mode. 96-bit nonce value is extracted from the NONCE field, as the 96 most significant bits of it. The highest 16 bits of the counter are the remaining 16 bits of the NONCE field. FN field value is then used as the counter. The 16 bit frame counter and 40 ms frames can provide for over 20 minutes of streaming without rolling over the counter[1]. This method adapts 16-bit counter to the standard 32-bit CTR for the encryption. FN counter always start from 0 (zero).

The nonce value should be generated with a hardware random number generator or any other method of generating non-repeating values. Nonce values must be used only once. It is obvious that with a finite number of nonce bits, the probability of nonce collision approaches 1. We assume that the transmission is secure for 237 frames using a single key. It is recommended to change keys after that period.

Warning: In CTR mode, AES encryption is malleable [CTR] [CRYPTO]. That is, an attacker can change the contents of the encrypted message without decrypting it. This means that recipients of AES-encrypted data must not trust that the data is authentic. Users who require that received messages are proven to be exactly as-sent by the sender should add application-layer authentication, such as HMAC. In the future, use of a different mode, such as Galois/Counter Mode, could alleviate this issue [CRYPTO].

To combat replay attacks, a 32-bit timestamp shall be embedded into the NONCE field. The field structure is shown in Table 9. Timestamp is 32 LSB portion of the number of seconds that elapsed since the beginning of 1970-01-01, 00:00:00 UTC, minus leap seconds (a.k.a. “unix time”).

Table 5.3: NONCE field structure

TIMESTAMP	NONCE	CTR_HIGH
32	64	16

CTR_HIGH field initializes the highest 16 bits of the CTR, with the rest of the counter being equal to the FN counter.

[1] The effective capacity of the counter is 15 bits, as the MSB is used for transmission end signalling

APPENDIX A

Address Encoding

M17 uses 48 bits (6 bytes) long addresses. Callsigns (and other addresses) are encoded into these 6 bytes in the following ways:

- An address of 0 is invalid.
- Address values between 1 and 262143999999999 (which is $40^9 - 1$), up to 9 characters of text are encoded using base40, described below.
- Address values between 262144000000000 (40^9) and 281474976710654 ($2^{48} - 2$) are invalid
- An address of 0xFFFFFFFFFFFF is a broadcast. All stations should receive and listen to this message.

Table 1.1: Address scheme

Address Range	Category	Number of addresses	Remarks
0x000000000000	RESERVED	1	For future use
0x000000000001-0xee6b27ffffff	Unit ID	262143999999999	
0xee6b28000000-0xfffffffffffe	RESERVED	19330976710655	For future use
0xffffffffffff	Broadcast	1	Valid only for destination field

A.1 Callsign Encoding: base40

9 characters from an alphabet of 40 possible characters can be encoded into 48 bits, 6 bytes. The base40 alphabet is:

- 0: A space. Invalid characters will be replaced with this.
- 1-26: “A” through “Z”
- 27-36: “0” through “9”
- 37: “-” (hyphen)
- 38: “/” (slash)
- 39: “.” (dot)

Encoding is little endian. That is, the right most characters in the encoded string are the most significant bits in the resulting encoding.

A.1.1 Example code: encode_base40()

```
uint64_t encode_callsign_base40(const char *callsign) {
  uint64_t encoded = 0;
  for (const char *p = (callsign + strlen(callsign) - 1); p >= callsign; p-- ) {
    encoded *= 40;
    // If speed is more important than code space,
    // you can replace this with a lookup into a 256 byte array.
    if (*p >= 'A' && *p <= 'Z') // 1-26
      encoded += *p - 'A' + 1;
    else if (*p >= '0' && *p <= '9') // 27-36
      encoded += *p - '0' + 27;
    else if (*p == '-') // 37
      encoded += 37;
    // These are just place holders. If other characters make more sense,
    // change these. Be sure to change them in the decode array below too.
    else if (*p == '/') // 38
      encoded += 38;
    else if (*p == '.') // 39
      encoded += 39;
    else
      // Invalid character or space, represented by 0, decoded as a space.
      //encoded += 0;
  }
  return encoded;
}
```

A.1.2 Example code: decode_base40()

```
char *decode_callsign_base40(uint64_t encoded, char *callsign) {
  if (encoded >= 262144000000000) { // 40^9
    *callsign = 0;
    return callsign;
  }
  char *p = callsign;
  for (; encoded > 0; p++) {
    *p = " ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-/."[encoded % 40];
    encoded /= 40;
  }
  *p = 0;

  return callsign;
}
```

A.1.3 Why base40?

