The digimorse Communications Protocol

Matt Gumbley, MØCUV

August 11, 2022

This document introduces an experimental alternative to amateur radio's traditional way of transmitting Morse code. It describes the rationale for, and the design of the digimorse communications protocol. The design is explained in the context of general digital communications systems, with discussions of the choices made at all stages of its implementation in the transceiver software.

The digimorse communications protocol exists to bring together two aspects of amateur radio that I particularly enjoy, to see if the benefits of one aspect might improve my enjoyment of the other. Namely, to see if digital encoding and error correction[1] might be used to provide a significant coding gain and a removal of noise from Morse Code - our historic mode of communication.

[1] Such as found in the WSJT-X weak-signal mode software by Joe Taylor K1JT et al.

Other features of modern digital amateur radio that I aim to bring to Morse operation with digimorse include:

- a waterfall display
- active station information
- meaningful signal reports

Waterfall displays are now standard features on modern SDR[2] transceivers such as the Yaesu FTDX10, and in software such as WSJT-X[3]. They graphically show the presence of any signals - and information about their senders - across the receiver's audio bandwidth. The decode window of WSJT-X shows the content of the decoded messages that have been received in recent receive cycles. With this display it is easy to see who is currently active on the band, how strong their signals are, and their country/locator square. This gives an excellent overview of propagation conditions and band activity. Signal reports are automatically derived from the incoming signal strength, and are therefore meaningful – rather than the typical 'you are 5 by 9... what was your name and callsign again?'.

[2] Software Defined Radio

[3] Steve Franke K9AN, Bill Somerville G4WJS (SK), and Joe Taylor K1JT. The FT4 and FT8 communications protocols. QEX (ARRL), July/August 2020

With digimorse, since the Morse code you send is encoded and decoded digitally, if the receiving station achieves a successful decode, the code will be played back exactly as it was sent, with no noise.

In amateur radio, Morse code is traditionally transmitted using a pure sinusoidal radio wave, oscillating at a fixed frequency. This gives rise to the name and acronym it is more commonly known as: Continuous Wave, or CW. This method of transmitting Morse originated around 1913, with the invention of the vacuum tube electronic oscillator by Edwin Armstrong and Alexander Meissner[4]. Prior to this, Morse was transmitted using primitive spark gap transmitters, which created much electromagnetic interference, and had wide bandwidth. They were outlawed in 1934. The bandwidth of a CW signal is very low, compared to the systems it replaced[5].

[4] See Wikipedia, 'Continuous wave'

[5] Mark Amos W8JI. An intuitive explanation of cw bandwidth. URL `http://www.w8ji.com/cw_bandwidth_described.htm`

Perhaps surprisingly, CW, as a method of transmitting Morse code using radio, has not changed significantly since then. In the intervening years, great improvements to receiver technology have been made: receiver sensitivity and selectivity have improved; filtering and digital signal processing techniques have been applied to try to reduce or remove the noise that plagues reception.

However, despite these advances, noise and fading effects introduced by ionospheric reflection still impact our ability to receive and decode Morse, as received using CW signals.

As an analogy, digimorse intends to do to Morse code what compact discs did for vinyl recordings. It's an upgrade for CW.

This paper introduces digimorse in the context of modern digital communications protocols for the radio amateur.

Contents

Overview of the digimorse transceiver

A VIEW OF THE FUNCTIONS OF THE DIGIMORSE TRANSCEIVER, to be explored more fully in later sections.

Transmission

The digimorse software uses real, hand-sent Morse Code - not typed, computer-generated 'perfect' Morse - but keyed on a straight key, or paddle[6]. The key or paddle is attached to the computer via an Arduino-based USB interface which samples your keying, sending a stream of millisecond-precise key up/down durations to the computer. In addition to sampled keying information, several other 'metadata' items are included in the outgoing stream of frames – callsign, location, keying speed and power. This stream is then compressed, protected by a CRC[7] error-detection code, then enhanced with an LDPC[8] error-correction code[9] - similar to that used in FT4/FT8. These measures mitigate the effects of noise and other interference during receive, thereby improving the ability of the software to correctly decode extremely weak signals. These blocks of data are then prefixed with a Costas array[10], to allow the receiver to precisely filter the frame stream from the rest of the received audio. The complete stream is then modulated into a narrow bandwidth of tones, taking care to minimise splatter. Many such transmissions can occur simultaneously within a transceiver's 3kHz SSB bandwidth. The resultant audio is then transmitted.

