Going Beyond 9600 Baud

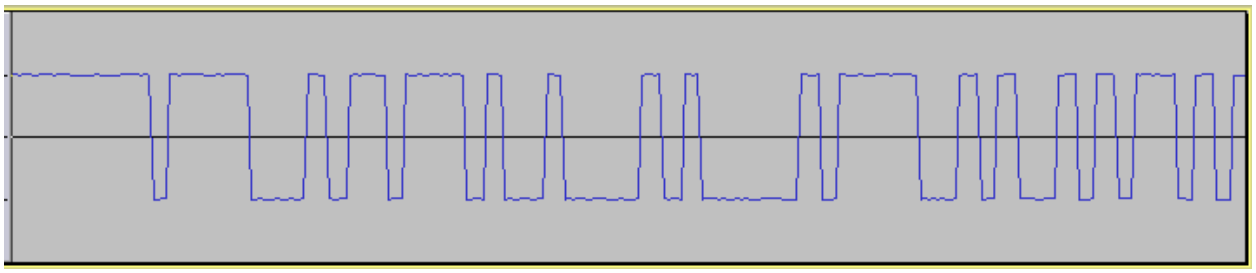
First rough draft - March 26, 2016

Improved PLL results – May 27, 2016

Recently there have been an increasing number of queries about making Dire Wolf handle data rates higher than 9600 baud.

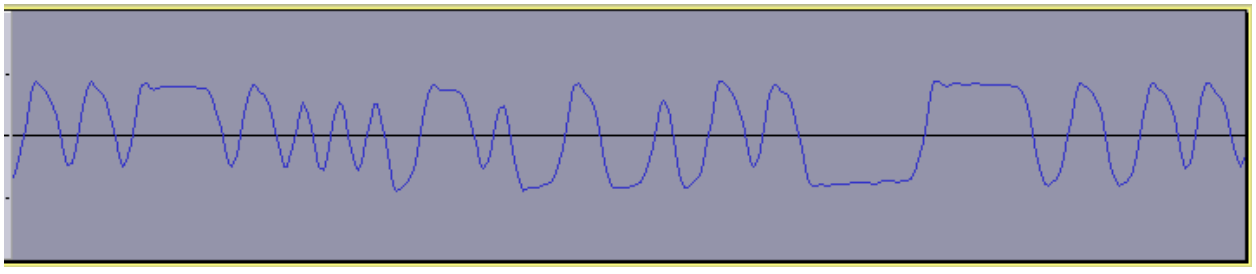
> *Note: The terms **baud** and **bits per second (bps)** are often used interchangeably. In this case they are the same number so it doesn't matter. When using more advanced modulation techniques, such as PSK, there is a difference and we need to be more careful about using the proper term. That is a subject for another place and another time.*

First, let's take a step back and review what a 9600 baud signal looks like. The ideal signal would look something like this with two voltage levels for the binary signal. Each pulse width is some integer multiple of 1/9600 second.



On the receiving end, we use a digital phase locked loop (DPLL), synchronized to the zero crossings, to determine where to sample for the data bits.

In the real world, signals end up looking like this after the bandpass limits of the transmitter and receiver.



The amplitude is not that important because we slice at the zero level to get a binary value. The jitter of the zero crossing points is very important because this is where we get our timing. If the sample rate is too low, we will get more jitter.

Dire Wolf version 1.3 and earlier placed artificial restrictions at 9600 bps for the data rate and 48000 Hz for the audio sample rate. Higher end "soundcards" can handle 96,000 and sometimes 192,000 samples per second. These limits, in software, have been lifted so we can start experimenting with higher rates. Of course, higher rates mean more CPU power is required.

