What’s in Your Audio Library?

Are your digital song files mismatched and overly compressed? We can help.

BY STEVE WALKER ·

PUBLISHED: DECEMBER 22, 2021 · UPDATED: MAY 4, 2022

At the time of writing, the author was assistant chief engineer for Radio One Dallas; he subsequently joined the technical support department at Wheatstone.

When our station KSOC became the first in Texas to broadcast in HD Radio, we knew that we needed to be on top of our audio quality in order to best take advantage of the new technology. But like many stations we really had no idea where much of our music came from.

We knew that, although the songs in our playout system were all stored as uncompressed WAV files, at least some had originated as MP3 files. We wanted to find a way to identify those songs so that they could be replaced with pristine, uncompressed audio files.

It’s important to use the best-quality source material for on-air broadcast, but especially so when you are broadcasting in HD. The GIGO principle applies: Garbage In, Garbage Out.

Kirk Harnack, senior solutions consultant at Telos Alliance, had this to say about broadcasting previously compressed source audio on an HD Radio station: “Audio that’s been psychoacoustically encoded and decoded is now missing the ‘low-hanging fruit’ that the original encoder identified and eliminated or modified. If we cascade another psychoacoustic audio encoding algorithm after the first one, the second encoder will not have the benefit of the natural audio’s content that was easy to eliminate.”

So it’s in our interest and that of our listeners to ensure that the audio we broadcast, which will be processed by the HD Radio encoder, hasn’t already been subjected to a lossy compression algorithm. With hard drive space as abundant and affordable as it is today, storage space is no longer a reason to obtain our music in a compressed format such as MP3.

But short of listening to every song in the library with a critical ear in a studio, how could we determine which songs had once been psychoacoustically compressed? All of our songs are now stored as WAV files so just looking at the file extension or the file size gives no hint whatsoever.

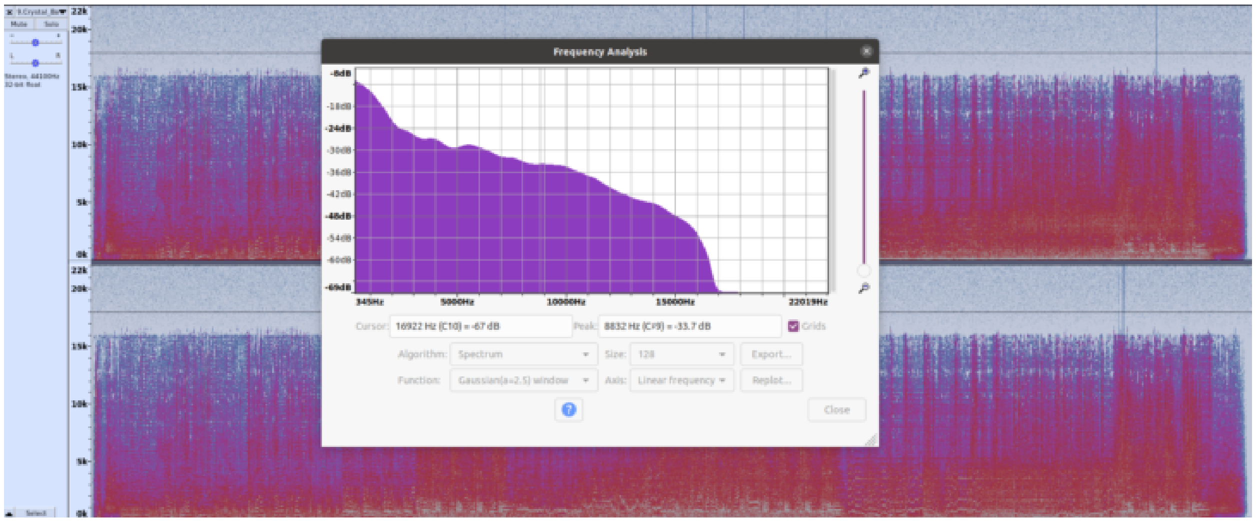
It turns out that there are certain clues visible in the spectrogram of an audio file that can help identify the formerly compressed songs. The most obvious one is the cutoff frequency used by the encoder.

When a file is compressed to MP3 format, the algorithm attempts to remove parts of the audio that the designers of the standard felt wouldn't be missed by the human ear in an attempt to reduce file size.

Part of this is the cutoff of all audio content above a certain frequency. That frequency varies according to the bitrate of the MP3 compression scheme.

From my tests, it seems that a bitrate of 192 kbps results in a cutoff of audio above about 18 kHz. A rate of 128 kbps cuts off above 16–17 kHz. This is easy to see when looking at the spectrogram of an MP3 song.

