M17 Protocol Specification

M17 Working Group

DRAFT

Authors: Mark KR6ZY

Jeroen PE1RXQ

Wojciech SP5WWP

Steve KC1AWV

Nikoloz SO3ALG

Formatted by Steve Miller, KC1AWV

Document History

Date	Comments
19 Dec 2019	Initial formatting – KC1AWV
20 Dec 2019	Author credits – KC1AWV
28 Dec 2019	Defined Physical layer parameters, some clean up – KR6ZY
30 Dec 2019	Added modulation description – SP5WWP
15 Jan 2020	Added scrambling description – SO3ALG, SP5WWP
30 May 2020	Removed a lot of outdated stuff, added error coding info – SP5WWP

Table of Contents

I. 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.

II. Physical Layer

1 4FSK generation

M17 standard uses 4FSK modulation running at 4800 symbols/s (9600bits/s) with a deviation index *h*=0.33 for transmission in 6.25kHz channel bandwidth. Channel spacing is 12.5kHz. The symbol stream is converted to a series of impulses which pass through a root-raised-cosine (α=0.5) shaping filter before frequency modulation at the transmitter and again after frequency demodulation at the receiver.

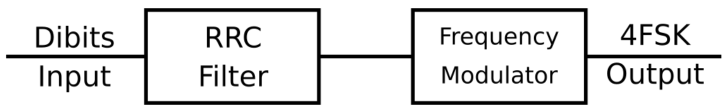


Figure 1: 4FSK modulator

The bit-to-symbol mapping is shown in ***Table 1.***

Table 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

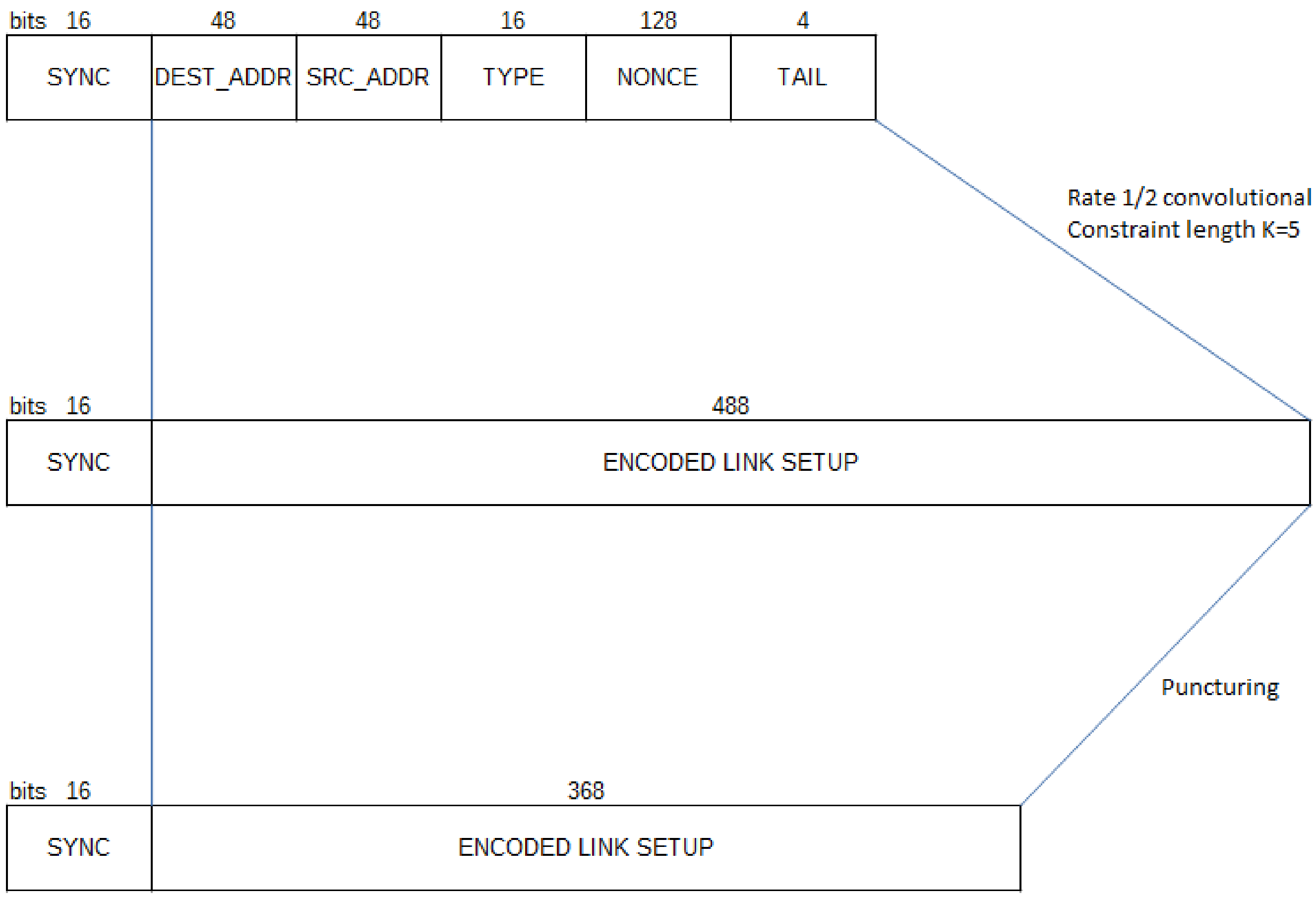
2 Preamble

The preamble is a repeating pattern of alternating +3 and -3 symbols, producing a 40 ms long, 2400 Hz tone.

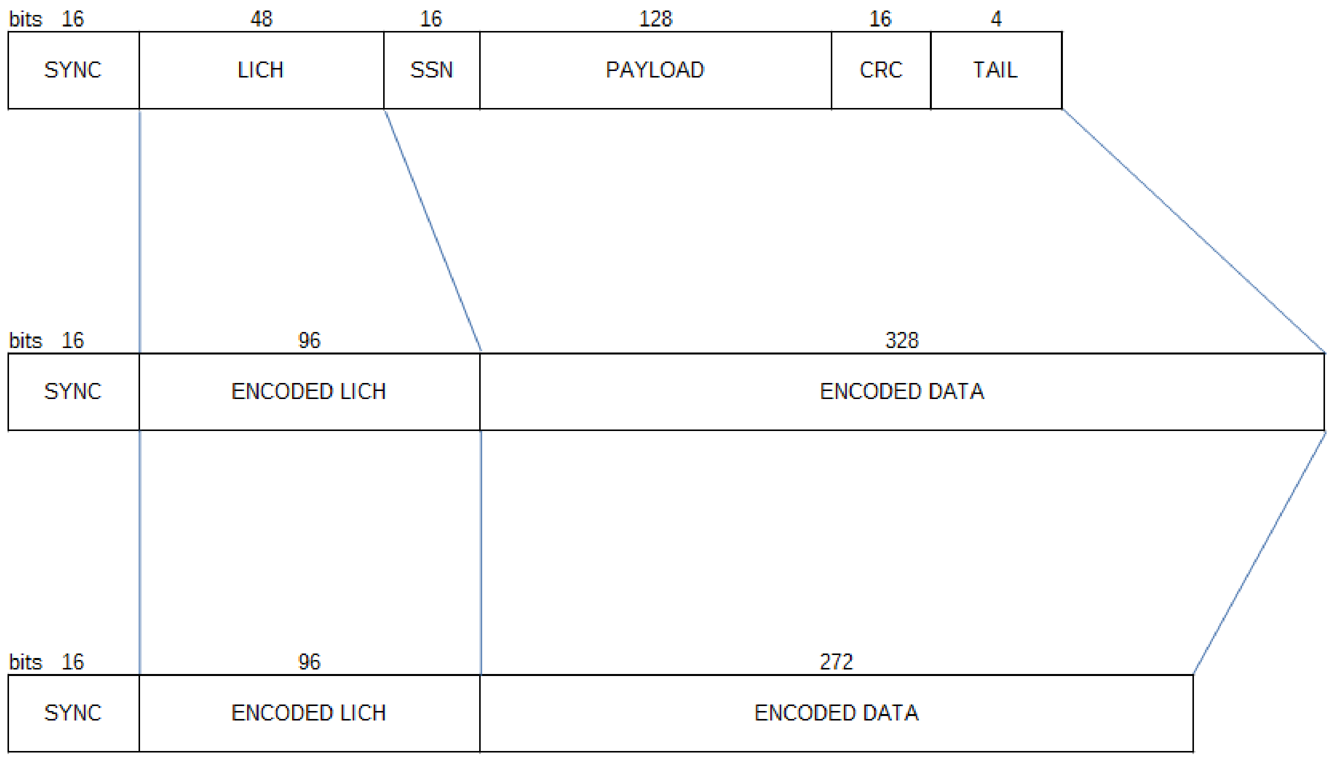
3 Error correction coding scheme

There are three distinct ECC schemes for different parts of the transmission.

