# One Bad Apple Don’t Spoil the Whole Bunch…

*Back in the early days of Dire Wolf (circa 2012), I tried an experiment to see if it was feasible to repair slightly corrupted AX.25 frames.*

*Attempting to fix single bit errors worked out pretty well when a heuristic was added to weed out obvious incorrect results. This is now the default with “FIX_BITS 1” the default.*

*Attempting to fix multiple bits was an interesting experiment but turned out to be a bad idea because it burned a lot of CPU time and very often produced incorrect results.*

*Around 2019, FX.25 forward error correction (FEC) was added. This greatly improves the reliability while retaining complete interoperability with existing equipment.*

There is an old proverb, “One bad apple spoils the barrel.” This also applies to AX.25 frames used for APRS and traditional packet radio. Each frame contains a 16 bit frame check sequence (FCS) used for error detection. If any one bit is corrupted along the way, the FCS is wrong and the entire frame is discarded.

The Osmond Brothers offered the advice, “Give it one more try before you give up…” That can also apply to AX.25 frames. From my observations, single bit errors are fairly common. Why not give it one more try before giving up?

My original attempt at receiving APRS signals performed the HDLC decoding real time on the bits as soon as they came out of the demodulator. If the FCS was wrong, the frame was discarded. The original bit stream was gone. No second chances.

In version 0.6, the HDLC decoder was rearranged to operate in two different phases. The first phase only looked for the special 01111110 “flag” patterns surrounding the frames. The raw received data was stored in an array of bits without undoing the “bit stuffing” at this time. This stream of bits was then processed in the second phase. This provides an opportunity to give it another try if it didn’t go well the first time.

For single bit errors, we can try to invert each of the bits – one at a time! – and recalculate the FCS. My experimentation found this recovered a lot of packets that would normally be discarded. Experimental results are summarized in a table later.

What about two or three adjacent bits getting clobbered along the way? If something is good, then more must be better. Right? The next experiment was to try modifying groups of two or three adjacent bits.

Why stop at modifying only adjacent bits? What about two non-adjacent (or “separated”) single bit errors? This also allowed a fair number of additional frames to be decoded but at a much larger cost. The processing time is proportional to the square of the number of bits so it climbs rapidly with larger packets. For larger frames this could be seconds rather than milliseconds.

**There is one little problem with flipping various bits trying to find a valid FCS. We get a lot of false positives on the FCS check and end up with bogus data. Callsigns contain punctuation characters. The information part has unprintable characters.**

The 16 bit FCS has 65,536 different possible values. Even if totally random data goes into the checking process, you will end up with a valid FCS one out of every 65,536 times. When you try hundreds or even thousands of bit flipping combinations and process lots of packets, a fair number will just happen to get past the FCS check and produce bad data.

My further refinement was to run the results through an additional sanity check. A good AX.25 frame will have:

- An address part that is a multiple of 7 bytes.
- Between 2 and 10 addresses.
- Only upper case letters, digits, and space in the addresses.
- Certain values in the frame control and protocol octets.
- For APRS, the information part has only printable ASCII characters *[see note below]* or these:
    - 0x0a line feed
    - 0x0d carriage return
    - 0x1c used by MIC-E
    - 0x1d used by MIC-E
    - 0x1e used by MIC-E
    - 0x1f used by MIC-E
    - 0x7f used by MIC-E
    - 0x80 seen in "{UIV32N}<0x0d><0x9f><0x80>"
    - 0x9f seen in "{UIV32N}<0x0d><0x9f><0x80>"
    - 0xb0 degree symbol, ISO Latin1
      (Note: UTF-8 uses two byte sequence 0xc2 0xb0.)
    - 0xbe invalid MIC-E encoding.
    - 0xf8 degree symbol, Microsoft code page 437

*[Later note: APRS comments and messages can also use UTF-8 characters but, in practice, this is extremely rare.]*

After applying this extra step of sanity checking, no bad data was visually observed for the single bit fixing case. In very large sample sizes, there were a few cases of bad data getting thru when flipping more than one adjacent bit. Obvious errors are fairly common when flipping two non-adjacent bits.

What about non-APRS frames? There is a configuration setting to perform a less stringent sanity check for general AX.25. The control and protocol bytes, of the frame, are not tested. The information part is not checked so "binary" data can be used. **A less strict sanity check makes it more likely for corrupted data to get through.**

In this example, the first decoder was able to achieve a valid FCS and plausible contents by flipping two non-adjacent bits. The third decoder received it with a correct CRC. Results were different so the duplicate detection did not combine them.

```
Digipeater WB6JAR-10 audio level = 23   [TWO_SEP]   .__
[0] N6VNI-14>APRS,WB6JAR-10*,WIDE,QIDE-6:!3356.05N/11758.61Wk Geo & Kris LaHabra,CA

Digipeater WB6JAR-10 audio level = 23   [NONE]   _:|
[0] N6VNI-14>APRS,WB6JAR-10*,WIDE,WIDE:!3356.05N/11758.01Wk Geo & Kris LaHabra,CA
```