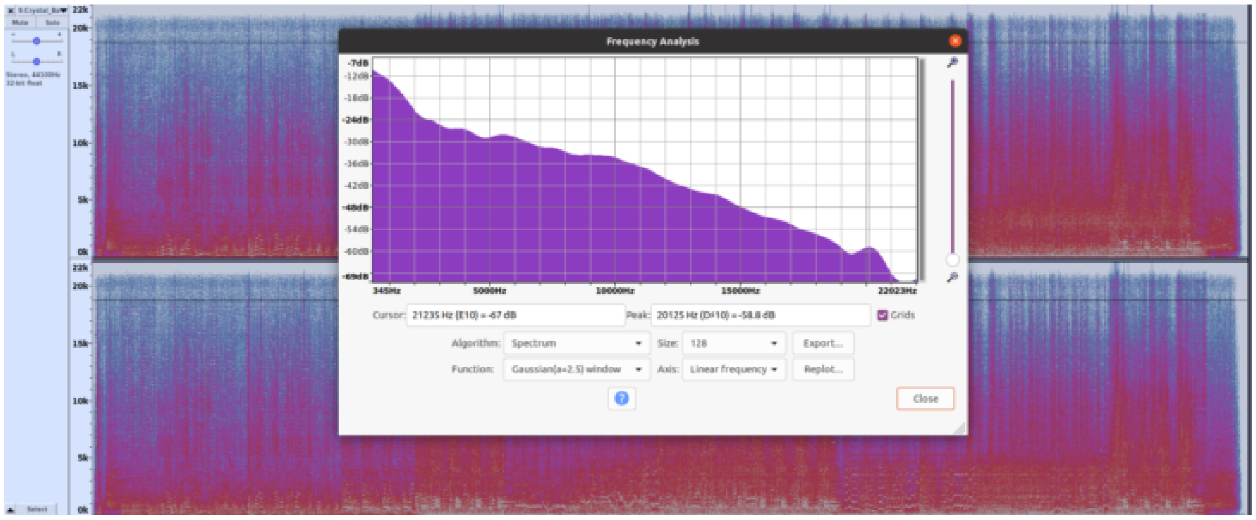
See Fig. 1. Notice that at 17 kHz, the audio levels of this file are already in the noise floor.



Cursor: 16922 Hz (C10) = -67 dB
Peak: 8832 Hz (C#9) = -33.7 dB
Algorithm: Spectrum
Size: 128
Function: Gaussian(a=2.5) window
Axis: Linear frequency

Fig. 1: Spectrogram of "Crystal Ball" by Styx as MP3

Looking at the uncompressed version of the same song in Fig. 2, we don't get to the –67 dB level until we reach the 21 kHz frequency range.



Cursor: 21235 Hz (E10) = -67 dB
Peak: 20125 Hz (D#10) = -58.8 dB
Algorithm: Spectrum
Size: 128
Function: Gaussian(a=2.5) window
Axis: Linear frequency

Fig. 2: The same song, linear uncompressed.

Put on your SoX

This finding allows us an opportunity to examine our library programatically. There is a command-line audio utility called “SoX” (for “sound exchange”) that we can use along with a scripting language to open files and examine their frequency content.

I decided to see if I could figure out how to use this utility to check out all our audio. The hope was that it would identify the songs that may have once been compressed so that we could examine these more closely and replace them if necessary.

I ended up with a Python script that loops through a folder with the audio files, opening each and using SoX to create a temporary file from the song after applying a high-pass filter at, say, 17 kHz. Then a second process takes the RMS amplitude value from this temporary file and compares it to a value previously discovered by experimentation. If below this nominal value, the file is flagged as a potential candidate for replacement.

These files can be examined manually with a program such as Audacity or Adobe Audition that offers a spectrogram view. The spectrogram can be examined and the file can be played in a controlled studio environment so that a determination can be made as to the need for replacement of the audio.

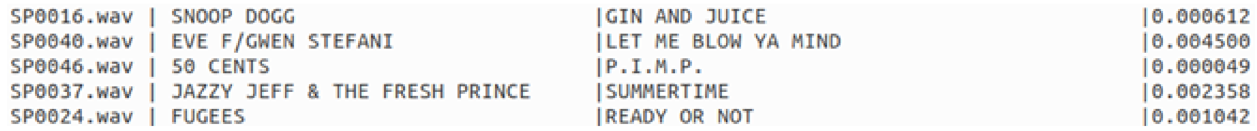
The line that creates the temporary high-pass-filtered audio file (filtered at 17 kHz) looks like this:

```
sox [original_filename] [output_file_name] sinc 17k
```

The code that does rest of the heavy lifting (determining the RMS value of the high-pass-filtered audio file) is a little scary looking:

```
sox output.wav -n stat 2>&1 | sed -n 's|^RMS amplitude:[^0-9]*\([0-9.]*\)$|\\1|p'
>>../rms.txt
```

All this really does is take a look at the temp file (output.wav) and call up the stats of the file. Then the sed program searches the resulting statistical output for the phrase "RMS amplitude" and writes the numerical value of that stat to a file called rms.txt. The rest of the code, not shown here, inserts the name of the song or audio file alongside the RMS value of that file. We end up with a list that looks something like Fig. 3.



```
SP0016.wav | SNOOP DOGG                      |GIN AND JUICE                           |0.000612
SP0040.wav | EVE F/GWEN STEFANI              |LET ME BLOW YA MIND                     |0.004500
SP0046.wav | 50 CENTS                        |P.I.M.P.                                |0.000049
SP0037.wav | JAZZY JEFF & THE FRESH PRINCE   |SUMMERTIME                              |0.002358
SP0024.wav | FUGEES                          |READY OR NOT                            |0.001042
```

Fig. 3

In this case, any song with a value below 0.001 is suspect. Subsequent inspection of those songs' spectrograms confirmed that they had a "flat top" at about 17 kHz, thus we know that, although they are WAV files now, they have likely been compressed at some point in the past. Those songs should be replaced with known linear audio.

It should be noted that at high bitrates such as 320 kbps, this method won't work as well because the frequency cutoff is close to 20 kHz.

If you'd like the complete Python script, email Radio World and I will send it along: radioworld@futurenet.com.