[6] Only straight keys are currently supported.

[7] Cyclic Redundancy Check.

[8] Low-Density Parity-Check

[9] Robert G. Gallager. Low-Density Parity-Check Codes. IRE Transactions on Information Theory, 1962

[10] Mike Hasselbeck WB2FKO. Synchronization in FT8, July 2019. URL `http://www.sportscliche.com/wb2fko/FT8sync.pdf`

Reception

The receiver detects all narrow-bandwidth digimorse signals across its 3kHz incoming audio bandwidth by searching for their initial Costas arrays. After demodulation, error detection and correction is applied, and if the stream is valid, the detected metadata frames are displayed above each stream of keying information so that you can easily see who is transmitting, from where, how strong their signals are, and how fast they are keying. Given this scrolling display of decoded streams, you may select individual signals, or apply a bandpass filter across part of the receive bandwidth. The result is that the single selected signal - or multiple filtered signals - are then decoded, and reconstituted into high-quality Morse code, with no noise. Each stream is located at some audio offset from the start of the 3kHz receiver bandwidth, and this offset is used as the tone for each signal. The inclusion of LDPC error correction information means that even if your RF signal is very weak at the receiver, with high levels of noise present, then some or all of your transmission should be decodable, giving a considerable coding gain over CW [11].

[11] Kazimierz "Kay" Siwiak KE4PT and Bruce Pontius NØADL. How much "punch" can you get from diferent modes? QST (ARRL), December 2013

Digital Communications Systems

THE OVERVIEW PRESENTED ABOVE WILL NOW BE EXPANDED UPON by reviewing a generic digital communications system, and then explaining how each of its building blocks is implemented in digimorse.

Claude Shannon presented a high-level variant of the following diagram in his seminal paper "A Mathematical Theory of Communication"[12]. The diagram shown below is a modern enhancement for digital communication systems.

[12] Claude Shannon. A mathematical theory of communication. The Bell System Technical Journal, July/October 1948

All of the modern digital communication systems you might use adhere to this same basic structure.

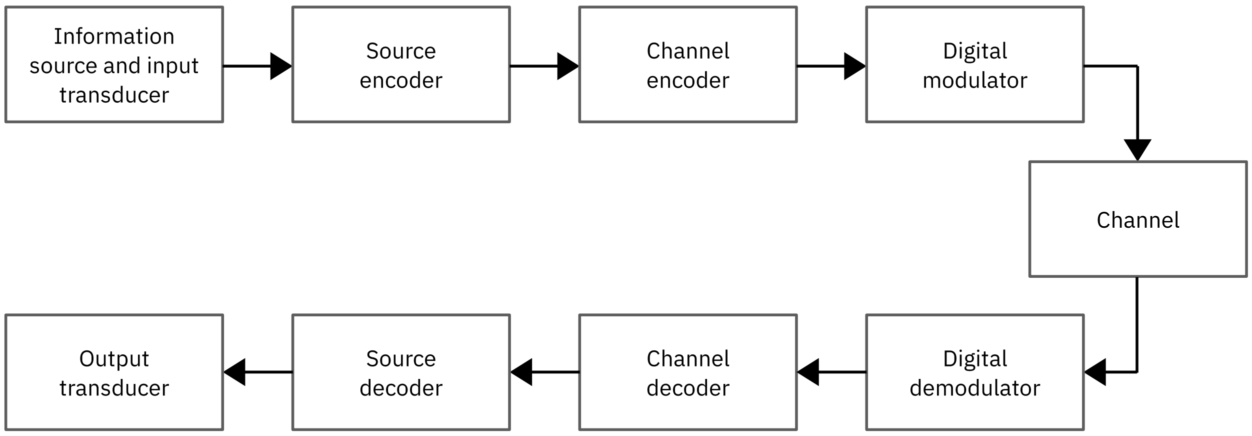

Figure 1: Basic elements of a digital communications system

The diagram starts at the top-left, and illustrates the path taken by some information (a message) from its source, to its eventual destination. The source could be analogue or digital, and is encoded into some digital form, such that it can be expressed in as few binary digits as possible. The output of the source encoder should exhibit as little redundancy as possible; it should be as compressed as possible.