In this example, the first and third decoders both found combinations of two bit changes that resulted in a valid FCS and plausible data. The second one does not look right with "/V" in the GPS sentence. The first one might or might not be correct. Checking the GPS checksum is left as an exercise for the reader.

```
N6QFD-9 audio level = 14   [TWO_SEP]   .__
[0] N6QFD-9>GPSTJ,WIDE2-
2:$GPRMC,020114,A,3409.7103,U,11804.0209,W,14.6,89.2,231105,13.5,E,A*30<0x0d><0x0a>

N6IFD-9 audio level = 14   [TWO_SEP]   __.
[0] N6IFD-9>GPSLJ,WIDE2-
2:$GPRMC,020114,A,3409.7103/V,11804.0209,W,14.6,89.2,231109,13.5,E,A*30<0x0d><0x0a>
```

Most of my earlier testing was done with Track 2 of the WA8LMF TNC Test CD (http://wa8lmf.net/TNCtest/index.htm). With the test CD, I got the following results for Dire Wolf version 0.6. Versions 0.7 and 0.8 had no changes in this area. In version 0.9, we use all 3 decoders running in parallel.

| | **Version 0.6** | | **Version 0.9** | | **Version 1.2** | |
|---|---|---|---|---|---|---|
| **Bits changed** | **Number of packets received** *(+ means change from previous line)* | **Percentage increase** *(in addition to preceding line)* | **Number of packets received** | **Percentage increase** | **Number of packets received** | **Percentage increase** |

| | | | | | | |
|---|---|---|---|---|---|---|
| None | 965 | - - - | 976 | - - - | 988 | |
| Single | + 12 | 1.2 | + 21 | 2.1 | + 15 | 1.5 |
| Two adjacent | + 2 | 0.2 | + 1 | 0.1 | + 3 | 0.3 |
| Three adjacent | + 0 | 0 | + 0 | 0.0 | +2 | 0.2 |
| Two separated | + 12 | 1.2 | + 24 | 2.4 | +9 | 0.9 |

Results were more impressive when listening to local stations live. About 23% additional packets were successfully decoded after flipping some bits and giving them another chance.

| | **Version 0.6** | |
|---|---|---|
| **Bits changed** | **Number of packets received** | **Percentage increase** |
| None | 6998 | - - - |
| Single | + 962 | 13.7 |
| Two adjacent | + 57 | 0.8 |
| Three adjacent | + 7 | 0.1 |
| Two separated (not adjacent) | + 572 | 8.2 |

Why such large disparities in the % increase? What is so much different about the local stations heard vs. the sample on the Test CD? I looked for a pattern in the packets that would normally be rejected but were recovered by flipping a single bit.

It doesn't seem to be correlated with a small number of stations. I tabulated where the signals came from (digipeater heard, not original source station) and they are from all over, not just a few stations. It doesn't seem to be correlated with audio deviation of the transmitted signal. Audio levels varied over a 9 to 1 ratio. A lot of people still don't get the concept of setting a proper transmit audio level.

Is it correlated to the type of system transmitting? Again, there doesn't seem to be a pattern. A wide variety of system types are represented.

