MASTER'S THESIS
COMPUTING SCIENCE

RADBOUD UNIVERSITY

Smaller Sound

Compression for FM MPX

Author
Mathijs Vos BSc.
m@thijsvos.nl

Supervisor
dr. David N. Jansen
dnjansen@cs.ru.nl

Company Supervisor
Hans van Zutphen MSc.
hans@thimeo.com

Second assessor
prof. dr. Jozef Hooman
hooman@cs.ru.nl

November 23, 2015

Abstract

Currently, no compression method is available that is suitable to cope directly with a multiplexed FM radio signal. Common audio codecs such as MP3 or Vorbis are not suitable because all information modulated into high frequencies (stereo, RDS) would be removed, since these codecs remove everything humans cannot hear.

In this thesis, we proposed a compression method suitable for FM MPX. It is capable of storing stereo audio and RDS (radio data system) information. We then performed an A/B/X test to find out if test subjects could distinguish the processed signal from the original signal. Using a bit rate of 320 kb/s, less than 25% of the test subjects could perceive a difference (confidence 95%). This makes the codec suitable for slower connections, such as T1 lines or satellite feeds.

Acknowledgements

Hans asked me: “is it customary to write ‘we’ in such a document?”. And yes, I believe it is. For I alone could not have completed this thesis. Not without the help of all people who contributed to it in some way.

So, first of all: thanks, Hans, for having many, many, many ideas. And sharing them with me so I could truly build a working codec that achieved impressive results (for me, at least). Thanks too for offering feedback and for putting your equipment at my thesis’ disposal.

Thanks David, for the fortnightly meetings where you gave lots of useful feedback, keeping my project on track and helping me to successfully complete it.

Thanks Jozef, for taking the time to assess my thesis.

The AB/X test could not have taken place without the help of Maarten Daam (N1), who provided access to the room we used as testing lab. Thank you!

All people who took the time to come to the testing lab and took part in the test, a warm thank you (I hope the liquorice and candy bars made up for the inconvenience).

Last but not least: thanks to Leif Claesson, for bringing *zure matten* to Hans’ office and for redecorating the place (which was much needed, sorry Hans). That helped a lot!

Contents

1 Problem statement

1.1 Introduction

While a lot of effort is put into popularising digital radio (e.g. DAB+) in Europe, the transition process is quite slow and in some countries digital radio seems to be reaching only a very small audience compared to FM (Kleinsteuber, 2006; Lax, Ala-Fossi, Jauert, & Shaw, 2008). In some countries the introduction of DAB+ has been put on hold for lack of popularity. This leads us to believe that FM-radio will still be here in the coming years.

The FM signal is usually not broadcast directly from the studio, but from separate transmitter sites some distance away. This means the audio signal has to be transported to one or more of these transmitter sites. A direct analog line is seldomly available, so instead the signal is transmitted digitally via a satellite connection or through a *Studio Transmitter Link*, which can be a leased network line or a wireless point-to-point connection. Wireless connections are error-prone and only useful if the distance between the transmitter site and the studio is relatively small (a maximum range of 15km is not uncommon). Older leased lines are not always fast enough to support transmitting an uncompressed signal (for example, T1 lines are quite common, providing a maximum data rate of 1.544 Mbit/s). Leasing satellite bandwidth is very expensive, so only slower connections are affordable. For example, Dutch radio station 538 (the #1 ranked station in august 2015) uses a satellite feed with a bandwidth of 384 kb/s to feed their transmitters.

Hence, a full signal containing all elements necessary for FM broadcasting cannot be transmitted directly to the transmitter site. Instead stereo audio is streamed to the transmitter site, using for example MP2 to compress the audio, lowering the bandwidth required to transmit the signal. Then, at the transmitter site, a full FM-signal has to be generated from the audio. This means using either a pc with a software-based FM processor (which might be undesirable because the transmitter site may be hard to reach in case something is wrong with the pc), or using a hardware-based processor (which might be undesirable because prices of $8000 or more for these devices are not uncommon).