The longest commonly assigned callsign from the FCC is 6 characters. The minimum alphabet of A-Z, 0-9, and a "done" character mean the most compact encoding of an American callsign could be: $log2(37^6) = 31.26$ bits, or 4 bytes.

Some countries use longer callsigns, and the US sometimes issues longer special event callsigns. Also, we want to extend our callsigns (see below). So we want more than 6 characters. How many bits do we need to represent more characters:

Table 1.2: bits per characters

characters	bits	bytes
7	$log2(37^7) = 36.47$	5
8	$log2(37^8) = 41.67$	6
9	$log2(37^9) = 46.89$	6
10	$log2(37^{10}) = 52.09$	7

Of these, 9 characters into 6 bytes seems the sweet spot. Given 9 characters, how large can we make the alphabet without using more than 6 bytes?

Table 1.3: alphabet size vs bytes

alphabet size	bits	bytes
37	$log2(37^9) = 46.89$	6
38	$log2(38^9) = 47.23$	6
39	$log2(39^9) = 47.57$	6
40	$log2(40^9) = 47.90$	6
41	$log2(41^9) = 48.22$	7

Given this, 9 characters from an alphabet of 40 possible characters, makes maximal use of 6 bytes.

A.2 Callsign Formats

Government issued callsigns should be able to encode directly with no changes.

A.2.1 Multiple Stations

To allow for multiple stations by the same operator, we borrow the use of the '-' character from AX.25 and the SSID field. A callsign such as "KR6ZY-1" is considered a different station than "KR6ZY-2" or even "KR6ZY", but it is understood that these all belong to the same operator, "KR6ZY"

A.2.2 Temporary Modifiers

Similarly, suffixes are often added to callsign to indicate temporary changes of status, such as "KR6ZY/M" for a mobile station, or "KR6ZY/AE" to signify that I have Amateur Extra operating privileges even though the FCC database may not yet be updated. So the '/' is included in the base40 alphabet. The difference between '-' and '/' is that '-' are considered different stations, but '/' are NOT. They are considered to be a temporary modification to the same station.

A.2.3 Interoperability

It may be desirable to bridge information between M17 and other networks. The 9 character base40 encoding allows for this:

DMR

DMR unfortunately doesn't have a guaranteed single name space. Individual IDs are reasonably well recognized to be managed by https://www.radioid.net/database/search#! but Talk Groups are much less well managed. Talk Group XYZ on Brandmeister may be (and often is) different than Talk Group XYZ on a private cBridge system.

- DMR IDs are encoded as: D<number> eg: D3106728 for KR6ZY
- DMR Talk Groups are encoded by their network. Currently, the following networks are defined:
- Brandmeister: BM<number> eg: BM31075
- DMRPlus: DP<number> eg: DP262
- More networks to be defined here.

D-Star

D-Star reflectors have well defined names: REFxxxY which are encoded directly into base40.

Interoperability Challenges

We'll need to provide a source ID on the other network. Not sure how to do that, and it'll probably be unique for each network we want to interoperate with. Maybe write the DMR/BM gateway to automatically lookup a callsign in the DMR database and map it to a DMR ID? Just thinking out loud.

We will have to transcode CODEC2 to whatever the other network uses (pretty much AMBE of one flavor or another.) I'd be curious to see how that sounds.

APPENDIX B

Decorrelator sequence

Table 2.1: Decorrelator scrambling sequence

Seq. number	Value	Seq. number	Value
00	0xD6	23	0x6E
01	0xB5	24	0x68
02	0xE2	25	0x2F
03	0x30	26	0x35
04	0x82	27	0xDA
05	0xFF	28	0x14
06	0x84	29	0xEA
07	0x62	30	0xCD
08	0xBA	31	0x76
09	0x4E	32	0x19
10	0x96	33	0x8D
11	0x90	34	0xD5
12	0xD8	35	0x80
13	0x98	36	0xD1
14	0xDD	37	0x33
15	0x5D	38	0x87
16	0x0C	39	0x13
17	0xC8	40	0x57
18	0x52	41	0x18
19	0x43	42	0x2D
20	0x91	43	0x29
21	0x1D	44	0x78
22	0xF8	45	0xC3