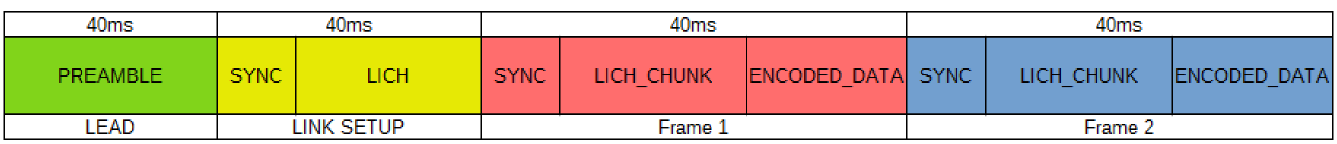
Error coding for the first frame of the stream:

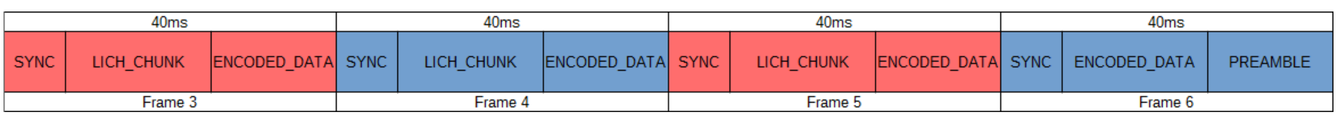


Error coding for the remaining frames, except the last one (2, 3, 4, 5):



The LICH channel is encoded using Golay (24,12) code.





TODO: add convolutional coding info (generating polynomials etc.). Replace these graphs with vectored versions. Add the last scheme. Add bits layering and interleaving info.

III. Data Link Layer

The Data Link layer is split into two modes:

1. ***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: Start Stream messages, beacons, etc.
2. ***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.~~

1 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 error correction is done at the application layer.~~

1.1 Packet Format

TODO More detail here about endianness, etc.

2 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).

Each frame contains a chunk of the LICH 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 LICH packet, so that the user who starts listening in the middle of a stream (*late-joiner*) is eventually able to reconstruct the LICH message and understand how to receive the in-progress stream.

2.1 Stream Frame Format

All stream frames are 48 bytes (384 bits) long. The leading frame is called a *preamble* and contains alternating 01 11 01 11… pattern. This is an equivalent of 40 milliseconds of 2400 Hz tone.

Frame 1 of the superframe contains full LICH data. It’s called the *link setup frame*.

Frames have the following format:

- SYNC: 2 bytes, 0x3243 (a portion of pi in hexidecimal)
- LICH chunk: TBF bytes
- Payload: TBF bytes

For a 9600 bps physical layer:

- A stream frame is sent every 40 ms.
- ~~This format gives 8400bps of payload throughput, an 87.5% efficiency.~~

2.2 Super Frames

Frames are grouped together into super frames, enough frames in a super frame to resend the entire StartStream packet, 4 bytes at a time, in the Preamble/Start Stream section of the frame, plus one more frame to send a Preamble. The Preamble signifies the beginning of a new super frame.

IV. Application Layer

PARTS 1 AND 2 REMOVED – will add this later.

3 Encryption Types

TODO Notes:

- Encryption uses CTR mode block ciphers and use the SSN as the counter. The 16 bit counter and 40ms frames can provide for over 43 minutes of streaming without rolling over the counter.

3.1 Null Encryption

Encryption Type = 0x00, Encryption Subtype = 0x00. No encryption is performed, payload is sent in clear text.

3.2 Scrambler

Encryption type = 0x21

Scrambling is an encryption algorithm that is a 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 ant 24-bit. Each shift register has an associated polynomial. The polynomials are listed in ***Table 2***. The LFSR is initialised with a *seed value* of the same length, as the shift register. Seed value acts as an encryption key. for Figures 2 to 4 show block diagrams of the algorithm.