The resulting information sequence is then fed to the channel encoder, which is tasked with preparing it for its journey through the channel. This is achieved by adding some redundancy back into the information, but in a structured manner, such that it can be used by the receiver to overcome the effects of noise and interference presented by the transmission over the channel. The reliability of decoding during reception is increased by this added redundancy.

The binary output of the channel encoder must be converted into a form suitable for transmission over the channel: this is the function of the digital modulator, which typically maps binary data into electrical signal waveforms.

The channel is the physical medium through which the signal is sent from transmitter to receiver. In amateur radio, this is the atmosphere, either a transmission via ground wave or sky wave. In either case, the signal is subject to interference, and is corrupted randomly by noise generated by other electronic devices, atmospheric noise (e.g. lightning discharges), human-generated noise such as car ignition noise or other switching. It can also be subject to various types of fading, multipath reflection, doppler effects, etc.

discuss digital demodulator discuss channel decoder discuss source decoder discuss output transducer

Starting with Morse code, I will now discuss each of these building blocks in turn, and their design and implementation in digimorse.

digimorse as a Digital Communications System

Before taking several deep dives into the building blocks, let's zoom out a little further, and look at the whole of digimorse - as there are additional parts that don't appear in the previous description of generic digital communications systems.

full block diagram goes here

Information Source and Input Transducer

Morse key and Arduino, keying protocol, USB Serial, protocol decoder, keying event stream. how the protocol decoder is wired to other block diagram elements. more, obvs

Source Encoder

Given a stream of keying information from the input layer, this must be encoded into a compressed form to remove redundancy, ensuring that the minimum number of bits are used (since the transmission of bits takes time; fewer bits are better).

The output of the source encoder is a fixed-length binary block of 112 bits. Keying and other metadata frames are appended to this block until:

See 'Block size', later

- It is full.
- Until the frame being appended will not fit at the end, causing the block to be filled with a padding frame. The frame will then be stored at the start of the next block.

The block is then sent on to the next major component in the transceiver, the channel encoder. The need for a fixed-length block is due to the channel encoder, whose error-correction algorithm requires it.

The full set of frame types that may be incorporated in the block is given in the following table on page 7.

The XXX|N notation means that the XXX value is encoded as an N-bit unsigned binary number.

When a frame has been added to a block, if there are fewer than four bits remaining, they are assumed to be padding, so the frame is emitted, and a new frame started.

In addition to the keying duration information, the source encoder must also encode the current speed in WPM, and the polarity of the key at the start of the frame. This ensures that in the event of

4-Bit Frame Id	Description	Encoding
0000	Padding to end of block	
0001	WPM/Polarity	WPM|6; Polarity|1
0010	MD Callsign	Callsign|28
0011	MD Callsign hash	Hash|22
0100	MD 4-Character Locator	C15
0101	MD Power	xxx
0110	Keying (Perfect dit)	
0111	Keying (Perfect dah)	
1000	Keying (Perfect wordgap)	
1001	Keying (End)	
1010	Keying (Delta dit)	See page 8
1011	Keying (Delta dah)	
1100	Keying (Delta wordgap)	
1101	Keying (Naïve)	See page 8
1110	unused	
1111	Extension	To be considered later...

Table 1: Frame types and their data, encoded into blocks by the source encoder.

a receiver being unable to correctly decode a frame, it will be able to 'pick up' the stream when it next has a successful decode. The WPM of the frame is sent in each frame[13] to ensure that the compressed keying duration data is correctly interpreted: it is expressed as a delta against the ideal timings - see page 8.

[13] In each frame that contains keying information. Most would.

Callsign and locator encoding are performed using the same mechanism as in JT65.[14]

[14] Thomas Clark W3IWI (SK) and Phil Karn KA9Q. Eme 2000: Applying modern communications technologies to weak signal amateur operations. Proc. Central States VHF Society, July 1996. URL `http://www.ka9q.net/papers/eme-2000.ps.gz`

The source encoder contains a rudimentary 'CQ detector' that detects when CQ has been sent at the start of a transmission. When this finds a CQ, it forces the source encoder to start embedding metadata frames. The current block will contain the Callsign frame; the next block will contain the 4-Character Locator frame; the next block will contain the Power frame.