APPENDIX C

Interleaving

Table 3.1: Interleaving table

input index	output ind	input index	output ind	input index	output ind	input index	output ind
0	0	92	92	184	184	276	276
1	137	93	229	185	321	277	45
2	90	94	182	186	274	278	366
3	227	95	319	187	43	279	135
4	180	96	272	188	364	280	88
5	317	97	41	189	133	281	225
6	270	98	362	190	86	282	178
7	39	99	131	191	223	283	315
8	360	100	84	192	176	284	268
9	129	101	221	193	313	285	37
10	82	102	174	194	266	286	358
11	219	103	311	195	35	287	127
12	172	104	264	196	356	288	80
13	309	105	33	197	125	289	217
14	262	106	354	198	78	290	170
15	31	107	123	199	215	291	307
16	352	108	76	200	168	292	260
17	121	109	213	201	305	293	29
18	74	110	166	202	258	294	350
19	211	111	303	203	27	295	119
20	164	112	256	204	348	296	72
21	301	113	25	205	117	297	209
22	254	114	346	206	70	298	162
23	23	115	115	207	207	299	299
24	344	116	68	208	160	300	252
25	113	117	205	209	297	301	21
26	66	118	158	210	250	302	342

Continued on next page

Table 3.1 – continued from previous page

input index	output ind	input index	output ind	input index	output ind	input index	output ind
27	203	119	295	211	19	303	111
28	156	120	248	212	340	304	64
29	293	121	17	213	109	305	201
30	246	122	338	214	62	306	154
31	15	123	107	215	199	307	291
32	336	124	60	216	152	308	244
33	105	125	197	217	289	309	13
34	58	126	150	218	242	310	334
35	195	127	287	219	11	311	103
36	148	128	240	220	332	312	56
37	285	129	9	221	101	313	193
38	238	130	330	222	54	314	146
39	7	131	99	223	191	315	283
40	328	132	52	224	144	316	236
41	97	133	189	225	281	317	5
42	50	134	142	226	234	318	326
43	187	135	279	227	3	319	95
44	140	136	232	228	324	320	48
45	277	137	1	229	93	321	185
46	230	138	322	230	46	322	138
47	367	139	91	231	183	323	275
48	320	140	44	232	136	324	228
49	89	141	181	233	273	325	365
50	42	142	134	234	226	326	318
51	179	143	271	235	363	327	87
52	132	144	224	236	316	328	40
53	269	145	361	237	85	329	177
54	222	146	314	238	38	330	130
55	359	147	83	239	175	331	267
56	312	148	36	240	128	332	220
57	81	149	173	241	265	333	357
58	34	150	126	242	218	334	310
59	171	151	263	243	355	335	79
60	124	152	216	244	308	336	32
61	261	153	353	245	77	337	169
62	214	154	306	246	30	338	122
63	351	155	75	247	167	339	259
64	304	156	28	248	120	340	212
65	73	157	165	249	257	341	349
66	26	158	118	250	210	342	302
67	163	159	255	251	347	343	71
68	116	160	208	252	300	344	24
69	253	161	345	253	69	345	161
70	206	162	298	254	22	346	114
71	343	163	67	255	159	347	251
72	296	164	20	256	112	348	204
73	65	165	157	257	249	349	341
74	18	166	110	258	202	350	294
75	155	167	247	259	339	351	63

Continued on next page

Table 3.1 – continued from previous page

input index	output ind	input index	output ind	input index	output ind	input index	output ind
76	108	168	200	260	292	352	16
77	245	169	337	261	61	353	153
78	198	170	290	262	14	354	106
79	335	171	59	263	151	355	243
80	288	172	12	264	104	356	196
81	57	173	149	265	241	357	333
82	10	174	102	266	194	358	286
83	147	175	239	267	331	359	55
84	100	176	192	268	284	360	8
85	237	177	329	269	53	361	145
86	190	178	282	270	6	362	98
87	327	179	51	271	143	363	235
88	280	180	4	272	96	364	188
89	49	181	141	273	233	365	325
90	2	182	94	274	186	366	278
91	139	183	231	275	323	367	47

APPENDIX D

M17 Internet Protocol (IP) Networking

Digital modes are commonly networked together through linked repeaters using IP networking.

For commercial protocols like DMR, this is meant for linking metropolitan and state networks together and allows for easy interoperability between radio users. Amateur Radio uses this capability for creating global communications networks for all imaginable purposes, and makes 'working the world' with an HT possible.

M17 is designed with this use in mind, and has native IP framing to support it.

In competing radio protocols, a repeater or some other RF to IP bridge is required for linking, leading to the use of hotspots (tiny simplex RF bridges).

