A Digital Signal Processing Software Demodulator for Satellite Telemetry Using Brute Force Techniques

Douglas D. Quagliana, KA2UPW/5
dquagliana@gmail.com

Keywords: brute force, brute force demodulator, BFD, digital signal processing, DSP, 9600 baud, G3RUH demodulation, satellite, telemetry, RTLSDR

"When in doubt, use brute force."
-Ken Thompson, designer and implementer of the Unix operating system

Score one for Team Good Enough

It is often the case in computer science (and in life) that we need to solve some particular problem, and there isn't just one unique answer that works as a valid solution. There could be multiple answers that are all acceptable or close enough. For example, real world solutions to problems like turning the dial to tune in a radio station on shortwave, pointing an antenna for best reception, or adding sweetener to your morning coffee don't require an exact solution, just one that is "good enough." If you can hear the station clearly, the dial and antenna are sufficiently correct. For the mathematical versions of this class of problems, there are frequently formulas and approximations available that will give an answer that is "good enough" by using techniques such as successive approximations, least mean squares, stepwise refinement, or gradient descent.

In the case of the software based demodulator for G3RUH modulation, as long as the bit clock and bit rate are "good enough" we will correctly demodulate the packet. There are multiple similar correct solutions that will give the same valid packet. For example, the demodulator could be making the decision of whether the currently received bit is a one bit or zero bit just a little tiny bit[8] earlier or just a little bit later than optimal and everything will still work correctly.

In some cases it is desirable to get a correct answer even if it takes extra effort. Receiving satellite telemetry is one such case, particularly on a low elevation pass with weaker signals. Usually the data from the satellite will only be transmitted once, so we want to do everything we can to correctly demodulate the data packet even if that means taking extra time and effort. For G3RUH demodulation, there are only a few variables to consider and they have a limited number and range of values that they can accept. If we know that the signal is a 9600 baud signal, then the real baud rate must be very close to 9600.00 baud. We can probably bound the baud rate value to be between 9598 and 9602 baud (or so) with high certainty. The phase at a particular moment in time could be any value between zero and 359.9 degrees.

Moore's Law and increasingly faster networking speeds have yielded computer and processors that have advanced to the point where calculations that were once not even considered are now easily performed. For example, it was once considered totally impracticable to search every computer on the public Internet by making a network connection to every one of the over four billion possible IPv4 address. But, researchers are now scanning the entire Internet for their research projects.[9] It's now possible to complete a scan of the entire Internet in under five minutes.[10]

It is often the case that we want to try to solve the problem efficiently using as little computing/processing power as possible. Everything else being equal, if one method takes very little time and another method takes several days, then we probably want to use the quicker method. For example, when using a laptop or cell phone, efficiency could increase battery life. But on a computer connected to the AC power mains we don't need to worry about battery life. However, if the difference is only a few milliseconds between the quicker and slower methods, then we might not even care. And we don't always need to be optimally efficient to be quick. A modern CPU is fast enough that a human will not even notice the extra processing. In fact, for some problems we can just try all possible values for all of the variables and see if any of them give us a valid answer. This is the Brute Force Demodulator (the "BFD"[3]) approach.

The Brute Force Demodulator (a.k.a. "The BFD")

As an example, let's consider the 9600 baud G3RUH AX.25 packet software modem. Specifically, let's look at the demodulator. The demodulator is usually looking at the audio output from a radio receiver, which we presume contains a radio signal representing a valid bit stream from a remote transmitter. In order to recover the bit stream, the 9600 baud G3RUH demodulator needs to correctly determine two things: when is the right time to decide the value of the current bit being received, and, at that instant, is the bit is a zero bit or a one bit[2]. The timing has to be decided about 9600 times a second, but there actually is an optimal instant in time to make the decision (the optimal time is when the eye is open widest) but we don't know ahead of time exactly when that optimal instant is. However, if we pick moment in time that is "close enough," when the eye is open "wide enough," then we will still get the correct answer for whether this is a one or a zero bit. Many demodulators use a local clock running at 9600 Hertz which is then adjusted using some scheme based on the input signal.