Ideally, all necessary FM processing is performed directly at the source (the studio), eliminating the need for separate expensive processors at the transmitter site. In this case, the full FM signal would need to be streamed to the transmitter. This is only possible if we can lower the bandwidth required, by performing compression on the stream. Ideally, only a very light device would be needed at the transmitter site, to decompress the signal and maybe to perform some light processing on the resulting signal.

1.2 FM stereo multiplexing

So what's exactly needed for FM broadcasting? Only one signal can be modulated onto the FM carrier frequency. However, radio stations want to broadcast in stereo (requiring a separate left and right audio channel) and they want to send textual information and other metadata along with their signal as well (*Radio Data Signal* or RDS, e.g. containing information on traffic, the type of music played and alternative frequencies)[1].

To allow all this information to be broadcast using just one signal, the independent signals are multiplexed to one signal which can then be modulated onto a carrier wave. The structure of the multiplexed signal (*MPX*) intended for FM broadcasts can be found in table 1.

A radio receiving the FM signal first demodulates the MPX signal from the carrier wave. It then detects if a 19 kHz tone (the stereo pilot) is present. If this is the case, it generates a 38 kHz tone by doubling the frequency of the pilot. It then multiplies the $L - R$ signal with this 38 kHz tone to obtain the original $L - R$ signal in the 0-15 kHz frequency range. Older radios that do not support stereo broadcasts automatically fall back to mono output, as they play back the whole signal as if it were mono – the stereo information present in the high frequencies is simply not heard by the human ear, or even not reproduced by the loudspeaker.

[1]See European Committee for Electrotechnical Standardization, 1998

Frequency				Data
0	-	15	kHz	$L + R$ signal, mono (sum of left and right audio channel)
		19	kHz	Stereo pilot (tone indicating the presence of and providing a way to demodulate the L-R part)
23	-	38	kHz	$L - R$ signal (right audio channel subtracted from left channel)
38	-	53	kHz	$L - R$ signal, mirrored (may be absent if 'single side band' used)
~54.6	-	~59.5	kHz	RDS data (Radio Data Signal)

Table 1: The lay-out of a multiplexed FM signal. More data may be present above 60 kHz, but we ignore that for now.

1.3 Digital MPX

The multiplexed signal contains frequencies up to around 60 kHz (if stereo and RDS are used). This means that a sample rate of at least 120 kHz is required to reconstruct the analogue signal according to the Nyquist-Shannon sampling theorem (Shannon, 1949). In practice, often a sample rate of 128 kHz is used, providing some headroom. Assuming the bit depth is 16, this would lead to a required bandwidth of about 2 Mb/s if the signal is transported in an uncompressed form. For modern glass-fiber connections this is not a problem. However, for older, slower connections (such as T1 lines), or for expensive satellite feeds, this is too high a bit rate. Hence we want to perform some form of compression on the signal.

1.4 Existing codecs

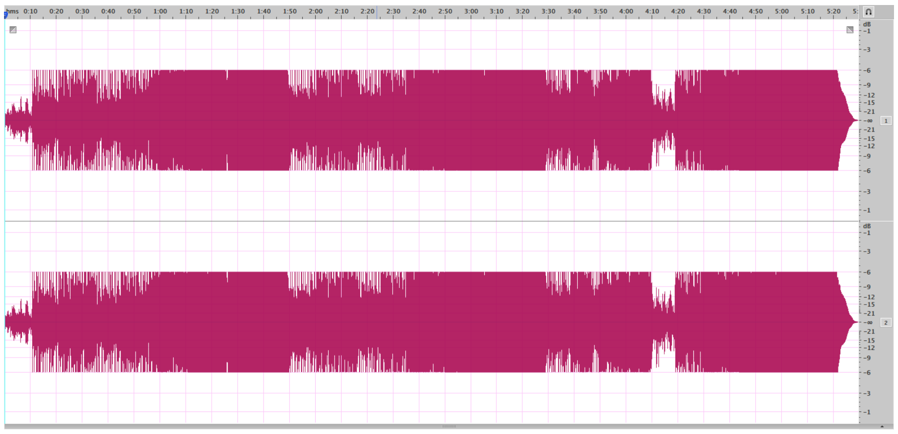
One way to compress the signal would be to use existing audio compression methods, such as the lossy codecs MPEG-1 Layer 2 (MP2), MPEG-1 Layer 3 (MP3) (e.g., Pan, 1995; Brandenburg, 1999) or Ogg Vorbis (Montgomery, 2015), or lossless codecs such as FLAC (Coalson, 2014). The compression reached with lossless codecs varies over time and highly depends on the input audio signal. This makes lossless codecs unsuitable for streaming, since no maximum bandwidth can be guaranteed.