It is intuitive that too low of a sampling rate will not get an adequate measure of the shorter pulse widths. Let's do some experiments to confirm this …

Eventually we will need to put radios in the middle, but we can gain some insights from simulation. "gen_packets" takes the place of the transmitter. "atest" is the receiver. There are no antennas, just an audio file between them. A typical test would look something like this:

```
$ gen_packets -r 44100 -B 9600 -n 100 -o test.wav
Audio sample rate set to 44100 samples / second.
Data rate set to 9600 bits / second.
Using scrambled baseband signal rather than AFSK.
Output file set to test.wav
built in message...

$ atest -B 9600 test.wav | grep "packets decoded in"
52 packets decoded in 0.170 seconds.  44.0 x realtime
```

This generates 100 packets with increasing levels of random noise. In this case we see that 52 of them were decoded successfully. Let's try this with other values and see what happens.

Improvement May 2016

The original processing of zero crossings was crude and added jitter. This has been improved so we now get better results with low audio sampling rates. This is in 1.4 development snapshot C.

gen_packets was temporarily adjusted to have a lower than normal noise level so these are not the numbers you will see if you play along at home. The difference between the two tables is the improved clock recovery strategy which allows a lower sample rate to data rate ratio.

Old style PLL:

Data Rate, bps (**-B** option)	Audio Samples per second (gen_packets **–r** option)				
	22050	**44100**	**48000**	**96000**	**192000**
9600	33	75	83	99	
14400	0	50	57	96	100
19200	0	33	33	82	99
28800	0	0	0	61	96
38400	0	0	0	31	82

New style PLL:

Data Rate, bps (**-B** option)	Audio Samples per second (gen_packets **–r** option)				
	22050	**44100**	**48000**	**96000**	**192000**
9600	45	88	91	99	
14400	0	66	69	98	100
19200	0	45	66	88	99
28800	0	0	0	64	98
38400	0	0	0	67	88

For a **sample rate** to **data rate** ratio of

- 2.3 (green) we went from 33 to 45. Big improvement but still pretty bad.
- 2.5 (pink) we went up from 31-33 to 66-67. The number was doubled! Still you would want to avoid a ratio this low.
- 5 (yellow) we went up from 82-83 to 88-91. Significant but not dramatic. This would be the lower end for best results.
- 6.7 (turquoise) we are at the point of diminishing returns. Negligible improvement by increasing the sample rate.

Lessons learned:

- Don’t use 22k sample rate.
- Increasing sample rate from 44.1k to 48k can provide some improvement for 9600 baud. (Note that the cheap USB audio adapters can usually handle only these two sample rates.)
- Strive for a ratio of 5 or more between the audio sample rate and the data rate. Going above 7 doesn’t seem to offer much additional improvement.

Of course, the big unknown is what the radios will do in the middle. If we optimize for best results in the simulated environment, we might make things worse in the real world.

How do we know what sample rates are supported by the hardware?

Sometimes having device drivers hide the physical reality is not a good thing. Consider this example.

What audio devices do we have? This is what we find on an old desktop x86 Linux box.

```
john@linux64:~$ arecord -l
**** List of CAPTURE Hardware Devices ****
card 0: Intel [HDA Intel], device 0: AD1984 Analog [AD1984 Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 0: Intel [HDA Intel], device 2: AD1984 Alt Analog [AD1984 Alt
Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: Device [C-Media USB Audio Device], device 0: USB Audio [USB
Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

"Card 0" is apparently this chip, http://www.analog.com/media/en/technical-documentation/obsolete-data-sheets/AD1984.pdf on the motherboard, which supports sample rates of 8, 11.025, 16, 22.05, 32, 44.1, 48, 88.2, 96, 176.4, and 192 kHz.

"Card 1" is a cheap C-Media USB-Audio adapter. When we read the spec sheet for the chip, we see that is it physically capable of only 44100 and 48000 samples per second. That's all.

We can determine its physical capabilities with a command like this:

```
sudo lsusb -vv | egrep 'Audio|tSamFreq' | egrep -v 'Descriptor|bInterfaceClass'

  iProduct                 1 C-Media USB Audio Device
        tSamFreq[ 0]          48000
        tSamFreq[ 1]          44100
        tSamFreq[ 0]          48000
        tSamFreq[ 1]          44100
```

Here we are asking card 1 for a sample rate of 192k, four times the physical limit, and the request is honored.