Also if a Callsign frame has not been embedded in a block for 15 minutes, one will be included in the current block.[15]

[15] This timed callsign does not cause the next block to contain the 4-Character Locator as described in the previous paragraph.

This mechanism ensures that all metadata is sent close to the start of a CQ, spread across subsequent blocks[16] so that receiving users decode the metadata of all stations calling CQ. It also ensures compliance with regulations that require a station identify itself frequently.

[16] So as to prevent 'starvation' of keying information in a single block.

If a block does not contain a Callsign, then it will contain a Callsign Hash, which is a much smaller length frame than a Callsign. Either of these can be used in the receiver to enable the decoding and playback of a single station's transmission. By selecting the station's metadata, it is possible to filter out all other keying streams. This could be done by choosing a narrow bandwidth filtering range, but that would not reject another station whose offset is very close to the DX station. By including the Callsign or Callsign Hash in each block, the stream can be identified precisely.[17]

[17] Hash collisions notwithstanding!

A block will only contain one metadata frame (as indicated by MD in the table above).

If the operator has perfect rhythm and timing, and their dits, dahs and wordgaps are precisely the correct duration for the current WPM setting, then the 'Perfect dit/dah/wordgap' frames will be used, as these require no further data than their 4-bit frame id.

This would be the case if digimorse eventually accommodates a machine-generated stream of Morse, perhaps by entering text on-screen, or by the use of macro buttons. This could also happen when the USB interface is enhanced to incorporate a keyer.

I'm not in favour of making digimorse highly automated - the rationale of the project is to enhance the human enjoyment of Morse, not replace human operation by computer operation. You could use PSK31, RTTY, FT8 etc. for that.

For the rest of us mortal operators with our shaky fists and dodgy timing, digimorse will try to record and reproduce our variations as precisely as it can, given its millisecond resolution.

The digimorse transceiver accommodates keying speeds of 5 to 60 words per minute. There are three major timing elements in Morse: the dit, the dah and the wordgap.

A clear description of the elements and how they are used to form words is given by Michael Maynard K4ICY, illustrating the 'standard word' PARIS, used for calculating keying speed in words-per-minute.

Michael A. Maynard K4ICY. YOU Can Learn Morse Code & Operate on CW! URL `http://www.k4icy.com/cw.html`

A table showing the durations in milliseconds for these three elements, for all speeds in this range is given in the appendix on page 15. The longest element that could be transmitted is the 5WPM wordgap at 1680ms; the shortest is the 60WPM dit at 20ms. An operator could pause for longer than 1680ms of course, until the keyer times out and sends an End signal which terminates transmission.

This timeout can be configured in the keyer up to 3000ms i.e. three seconds.

How should this range of timings be encoded in the most compact form?

Naïve encoding

A number in the range 0 to n can be represented in $1 + \lfloor \log_2(n) \rfloor$ bits. The longest duration of 1680ms could be expressed as a binary field of 11 bits, as could any value up to 2047. This naïve approach to encoding could work, but would be inefficient, as will be shown next.

Do The Simplest Thing That Could Possibly Work

Delta encoding based on code speed

I took an example of a typical Morse QSO[18] and keyed it into digimorse using the KEYERDIAG option, which records your keying into a .CSV file containing MARK/SPACE duration data, and plays a sidetone as you operate. The resulting data is shown as a histogram in Figure 2 on page 9. Several aspects of this are worth pointing out:

[18] Hans Summers G0UPL. Beginner's CW QSO. URL `https://qrp-labs.com/qcx/cwqso.html`

- The modal value in the data is 72ms, so taking this as the duration of a dit, this suggests (from Appendix A) a speed between 16 and 17 WPM - say 16. This gives a dah of 225ms and a wordgap of 525ms. These ideal durations are shown with the three vertical bars.
- dit is far more common than dah or wordgap.
- There is no large spike for dah - the peak nearby is at a longer duration than the ideal, so my dah is slow. There is no spike at all for wordgap - just a smattering of points - indicative of the relative frequency of word gaps in the text.