The main problem with using existing codecs to code MPX is that they are optimized for the way the human ear works. For example, everything outside the hearing range of the human ear is thrown away (Pan, 1995; Brandenburg, 1999). This makes lossy codecs unsuitable to directly store an MPX signal: a lot of data is modulated above 20 kHz, which would all be discarded. Another problem is that it is not guaranteed that peaks in the signal before compression are at the same level as peaks in the signal after decompression (see figure 1). This may cause the signal to temporarily become too loud. If the signal that is too loud is then modulated onto a carrier wave, this will cause the FM signal to become too 'wide' (i.e. wider than the 75 kHz bandwidth allowed), which will not only harm the quality of signal reception, but is illegal as well.

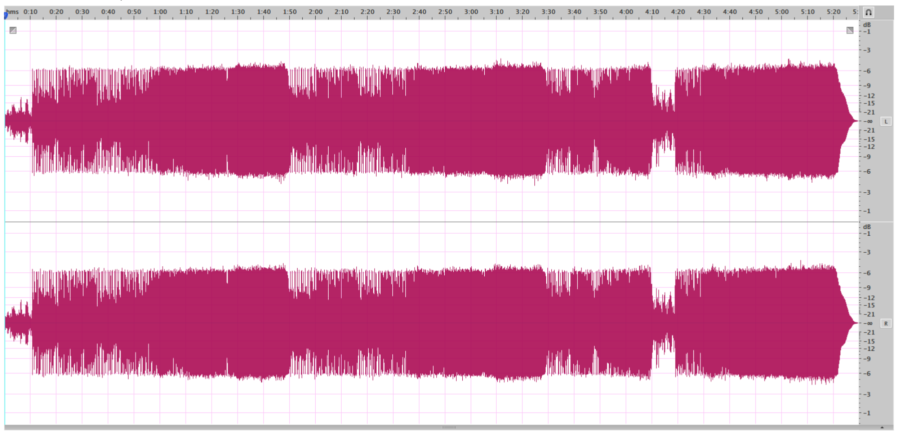
1.5 About this thesis

The goal of this thesis is to design, implement and evaluate an audio codec that can be used to compress FM MPX signals. In order to do this, we will first discuss the various existing techniques to compress audio signals (section 2). Then we will develop an algorithm to compress MPX (section 3), which will have the following characteristics:

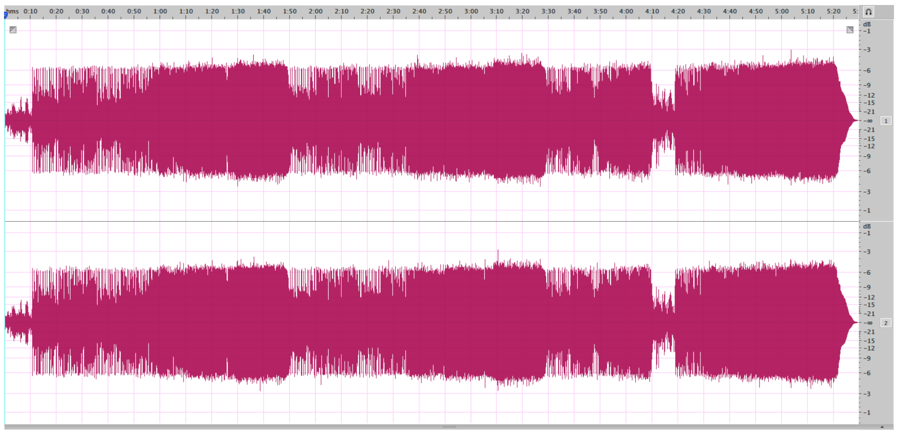
- It does not cause peaks in the audio signal
- It does store all data to be able to reconstruct the full MPX signal upon decompression, including RDS and stereo pilot
- It does result in a bit rate low enough to at least be able to transport the signal using a T1 connection, but ideally even lower (around the bit rate of a high-quality MP3-stream, 320 kb/s)