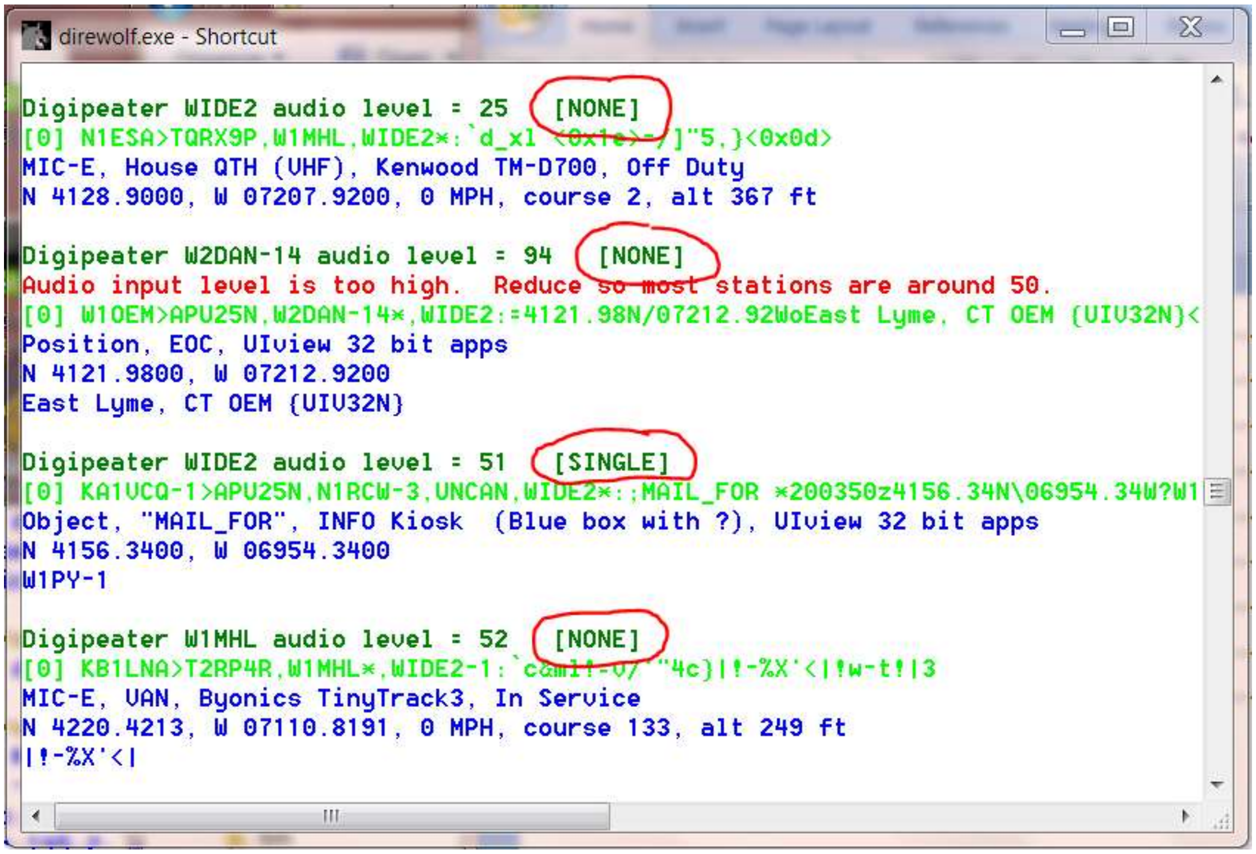
By default, only single bit fix up is enabled. You can experiment with the others with a configuration file setting. In the configuration file, specify the maximum level of correction to be attempted:

0 [NONE] - Don't try to repair.
1 [SINGLE] - Attempt to fix single bit error. (default)
2 [DOUBLE] - Also attempt to fix two adjacent bits.
3 [TRIPLE] - Also attempt to fix three adjacent bits.
4 [TWO_SEP] - Also attempt to fix two non-adjacent (separated) bits.

Example: Limit attempt to fixing a single bit:

```
FIX_BITS 1
```

The audio level line contains the number of bits that were changed to get a valid FCS on the frame. In most cases this will be NONE. Here is an example, where a frame that would normally be rejected, was recovered by changing a SINGLE bit.



```
Digipeater WIDE2 audio level = 25  [NONE]
[0] N1ESA>TQRX9P,W1MHL,WIDE2*:`d_xl <0x1e>-/]"5,}<0x0d>
MIC-E, House QTH (VHF), Kenwood TM-D700, Off Duty
N 4128.9000, W 07207.9200, 0 MPH, course 2, alt 367 ft

Digipeater W2DAN-14 audio level = 94  [NONE]
Audio input level is too high.  Reduce so most stations are around 50.
[0] W1OEM>APU25N,W2DAN-14*,WIDE2:=4121.98N/07212.92WoEast Lyme, CT OEM {UIV32N}<
Position, EOC, UIview 32 bit apps
N 4121.9800, W 07212.9200
East Lyme, CT OEM {UIV32N}

Digipeater WIDE2 audio level = 51  [SINGLE]
[0] KA1VCQ-1>APU25N,N1RCW-3,UNCAN,WIDE2*:;MAIL_FOR *200350z4156.34N\06954.34W?W1
Object, "MAIL_FOR", INFO Kiosk  (Blue box with ?), UIview 32 bit apps
N 4156.3400, W 06954.3400
W1PY-1

Digipeater W1MHL audio level = 52  [NONE]
[0] KB1LNA>T2RP4R,W1MHL*,WIDE2-1:`c&m1!-V/`"4c}|!-%X'<|!w-t!|3
MIC-E, VAN, Byonics TinyTrack3, In Service
N 4220.4213, W 07110.8191, 0 MPH, course 133, alt 249 ft
|!-%X'<|
```

It is important to remember that **we are not repairing errors**, we are only changing bits until we get a valid CRC. We could be adding additional corruption instead of repairing corruption.

The “sanity” check is **not a validity check**. We catch things that obviously look crazy but corrupted data could get through. One person compared this to playing a game of Russian Roulette. Most of the time you get lucky but sometimes, it doesn’t work out so well.

This feature is being provided for those who want to experiment with it. Don’t use if handling important information such as emergency communications. When using a “FIX_BITS” value greater than 1 you ***will*** get occasional bad data.

The Digipeater and IGate functions will process **only packets received with a correct CRC** to avoid relaying possibly corrupted data. Forwarding possibly corrupted data would be a disservice to the community.

Note that the possibly corrupted frames are passed along to any attached applications.  If you are using some other application to perform the digipeater or IGate functions, you could be passing along corrupted data.   Be sure to specify “FIX_BITS 0” for those radio channels.