The TR-9 and other M17 radios may support IP networking directly, such as through the ubiquitous ESP8266 chip or similar. This allows them to skip the RF link that current hotspot systems require, finally bringing to fruition the "Amateur digital radio is just VoIP" dystopian future we were all warned about.

D.1 Standard IP Framing

M17 over IP is big endian, consistent with other IP protocols. We have standardized on UDP port 17000, this port is recommended but not required. Later specifications may require this port.

Table 4.1: Internet frame fields

MAGIC	32 bits	Magic bytes 0x4d313720 ("M17 ")
StreamID (SID)	16 bits	Random bits, changed for each PTT or stream, but consistent from frame to frame within a stream
LICH	sizeof(LICH)*8 bits	A full LICH frame (dst, src, streamtype, nonce) as defined earlier
FN	16 bits	Frame number (exactly as would be transmitted as an RF stream frame, including the last frame indicator at (FN & 0x8000)
Payload	128 bits	Payload (exactly as would be transmitted in an RF stream frame)
CRC16	16 bits	CRC for the entire packet, as defined earlier (TODO: specific link)

The CRC checksum must be recomputed after modification or re-assembly of the packet, such as when translating from RF to IP framing.

D.2 Control Packets

Reflectors use a few different types of control frames, identified by their magic:

- *CONN* - Connect to a reflector
- *ACKN* - acknowledge connection
- *PING/PONG* - keepalives for the connection
- *DISC* - Disconnect (client->reflector or reflector->client)

D.2.1 CONN

Table 4.2: Bytes of a CONN packet

Bytes	Purpose
0-3	Magic - ASCII "CONN"
4-9	6-byte 'From' callsign including module in last character (e.g. "A1BCD D") encoded as per *Address Encoding*
10	Module to connect to - single ASCII byte A-Z

A client sends this to a reflector to initiate a connection. The reflector replies with ACKN on successful linking, or NACK on failure.

D.2.2 ACKN

Table 4.3: Bytes of ACKN packet

Bytes	Purpose
0-3	Magic - ASCII "ACKN"
4-9	6-byte callsign including module in last character (e.g. "A1BCD D") encoded as per *Address Encoding*

D.2.3 NACK

Table 4.4: Bytes of NACK packet

Bytes	Purpose
0-3	Magic - ASCII "NACK"

D.2.4 PONG

Table 4.5: Bytes of PONG packet

Bytes	Purpose
0-3	Magic - ASCII “PONG”
4-9	6-byte ‘From’ callsign including module in last character (e.g. “A1BCD D”) encoded as per *Address Encoding*

Upon receing a PING, the client replies with a PONG

D.2.5 DISC

Table 4.6: Bytes of DISC packet

Bytes	Purpose
0-3	Magic - ASCII “DISC”
4-9	6-byte ‘From’ callsign including module in last character (e.g. “A1BCD D”) encoded as per *Address Encoding*

Sent by either end to force a disconnection. Acknowledged with 4-byte packet “DISC” (without the callsign field)

Bibliography

[TETRA] Dunlop, John; Girma, Demessie; Irvine, James "Digital Mobile Communications and the TETRA System" Wiley 1999, ISBN: 9780471987925

[DMR] ETSI TS 102 361-1 V2.2.1 (2013-02): "Electromagnetic compatibility and Radio spectrum Matters (ERM); Digital Mobile Radio (DMR) Systems; Part 1: DMR Air Interface (AI) protocol" https://www.etsi.org/deliver/etsi_ts/102300_102399/10236101/02.02.01_60/ts_10236101v020201p.pdf

[ECC] Moreira, Jorge C.; Farrell, Patrick G. "Essentials of Error-Control Coding" Wiley 2006, ISBN: 9780470029206

[NXDN] NXDN Technical Specifications, Part 1: Air Interface; Sub-part A: Common Air Interface

[QPP] Trifina, Lucian, Daniela Tarniceriu, and Valeriu Munteanu. "Improved QPP Interleavers for LTE Standard." ISSCS 2011 - International Symposium on Signals, Circuits and Systems (2011): n. pag. Crossref. Web. https://arxiv.org/abs/1103.3794

[CTR] McGrew, David A. "Counter mode security: Analysis and recommendations." Cisco Systems, November 2, no. 4 (2002).

[CRYPTO] Rogaway, Phillip. "Evaluation of some blockcipher modes of operation." Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan (2011).

Index