- The points around these longer ideal timings are not as pronounced as they are for dit - suggesting I have a wide variance in my keying of the longer elements, but I'm more accurate with dit.

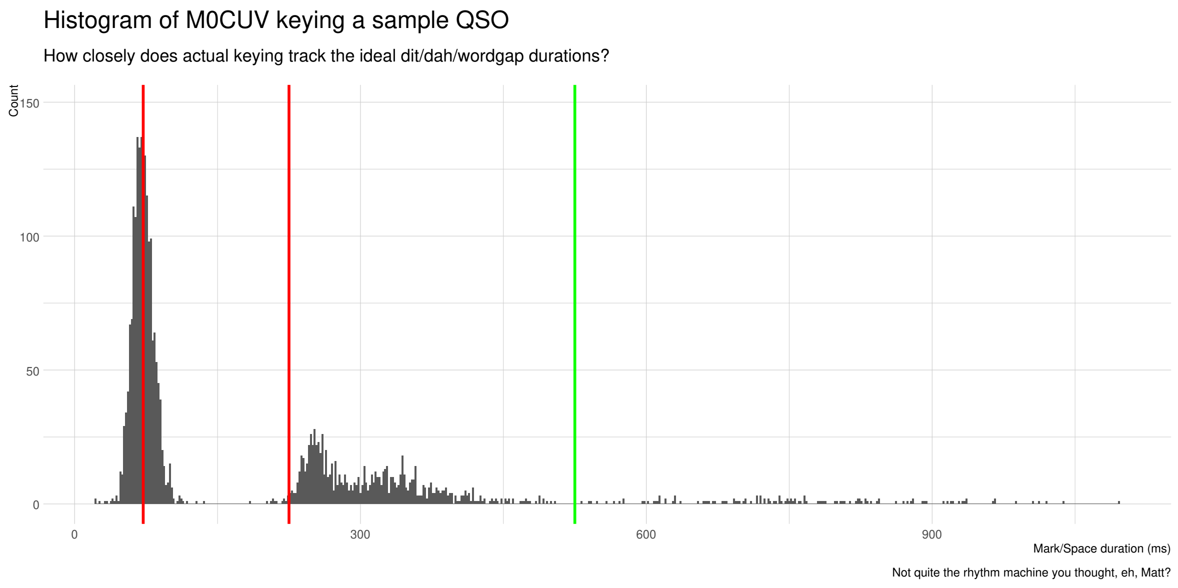

Figure 2: Keying of a sample QSO

To reduce the number of bits required to encode keying, determine whether the duration is closest to dit, dah or wordgap for the current WPM setting, then take the difference between this ideal duration and the duration under consideration.

As an example, the first durations in the sample QSO (the 'C' of the CQ) are:

Table 2: Delta encoding of Morse elements.

Mark/Space	Duration	Characterised as	Delta
MARK	210	dah	225 - 210 = 15
SPACE	72	(dit)	72 - 72 = 0 (Perfect)
MARK	73	dit	72 - 73 = -1
SPACE	64	(dit)	72 - 64 = 8
MARK	250	dah	225 - 250 = -25
SPACE	65	(dit)	72 - 65 = 7
MARK	61	dit	72 - 61 = 11
SPACE	298	(dah)	225 - 298 = -73

-73 could be encoded in 6 bits plus one sign bit. However a mechanism is required to indicate how many bits are used to encode the value. There are many such variable length encodings, and none are ideal for this task - they rely on embedding a lookup table of durations $\rightarrow$ code-words in each frame, then denoting the durations by their code words. Durations that occur with greater probability are allocated a shorter code word.

Echoing the design of Morse code itself, where more frequent letters have shorter Morse representations: consider E T A O I N S R H D L U etc.

To determine how many bits might be required to send the delta values, consider the ideal durations of dit, dah and wordgap at the slowest speed supported, 5 WPM. A dit is 240ms, with a delta of ± 240. A dah is 720ms with a delta of $[-240 .. + 480]$. A wordgap is 1680ms with a delta of $[-480 .. + 367]$.

The range of dah is [midpoint between dit and dah .. midpoint between dah and wordgap].

The upper bound of wordgap's delta is actually +480, but wordgap+367 = 2047 which fits in 11 bits, as noted earlier.