The bits might be arriving at "about" 9600 baud. The 9600 baud clock of the transmitter might be slightly off frequency, and/or his digital to analog converter might be off, and/or our own analog to digital converter might be slightly askew and/or there might be Doppler. As seen by our demodulator, the bits could be arriving at 9599.9 bits per second or 9600.1 bits per second or some other nearby value. In any of these cases, if we were to blindly make the bit decisions at a fixed rate of exactly 9600.000 bits per second, then rather quickly we will start seeing bit errors because we are no longer deciding near the optimum moment in time. That is, our clock is "slipping" compared to the transmitter's clock and the two clocks will only align briefly with each other some of the time. The usual solution for a demodulator is to adjust the local clock repeatedly to keep it in synchronization with the transmitter's clock. For example, one approach for solving the clocking problem is to look at the received waveform and see if we are deciding the bit a little bit early or a little bit late or at just the right

time[4] and then giving our local clock a nudge in the right direction. Another clocking approach is to model the bit clock as a phase locked loop (PLL) and adjust the PLL slightly on each received bit until we achieve "lock." Both methods are relatively simple, work well, and involve only minor calculations. But if we change the local clock too much (or not enough) then we can cause bit errors by deciding at the wrong instants.[15] If the demodulator does decide incorrectly (only one time), it could also go back and assume the weakest valued bit was the wrong bit, flip it, and try checking the CRC for the packet again (with that one bit changed) just to see if that fixes everything. Some software demodulators will do this. However, there's another simple approach: brute force.

Just try every possible value. If it's in there, we'll find it.

Brute force is a method that doesn't use any intelligence, but just blindly tries solutions until it finds the answer. Brute force safe crackers would try every possible combination to the safe ("left zero, right zero, left zero", then "left zero, right zero, left one", and so on). Brute force password programs can try every possible eight letter password, for example, starting with "aaaaaaaa", then "aaaaaaab", then "aaaaaaac", through "aaaaaaaz" to "aaaaaaba" and so on. Computers are really good at this sort of thing, and they can do it really fast.

Using brute force for a demodulator is not a new idea, but the idea has not received as much attention as other methods. In the brute force approach for a software G3RUH demodulator, the demodulator just tries all possible values for the bit clock frequency and phase. Actually, we don't have to try every possible value. As soon as a combination is found that yields a valid bit stream we can stop. We can also speed up the work by trying the most likely values first and then try the less likely values, or for a continuous signal we can try the values that worked last time and if they don't work then try all possible values.[12]

In the brute force demodulator approach, the demodulator tries multiple bit rates and only makes the assumption that the exact bit rate, whatever it is, is fixed for the duration of the AX.25 frame that it is trying to demodulate. For a software modem we can implement a brute force approach in a couple of ways. By analogy of a depth-first-search, if we have buffers full of samples, then we can try a particular frequency/phase starting "here" at this sample and see if we get a valid frame. If not, increment the frequency (or the phase) by a small amount and try again on the same buffer of samples until you either see a valid frame or until you have tried all possible values (which is what will happen if there's just noise being received). In the latter case, advance forward in the buffer by some number of samples (or onto the next buffer) and start over.

For example, for 9600 baud G3RUH, we could try to demodulate AX.25 frames by creating a single demodulator and starting with the bitclock phase at 0 degrees, and the baud at 9598.0 bits per second. If we don't get a valid packet then we start over, incrementing the bit rate slightly and try again on the same incoming samples. Keep incrementing until we get to, for example, 9602.0 baud, and then if we still don't have a valid AX.25 frame we can increment the starting phase by a few degrees and try all the possible baud rates all over again. Keep going until all possible phases have been tried or until you have a valid packet. Alternatively, we could implement multiple demodulators all running in parallel

(think: breadth first search) and send each digital sample to all the demodulators until one of them finds a valid frame at which point we reset all the demodulators. The breadth first search has the advantage that all of the demodulators run at the same time, they are all looking at the same sample, and we don't need a buffer of samples or buffer management of the incoming samples except for the current sample.

In the C programming language, a struct for defining a single demodulator might look like this[17]

```
struct demodulator {
   double  clockfreq;       // nominal clock frequency (near 9600.0)
   double  startclockphase; // zero to 2-pi (or 0 to 360)
   double  clockphase;      // current clock phase
   double  phaseinc;        // phase increment for this clock frequency
};
```

We can declare a single demodulator, call it "dem," using the above struct. This demodulator has a fixed clock frequency near but not exactly 9600 baud, a starting clock phase (zero to two pi, although you could just as easily use zero to 360 degrees), the current phase of the clock and the phase increment amount to be used at this clock frequency (stored so that we don't have to recalculate it on every sample).

Just a easily, we can define a hundred (or even a thousand!) software demodulators as an array of dem[] and then initialize all of them as such

```
i = 0;
for (j=BAUD_LOW; j <= BAUD_HIGH; j++ ) {
   for ( k=0; k < CLOCK_PHASES_PER_BAUD; k++ ) {
     dem[ i ].clockfreq = j;
     dem[ i ].startclockphase = (M_TWOPI / CLOCK_PHASES_PER_BAUD) * k;
     dem[ i ].clockphase      = (M_TWOPI / CLOCK_PHASES_PER_BAUD) * k;
     dem[ i ].phaseinc        = ( j * M_TWOPI ) / samplerate;
      i++;
   }
}
intTotalDems = i;
```

then as each new "current sample" arrives we look at each demodulator, increment the current clock phase using the phase increment amount. If the current clock phase just went over two pi (or over 360), then it is time to decide the bit, otherwise there's nothing to do yet.

```
while "there are still more samples" {
   for ( z = 0; z < intTotalDems; z++ ) {
      // get the address of the current demodulator
      p = &dem[ z ];

     // increment the clock for the current sample
      p->clockphase = p->clockphase + p->phaseinc;

     // Is it time to decide this bit?
```

```
      if (p->clockphase > M_TWOPI) {
            p->clockphase = p->clockphase - M_TWOPI; // reset it
            if ( currentsample >= 0 ) {
               currentbit = 1;
            } else {
               currentbit = 0;
            } // else
      } // if
   } // for
} // while
```