(a) The original, non-compressed audio Waveform (Red Hot Chili Peppers - Californication). Measured peak level: −06.02 dBFS



(b) The waveform of the same track, after compression with MP3 at 128 kb/s CBR. Measured peak level: −03.82 dBFS



(c) The waveform of the same track, after compression with Ogg/Vorbis at q4 (roughly 128 kb/s VBR). Measured peak level: −02.79 dBFS

Figure 1: Comparison of peak level after compression with various codecs

- It does allow for streaming audio with low latency (1 ms), although lower bit rates may mean higher latency and vice versa.

The compression will not be lossless, hence the audio quality will not be fully preserved. We will evaluate the compression method in terms of perceived audio quality (section 4).

2 Existing audio compression techniques

2.1 Lossless compression

Lossless compression of audio enables the exact reconstruction of the original waveform when decompressing the signal. Most audio files do not contain random noise but 'normal' sound (like music). This can be exploited in the compression process, as some assumptions can be made about the data. It is even possible to predict to some extent what comes next. One compression method that works in this way is for example SHORTEN (Robinson, 1994).

In SHORTEN, for every sample that is to be encoded, first a prediction is made about the value of that sample. A linear predictor is used, meaning that the predicted value of the sample is calculated as a linear combination of a given number of samples that were seen before. This is called the *prediction*-step. In the next step, called *residual coding*, the difference between this prediction and the actual value is calculated. The difference-values of all samples are further compressed using a Huffman-code. After compression a file size of about 42% compared to the original can be reached (Robinson, 1994). This means that –in theory– we could reach a bit rate of roughly 840 kb/s for the MPX-signal we started with.

One problem however is that this method cannot guarantuee a fixed bit rate. Also a bit rate of around 840 kb/s is still too high for what we want to achieve.

2.2 Lossy waveform-based compression

In the paper by Robinson (1994) a lossy version of SHORTEN is briefly described. This method allows for a fixed bit rate to be used. To achieve this fixed bit rate, the signal-to-noise ratio varies over time. The downside of this is that the quality of the encoded audio may vary a lot depending on the original audio signal.

With *Pulse Code Modulation* (PCM), audio is sampled at a fixed interval (e.g. for CD-audio the sample rate is 44.1 kHz). For each sample the value is stored with a fixed bit depth (16 bits per sample for CD-audio). A way of compressing audio is by using ADPCM (Benvenuto, Bertocci, Daumer, & Sparrell, 1986). Nowadays it is used mainly in telephony, but it is suitable to compress any form of audio.

With ADPCM, instead of the exact sample value, only the difference compared to the previous sample is stored. This in itself does not yet allow for compression, as this difference value may still be so big that a full 16-bit integer is needed to store it. However the number of bits per *delta-value* may be reduced by storing an approximation of this value instead of the exact value. With μ-law or A-law coding as used in telephony (Benvenuto et al., 1986) only four bits per delta-value are used. To still be able to cover the full range a logarithmic scale is used to map the four bit values to a larger range. This does cause a problem: this method of compression may cause peaks, because the delta-value is approximated and the value reconstructed upon decompression may be larger than the actual value.

Both ADPCM and lossy SHORTEN are waveform-based compression methods. This means that they store an approximation of the waveform, without taking the auditive perception of human beings into account.

2.3 Lossy perception-based compression

While waveform-based compression methods often try to reduce the bit rate by trying to efficiently code samples in the time-domain, other codecs exist that operate on the frequency domain. Popular lossy codecs such as MP3 (e.g., Pan, 1995; Brandenburg, 1999) and Ogg Vorbis (Montgomery, 2015) are based on this principle. One advantage of operating on the frequency domain, is that these codecs are able to take certain properties of the human auditive system into account. The human ear is more sensitive to detail in some frequency ranges than in others, so to save data some frequency ranges can be stored with less precision. Usually these codecs are based upon some model of the way the human ear responds to sound, and they exploit its properties to throw away data or to store data with less precision.