$\log_2(240) = 7.9$ and $\log_2(480) = 8.9$ suggesting that the dit range could be held in 8 bits, and the dah and wordgap ranges could be held in 9 bits - with the addition of a sign bit.

This is at the slowest speed supported. At higher speeds, the delta ranges are different.

At 35 WPM, the dit range is ± 120, and $\log_2(120) = 6.9$ which would require 7 + 1 sign bit. The dah range is $[-120 .. + 240]$ requiring 8 + 1 sign bits. The wordgap range is also ± 240.

The highest speed of 60 WPM has a dit range of ± 20, a dah and wordgap range of ± 40. $\log_2(20) = 4.3$ (requiring 5 + 1 bits) and $\log_2(40) = 5.3$ (requiring 6 + 1 bits).

So a delta could be encoded as a variable-length field between 5 and 9 bits plus 1 sign bit.

A little better than the 11 bits of the naïve encoding.

There would need to be a mechanism to indicate how many bits are used to encode the delta - although since every block contains a WPM/Polarity frame, the speed could be used to derive the expected number of bits for each encoded duration's delta:

WPM range	dit range bits	dah range bits	wordgap range bits
5 .. 9	8	8 .. 9	9
10 .. 18	7	7 .. 8	8
19 .. 37	6	6 .. 7	7
38 .. 60	5	5 .. 6	6

Table 3: Delta encoding bit ranges for various speed ranges. These ranges do not include the sign bit.

the graph of ideal durations, and the triangular delta ranges for each element should go here to help explain the table above.

The Keying (Delta) frames would be used to encode durations that fit within these ranges. Of course it's possible to send elements whose durations are well outside these ranges, and for these, the Keying (Naïve) frame would be used.

The approach is therefore:

- If the duration is perfect (possibly $\pm$ a small number of milliseconds), use the Keying (Perfect) frames.
- If it is within the ranges given in the discussion above, use the Keying (Delta) frames.
- Otherwise use the Keying (Naïve) frame - which does not distinguish between dit, dah and wordgap: it's not relative to any of them.

Block size

How large should the digimorse block be? Each block will contain a WPM/Polarity frame, followed by a possible Metadata frame, then several Keying frames, which as discussed in the previous section, can be various lengths, depending on the chosen keying speed, and the operator's accuracy. Ideally, we want to reduce the use of wasted Padding to end of block frames.

To recap, WPM/Polarity frame are 4+7 bits; Keying (Perfect) frames are 4 bits; Keying (Naïve) are 4+11; Keying (Delta) are between 4+5 and 4+9 bits - let's say an average of 4+7 bits.

Metadata encoding has not yet been incorporated into the code, so I'm not considering that in the block size.

So how many average Keying frames should each block contain? There should be sufficient to ensure that the overhead of WPM/Polarity, and the extra bits added due to error correction and synchronisation do not dwarf the actual keying. Without doing any exhaustive simulation of this, I'm going to choose "two typical Morse characters" to be encodable in a block. Of the A-Z 0-9 , / ? AR . set of characters, 14 have 5 elements, 12 have 4, 8 have 3. So, two of the most common 5 element characters, which would require $(2 * 5 - 1)$ frames, with an average Keying frame length of 4+7.

This leads to $(4 + 7) + ((2 * 5 - 1) * (4 * 7))$ bits, or 110 bits, which, rounded up to a multiple of 8 bits gives a block size of 112 bits.

This block size is up for discussion!

Channel Encoder

Error Detection

The block is then checksummed by a 14-bit CRC - the same as used in WSJT-X's FT8 encoder: using the polynomial 0x2757 and initial value of zero with no reflection[19]. I used this as it is 'good enough' - see the appendices for further details of CRC design.

[19] Ross Williams. A painless guide to crc error detection algorithms. URL `http://www.ross.net/crc/download/crc_v3.txt`

The 14 bit CRC is appended to the 112 source encoder bits yielding a 126 bit messsage.

Forward Error Correction

The digimorse message size (126 bits) is greater than that in FT8 (91 bits), but I'd like to achieve similar excellent error correction, so am fixing the code rate to be very similar to FT8. I could not find discussion on how the FT8 LDPC dimensions were determined; I'm assuming that choosing a similar rate would yield similar performance - I'm still studying LDPC design. Perhaps a simulation should be done to determine whether the dimensions I have chosen here are suitable.