Table 2: LFSR scrambler polynomials

Scrambling subtype	LFSR polynomial	Seed length	Sequence period
0x00	$x^8+x^6+x^5+x^4+1$	8-bit	255
0x01	$x^{16}+x^{15}+x^{13}+x^4+1$	16-bit	65,535
0x02	$x^{24}+x^{23}+x^{22}+x^{17}+1$	24-bit	16,777,215

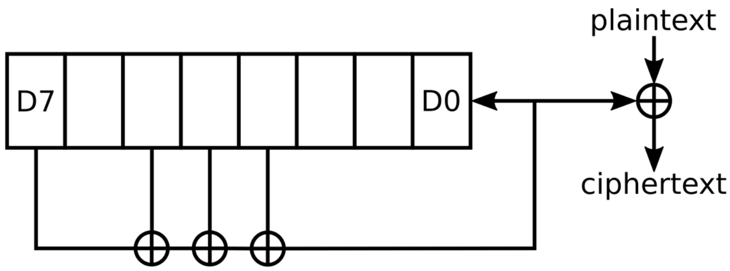


Figure 2: 8-bit LFSR taps

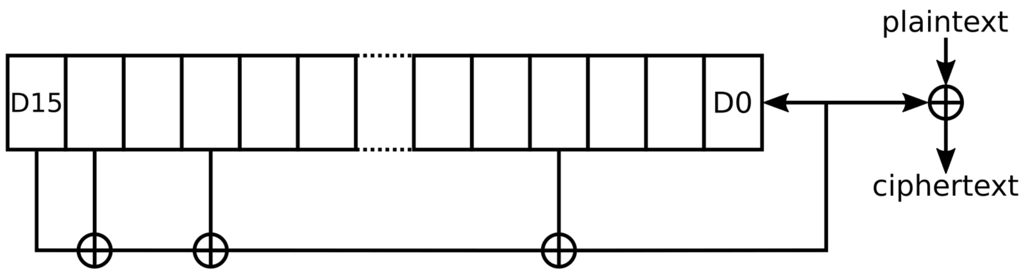


Figure 3: 16-bit LFSR taps

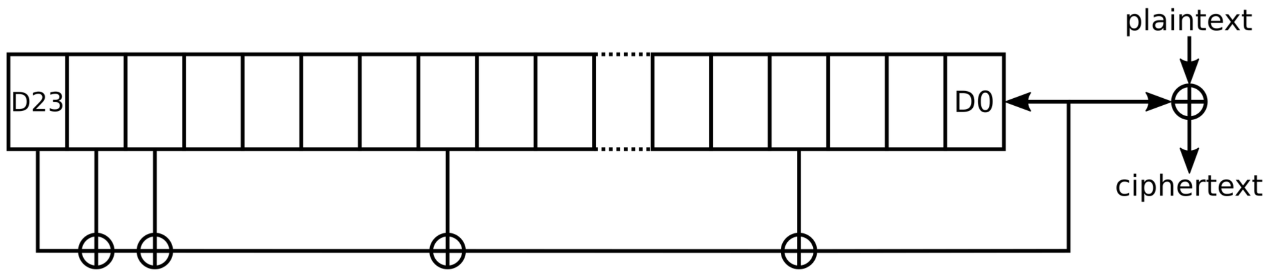


Figure 4: 24-bit LFSR taps

VI. 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.
  - **TODO** Do we want to use zero as a flag value of some kind?
- 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
  - **TODO** Can we think of something to do with these 19330976710654 addresses?
- An address of 0xFFFFFFFFFFFF is a broadcast. All stations should receive and listen to this message.

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: An invalid character, something not in the alphabet was provided.
- 1-26: 'A' through 'Z'
- 27-36: '0' through '9'
- 37: '-'
- 38: '/'
- 39: TBD

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

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, represented by 0.
      //encoded += 0;
      ;
  }
  return encoded;
}
```

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 = "xABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-/."[encoded % 40];
        encoded /= 40;
    }
    *p = 0;
    return callsign;
}
```

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: $\log_2(37^6)$=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:

- 7 characters: log2(37^7)=36.47 bits, 5 bytes
- 8 characters: log2(37^8)=41.67 bits, 6 bytes
- 9 characters: log2(37^9)=46.89 bits, 6 bytes
- 10 characters: log2(37^10)=52.09 bits, 7 bytes.

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?

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

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

2 Callsign Formats

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

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".

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. **TODO** I'm not sure what impact this actually has.

2.3 Interoperability

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

TODO Define more interoperability standards here. System Fusion? P25? IRLP? AllStar?

2.3.1 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`
    - More networks to be defined here.

2.3.2 D-Star

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

TODO Individuals? Just callsigns?

2.3.3 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.