```
john@linux64:~$ direwolf -B 19200 -r 192000
Dire Wolf DEVELOPMENT version 1.4 A (Mar 26 2016)
Includes optional support for:  gpsd

Reading config file direwolf.conf
Audio device for both receive and transmit: plughw:1,0  (channel 0)
Channel 0: 19200 baud, K9NG/G3RUH, +, 192000 sample rate x 2.
The ratio of audio samples per sec (192000) to data rate in baud
(19200) is 10.0
This is a suitable ratio for good performance.
Note: PTT not configured for channel 0. (Ignore this if using VOX.)
Ready to accept AGW client application 0 on port 8000 ...
```

```
Use -p command line option to enable KISS pseudo terminal.
Ready to accept KISS client application on port 8001 ...
```

I suspect that the driver is performing a rate conversion and returning 4 samples to the application for every one sample from the hardware. This doesn't help us any. We need a way to find out about the physical capabilities of the hardware and make sure that the drivers are not hiding that reality.

Here is something interesting. It's perfectly happy with any crazy sample rate. Let's try 54321. It works!

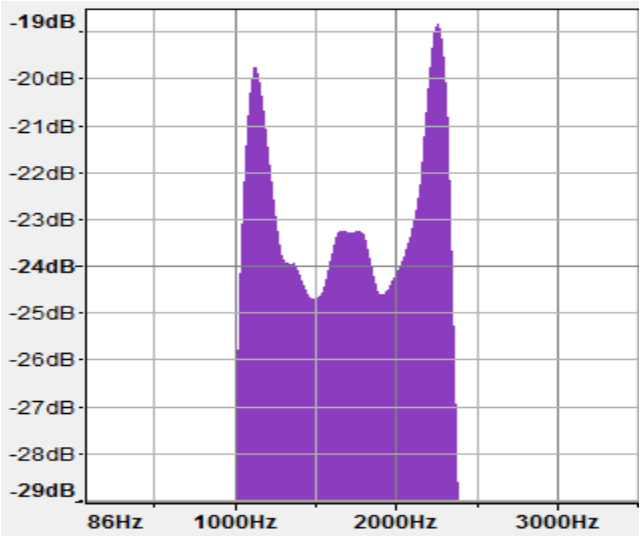
```
john@linux64:~$ direwolf  -r 54321
Dire Wolf DEVELOPMENT version 1.4 A (Mar 26 2016)
Includes optional support for:  gpsd

Reading config file direwolf.conf
Audio device for both receive and transmit: plughw:1,0  (channel 0)
Channel 0: 1200 baud, AFSK 1200 & 2200 Hz, E+, 54321 sample rate.
Note: PTT not configured for channel 0. (Ignore this if using VOX.)
Ready to accept AGW client application 0 on port 8000 ...
Ready to accept KISS client application on port 8001 ...
Use -p command line option to enable KISS pseudo terminal.

Digipeater W1MRA audio level = 1(0/0)   [NONE]   ||||||___
[0.2] N1OHZ>T2QT2T,W1MRA*,WIDE2-1:'cN]l <0x1c>-/
MIC-E, House, Unknown manufacturer, In Service
N 42 14.2400, W 071 50.6500, 0 MPH