Now, in actual 9600 baud AX.25 we still have to save the currentbit and unscramble the bits, undo the NRZI encoding, build up the current byte and check for HDLC flags, undo the zero bit stuffing, and start building the AX.25 packet frame.[1] If there's a valid signal present, then sooner or later one of the demodulators sees the second HDLC flag. We check the CRC and if it is a match then we output the valid AX.25 frame, reset all the demodulators and start all over again.[16] It's important to note that the demodulator that gets a valid frame might not be deciding at the optimal time, but it's good enough that we decided all of the bits correctly and we got a valid CRC. We only need to receive this packet once. As long as it's been verified as a valid frame, it doesn't matter if the solution was not optimal or how much computing power was expended. We got the right answer.

But, some people might say, that's a lot of wasted computations. There are more efficient ways that would use fewer CPU cycles. Perhaps, but were you really doing something else important with those CPU cycles? Probably not. A modern computer will be running with a clock speed over 2 GHz with (at least one) quad core CPU. On a single quad core CPU that's "about eight billion" CPU instructions per second. The above calculations take a few hundred, perhaps a few thousand machine instructions per demodulator leaving over seven billion machine instructions per second left over for other uses. Most modern CPUs actually spend most of their time idle doing absolutely nothing unless the CPU has purposely been configured to be kept busy with a background task like video transcoding or a BOINC project such as SETI@Home.

Brute force uses more CPU power, more time, and more electricity, but if it's in there, we'll find it. Is it efficient? No. Does it give the right answer? Yes, and more often than some other software demodulators. Is this the be-all-and-end-all of software modems? Actually, it's not. There is still other processing that can improve the performance. The incoming audio still needs to be filtered to remove noise added by the receiver and the communications channel, and the receiver could do a better or worse job. The receiver itself could be mistuned, which on an FM receiver may or may not result in a DC offset of the signal that might need to be corrected. In addition, the actual frequency of the transmission could change if the transmitter is drifting (or changing due to Doppler). In addition, there could be more than one transmitter on the same channel (think: APRS) which will require the receiver to re-synchronize on potentially every received frame. (It is left as an exercise to the reader to determine whether or not a collision, that is, two stations transmitting at the same time, can be undone by subtracting one received signal from the combined received signals leaving only the second signal.) Other noise or interference on the channel could be characterized and notched or digitally subtracted.

And all of this work is just for a single demodulator for a single channel. In the future, a wideband software defined receiver could do signal classification to identify the types of data signals being received and then create dozens or even hundreds of different software modems to receive and demodulate many different types of signals simultaneously.[5]

Next time you have recording of a signal where you know there's a signal in the recording, but you can't demodulate it correctly, consider trying brute force techniques.

Footnotes

1 For a good discussion of all of the lower level details of AX.25, particularly with respect to Bell 202 modulation, but also applicable for 9600 G3RUH AX.25 see
Finnegan, Kenneth W. and Benson, Bridget, PhD. "Clarifying the Amateur Bell 202 Modem." 33rd ARRL and TAPR Digital Communications Conference. 2014. Also available at
https://www.tapr.org/pdf/DCC2014-Amateur-Bell-202-Modem-W6KWF-and-Bridget-Benson.pdf
In addition, there's some good technical documentation and C code that does all of the low level AX.25 packet work in Moe Wheatley's KS12V10 software at http://www.moetronix.com/ae4jy/projects.htm.

2 If we were considering a PSK demodulator, or a G3RUH 9600 signal from a wide bandwidth RF receiver like an RTLSDR dongle rather than the audio from the discriminator, then we would also need to brute force the solution to the carrier frequency. To keep the example simpler, in this paper we just consider the filtered 9600 baud audio signal which does not need to worry about the RF carrier frequency. The constructing of a signal classifier to watch multiple megahertz of received RF from an RTLSDR dongle[7] for multiple 9600 baud signals at arbitrary (and Doppler-shifting) frequencies from different transmitters, demodulate and then brute force each one is left as an exercise to the motivated reader with an excess of disposable free time. The author would appreciate a copy when it's finished. Interested programmers should contact the author.