The characteristics of LDPCs are:

- k = number of information bits
- n = number of codeword bits
- $R = \frac{k}{n}, n > k$ = rate of the code
- $n - k$ = number of redundant bits

For FT8, this is: $(n = 174, k = 91)$ so $R = \frac{91}{174} = 0.522988505747126$. There are $174 - 91 = 83$ redundant bits.

Using this rate R in digimorse, $\frac{126}{n} = 0.522988505747126$. So $n = 240.923076923077124$, which is not an integer. n must also be divisible by 3, so it can be partitioned into groups of 3 bits. So choose $n = 240$, which is divisible by 3, so that $n - k = 240 - 126 = 114$.

$R = \frac{k}{n} = \frac{126}{240} = 0.525$.

Therefore digimorse uses a (240,126) LDPC, giving 114 bits of redundant parity information.

The LDPC generator matrix was generated using the ldpc Rust library by Maxime Tremblay.

Doppler Shift protection and Synchronisation

The resulting 240 bits are partitioned into 3-bit groups, and transformed by the following Gray code, to ensure that bit sequences with adjacent tones differ in only one position.

The start and end of the symbols are surrounded by a Costas Array of tones (x,x,x,x,x,x,x). [20]

[20] Mike Hasselbeck WB2FKO. Synchronization in FT8, July 2019. URL `http://www.sportscliche.com/wb2fko/FT8sync.pdf`

Digital Modulator

8-GFSK modulation? Bandwidth vs transmission speed.

Channel

Typically HF or VHF, SSB 3kHz.

Digital Demodulator

Detecting Costas array, training narrow bandwidth filter on incoming signals.

Channel Decoder

Apply LDPC and CRC to recover frame. Output is a binary frame. Second pass to recover signals on top of each other?

Source Decoder

Given a binary frame that's correct, process its metadata.

Output transducer

If the decoded frame is within the current audio filter bandwidth, enqueue the Morse keying at the appropriate frequency.

Miscellaneous topics

The CQ detector. Transmit/receive switching.

Project Website

The website for the digimorse protocol and transceiver software is located at `https://devzendo.github.io/digimorse`. On the website, you'll find links to our GIT repository, mailing lists, bug tracker, and documentation.

Appendices

Morse Code Speeds

WPM	ms/dit	ms/dah	ms/wordgap
5	240	720	1680
6	200	600	1400
7	171.4285714	514.2857143	1200
8	150	450	1050
9	133.3333333	400	933.3333333
10	120	360	840
11	109.0909091	327.2727273	763.6363636
12	100	300	700
13	92.30769231	276.9230769	646.1538462
14	85.71428571	257.1428571	600
15	80	240	560
16	75	225	525
17	70.58823529	211.7647059	494.1176471
18	66.66666667	200	466.6666667
19	63.15789474	189.4736842	442.1052632
20	60	180	420
21	57.14285714	171.4285714	400
22	54.54545455	163.6363636	381.8181818
23	52.17391304	156.5217391	365.2173913
24	50	150	350
25	48	144	336
26	46.15384615	138.4615385	323.0769231
27	44.44444444	133.3333333	311.1111111
28	42.85714286	128.5714286	300
29	41.37931034	124.137931	289.6551724
30	40	120	280
31	38.70967742	116.1290323	270.9677419
32	37.5	112.5	262.5
33	36.36363636	109.0909091	254.5454545
34	35.29411765	105.8823529	247.0588235
35	34.28571429	102.8571429	240
36	33.33333333	100	233.3333333
37	32.43243243	97.2972973	227.027027
38	31.57894737	94.73684211	221.0526316
39	30.76923077	92.30769231	215.3846154
40	30	90	210
41	29.26829268	87.80487805	204.8780488
42	28.57142857	85.71428571	200
43	27.90697674	83.72093023	195.3488372
44	27.27272727	81.81818182	190.9090909
45	26.66666667	80	186.6666667
46	26.08695652	78.26086957	182.6086957
47	25.53191489	76.59574468	178.7234043
48	25	75	175
49	24.48979592	73.46938776	171.4285714
50	24	72	168
51	23.52941176	70.58823529	164.7058824
52	23.07692308	69.23076923	161.5384615
53	22.64150943	67.9245283	158.490566
54	22.22222222	66.66666667	155.5555556
55	21.81818182	65.45454545	152.7272727
56	21.42857143	64.28571429	150
57	21.05263158	63.15789474	147.3684211
58	20.68965517	62.06896552	144.8275862
59	20.33898305	61.01694915	142.3728814
60	20	60	140

Table 4: Morse code speed in words per minute and the timing of the elements in milliseconds.