Digipeater WIDE2 (probably UNCAN) audio level = 2(0/0)   [NONE]
_||||:___
[0.3] N1OHZ>T2QT2T,W1MRA,UNCAN,WIDE2*:'cN]l <0x1c>-/
MIC-E, House, Unknown manufacturer, In Service
N 42 14.2400, W 071 50.6500, 0 MPH
```

The application is getting 54321 samples per second. The hardware is capable of only 44100 and 48000.

So, there is our next challenge. How do we find out the actual hardware capabilities? How do we make sure that the driver isn't deceiving us? What bandpass restrictions are imposed by the analog amplification stages?

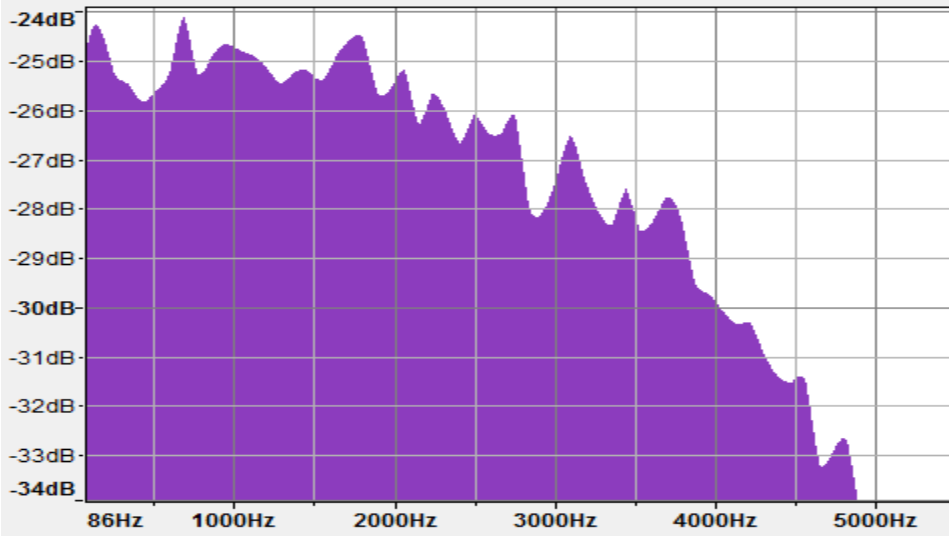
Radio bandwidth

The spectrum for a 1200 baud AFSK signal looks like this:



This easily fits into the normal voice range and is not fussy about the radio audio characteristics.

The spectrum for a 9600 baud signal looks like this:



Note the peak at 4800 Hz. This corresponds to the maximum frequency of alternating 0 and 1 bits. The upper end of the audio passband needs to extend to at least 5000 Hz.

If you want to use 19.2k baud, double that. For 38.4k, double it again.

Circuits in the radio and your “soundcard” must have a bandwidth of at least half the baud rate.

It's not as obvious how low we need to go. There is a peak around 200 so I suspect that a high pass filter, intended to keep CTCSS frequencies away from the speaker, would cause issues.

IT WON'T WORK WITH THE MICROPHONE AND SPEAKER CONNECTIONS!

The pre-emphasis and de-emphasis will distort the signal and make it completely unusable.

If your radio doesn't have a suitable "data" connector, which bypasses the normal audio processing, you will need to do some surgery. (See http://wa8lmf.net/miscinfo/MiniDIN6-Packet.pdf for a good description of the connector.)

It probably won't work if you have audio transformers in the middle.

Soundcard bandwidth

Your "soundcard" must also have wide bandwidth. A 9600 baud signal could contain a 4800 Hz square wave if the right combination of bits is present. It also needs low frequency response.

Take a look at the response of a popular chip used in cheap USB Audio adapters: http://www.hardwaresecrets.com/datasheets/CM108.pdf Look in section 9.3.2 of the data sheet and notice that the frequency response is flat, within 1 dB, from roughly 50 Hz to 15 kHz. This is what we want. Wide and flat to minimize distortion.

Compare the CM108 frequency response with the SignaLink USB: http://www.frenning.dk/OZ1PIF_HOMEPAGE/SignaLinkUSB-mods.html Response is bumpy and falls off a cliff above 2.5 KHz. OK for 1200 baud but there is no possible way this could ever work for 9600 baud.

The next step on our journey

Questions:

1. How can we determine which sample rates are supported by the hardware?
2. How can we verify that we are receiving the native sample rate and the device driver is not performing a rate conversion?
3. How can we measure the bandpass characteristics of the soundcard?

Ideally: We would like a little application to assess the hardware capabilities. The user would connect a cable from the audio output to the audio input. The application would:

- Determine what native sample rates are available.
- Run a bandpass test for each of them.
- Plot the results.
- Provide some commentary on how it might perform for higher speeds.