3 To be clear, the acronym "BFD" as used here is the "Brute Force Demodulator." It is not a municipal Fire Department, nor should it be confused with other popular culture BFx acronyms such as the "BFG", which depending on your age and background is either a fictional giant by author Roald Dahl or a weapon from the computer game Doom. Neither is "BFD" related to "BFM", which is both an acronym from quantum field theory and also a special kind of black magic. Nor is it related to the "BFR", a SpaceX rocket, nor to the "BFB", a really large burrito from a local Mexican restaurant. But I digress…

4 Recovering the clocking rate using this method is usually called early-late gate timing recovery but it might be more memorable as the Goldilocks approach ("This clock is too fast; this clock is too slow; but this clock is just right.") It is relatively simple and it works well. Call it early-late gate or call it Goldilocks, just don't call me late for the satellite pass. Time, tide, and orbital mechanics wait for no one.

5 Imagine running one program to look at all of the HF bands during Field Day or CQWW and having it show you all the CW signals, all the PSK31 signals, all the FT8 signals, all the RTTY signals, all the slow scan pictures and having it performing speech recognition on the SSB signals[6] to show you all of their callsigns. Additionally, it should tell you which stations you've already worked in this contest and

prioritize which ones you should work next for maximum point values. Today, CW Skimmer can receive multiple CW signal within the receiver bandpass and simultaneously demodulate all of them, however it only demodulates the CW signals. Imagine the same sort of thing, but simultaneously for all bands and all modes including voice.

6 Hint: A general purpose English language speech-to-text engine is not needed here. We just need someone to write a speech recognition engine that recognizes ITU (and DX) phonetics and then runs the results through a super regular expression to check for "CQ" and valid amateur radio callsigns. Interested programmers should contact the author.

7 Of course, the ultimate version of this receiver has an array of multiple high-speed wide-bandwidth receivers (such as the RTLSDR dongles) which can have their received digital signals arbitrarily phased together allowing simultaneous reception of multiple satellite signals (and demodulation of multiple different types of signals) with all of the advantages of phased arrays (such as a lack of moving parts, directional gain, target tracking, putting a null on the unwanted signals, reprocessing the same data phased differently for different targets, and so on). And, oh yes, I think you can track multiple amateur radio satellites at the same time and it can be done today. For an example of a RTLSDR phased array that uses four RTLSDR dongles, that tracks multiple GPS satellites, and that does not need any hardware to synchronize the dongles, see https://www.gnuradio.org/wp-content/uploads/2017/12/Wil-Myrick-GPS-Beamforming-with-Low-Cost-RTL-SDRs.pdf and then watch his YouTube video from GRCon17 at https://www.youtube.com/watch?v=IsmCQs5KVCs.[18] Now we just need to do a similar project for the amateur radio satellites. Interested programmers should contact the author…Hey, don't look at me like that. It's only software.

8 Pun intended.

9 Heninger, Nadia. Durumeric, Zakir. Wustrow, Eric. Halderman, J. Alex. "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices." https://factorable.net/weakkeys12.extended.pdf

10 Your mileage may vary, some restrictions apply.[11]

11 "On a computer with a gigabit connection, ZMap can scan the entire public IPv4 address space in under 45 minutes. With a 10gigE connection and PF_RING, ZMap can scan the IPv4 address space in 5 minutes." See https://zmap.io/ for details. This could be useful the next time your ISP changes your home IP address while you're away on vacation.

12 Karn uses this scheme in some of his BPSK software demodulators including his demodulators for ARISSat[13] and ACE[14].

13 http://www.ka9q.net/bpsk1000.html. See the section on "Tracking and demodulation".

14 http://www.ka9q.net/code/acedemod/. See the section on "Offset Searching".

15 Possibly the demodulator decides incorrectly too many times and even the forward error correction (FEC) can't fix it up in post analysis. You ***are*** using forward error correction on your satellite downlink, aren't you? If you aren't using FEC on your digital satellite downlink, then you're doing it wrong!

16 Sometime random noise looks like an AX.25 packet. You only need some noise that looks like an HDLC flag, then a little later some more noise that also looks like an HDLC flag. Usually the noise before the second HDLC flag will not demodulate to a valid CRC value, but about one of every 65536 times, just by dumb luck, it will. These "packets" almost always look like complete garbage, particularly the callsign fields, so they are easily filtered and ignored if you are looking for satellite telemetry from a known callsign or set of callsigns. This "false frame" problem isn't unique to software modems, but it is more common with brute force techniques than you would see otherwise.

17 If there's interest in the C code for the G3RUH brute force demodulator, I'll put it up on github or some other open source code repository. Where do you think it should be published?

18 My thanks to Michelle Thompson for pointing me at these references when I half-remembered them and then couldn't find them again.