On the choice of CRC width and polynomial

There are many possible CRCs, varying in width and choice of polynomial, to yield a specific Hamming Distance.[21] Some of the most effective ones are evaluated in Philip Koopman's work[22]. However Koopman's research is only correct for CRCs that protect data that is otherwise unencoded with further error correction, such as you might have in a network protocol. In the case of FT8 and digimorse (and NASA deep-space applications), the data is further protected by a good forward error correction mechanism, such as an LDPC. [23]

[21] Ross Williams. A painless guide to crc error detection algorithms. URL `http://www.ross.net/crc/download/crc_v3.txt`

[22] Philip Koopman. 32-bit cyclic redundancy codes for internet applications. Dependable Systems and Networks, 2002. URL `https://users.ece.cmu.edu/~koopman/networks/dsn02/dsn02_koopman.pdf`

[23] See the discussion on the WSJT-X development mailing list that starts at https://sourceforge.net/p/wsjt/mailman/message/36204768/

The CRC is used to verify the output of the LDPC error correction, which only emits valid codewords (or none at all). The WSJT-X developers found that any incorrect decodes that were valid codewords (i.e. satisfy all the parity checks) were most likely to have 20 bit errors. There are no CRCs in Koopman's research with a Hamming Distance this large. The developers say they have no reason to believe that a polynomial with a decent HD (say, 4) would have any bearing on how it would perform with 20 bit errors. Also, the polynomial chosen was subjected to tests that showed its effectiveness: the undetected error rate is very low. The degree of the polynomial is more important than its coefficients - the probability of an error being undetected is inversely proportional to $2^{14} = 16384$. All the CRCs they tested (including from Koopman's list) reduced the number of incorrect decodes by a similar factor.

Overview of the digimorse protocol and characteristics

TODO summarise all the key points of the protocol, framing, modulation, etc.

References

Mark Amos W8JI. An intuitive explanation of cw bandwidth. URL `http://www.w8ji.com/cw_bandwidth_described.htm`.

Thomas Clark W3IWI (SK) and Phil Karn KA9Q. Eme 2000: Applying modern communications technologies to weak signal amateur operations. Proc. Central States VHF Society, July 1996. URL `http://www.ka9q.net/papers/eme-2000.ps.gz`.

Steve Franke K9AN, Bill Somerville G4WJS (SK), and Joe Taylor K1JT. The FT4 and FT8 communications protocols. QEX (ARRL), July/August 2020.

Robert G. Gallager. Low-Density Parity-Check Codes. IRE Transactions on Information Theory, 1962.

Mike Hasselbeck WB2FKO. Synchronization in FT8, July 2019. URL `http://www.sportscliche.com/wb2fko/FT8sync.pdf`.

Philip Koopman. 32-bit cyclic redundancy codes for internet applications. Dependable Systems and Networks, 2002. URL `https://users.ece.cmu.edu/~koopman/networks/dsn02/dsn02_koopman.pdf`.

Michael A. Maynard K4ICY. YOU Can Learn Morse Code & Operate on CW! URL `http://www.k4icy.com/cw.html`.

Claude Shannon. A mathematical theory of communication. The Bell System Technical Journal, July/October 1948.

Kazimierz "Kay" Siwiak KE4PT and Bruce Pontius NØADL. How much "punch" can you get from diferent modes? QST (ARRL), December 2013.

Hans Summers G0UPL. Beginner's CW QSO. URL `https://qrp-labs.com/qcx/cwqso.html`.

Ross Williams. A painless guide to crc error detection algorithms. URL `http://www.ross.net/crc/download/crc_v3.txt`.