The lossy perception-based codecs mostly use the Modified Discrete Cosine Transform (e.g. as in Pan (1995)), Discrete Wavelet Transform (as in Sinha and Tewfik (1993)) or a combination of the two (as in Sinha and Johnston (1996)) to transform data in the time domain (samples) to data in the frequency domain (bins) where the data is then compressed in some way. However, this method of compressing audio is meant to keep the audio *perceptually* equal to the original signal. In practice this means that while the audio may sound the same to the human ear, the waveform is not kept intact at all, and often peaks are introduced due to the way the compression works.

2.4 Overview

An overview of basic audio compression techniques is given in table 2. These techniques are used in popular audio codecs, such as MPEG-1 layer 3, Ogg/Vorbis and FLAC.

Psycho-acoustic modeling forms the basis of lossy codecs such as Ogg/Vorbis. In combination with filter banks it allows for a high compression ratio without introducing too many audible artifacts. In this thesis, however, we will not use either of these two methods. The main reason is that when using this method for compression, an approximation of the audio data is stored in the frequency domain. This makes it impossible to guarantee no peaks occur in the signal upon decoding.

3 Building the algorithm

What follows is a description of the compression principles we selected and an explanation of how we got to this solution. We combined several existing techniques together with ideas of our own. We discuss technical properties of the algorithm where we think this is relevant. Further evaluation of the sound quality is done in a subjective way, this is described in section 4. A scheme describing the global data streams within this algorithm can be found in figure 2.

3.1 Online or offline

Most codecs available nowadays are designed to be used in an *offline* way (i.e. all audio data to be compressed is already available at the beginning of the process). Some can be used *online* (i.e. in audio streaming applications), however in this case the latency is often high as the algorithm is designed to be used on blocks of data of a fixed size (introducing a minimum delay).

Our goal is to create a codec that allows for very-low-delay streaming. As such we attempt to design the largest part of the codec such that it works on individual samples instead of on blocks of samples. However, it seems infeasable to transfer the samples one-by-one over a network connection (because the relative overhead of e.g. metadata would become very large and because the signal quality would be heavily influenced by network jitter). Dividing the data into blocks thus is inevitable at some point in the process. The stability of the stream could be improved further by maintaining a small buffer on the decoding end, however this does introduce extra latency.

3.2 Samplerate

Most lossy codecs reach the same bitrate regardless the samplerate of the original file. Take for example a file containing audio with frequencies up to 20 kHz. When the file is sampled at 48 kHz, the bitrate after compression with a lossy codec such as MP3 will be the same as when compressing the same file sampled at 44.1 kHz. This is easily explained by the way these codecs work. They store frequencies rather than samples. Thus only the frequencies present in

Name	Description	Used in
Filter banks	A filter bank is used to divide the input audio signal into multiple frequency bands. Since the human ear is more sensitive to some frequencies than to others, the frequencies the ear is less sensitive to can be stored in a less precise way while still not causing audible differences.	Most perception-based lossy codecs, such as the Modified Discrete Cosine Transform in MPEG-1 Layer 3 (MP3) (Pan, 1995), Ogg/Vorbis (Montgomery, 2015) and the proposal of Princen and Johnston (1995), or Discrete Wavelet Transform in the proposal of Sinha and Johnston (1996).
Psycho-acoustic modeling	To determine what data needs to be stored more precisely to minimize the audible distortion, an algorithm exploiting a psychoacoustic model that describes certain properties of the human auditory system can be used.	For example MP3 (see Pan, 1995) and Ogg/Vorbis (Montgomery, 2015), where it is used on top of filter banks.
Entropy coding	In order to reduce the number of bits needed to store the audio data, entropy-coding is used to get better compression. Well-known entropy-coding methods include Huffman coding (Huffman et al., 1952) and arithmetic coding (Moffat et al., 1998).	Used in both lossless and lossy codecs (a.o. FLAC (Coalson, 2014) and MP3).
Intra-channel decorrelation	Instead of storing each channel separately (left and right in stereophonic audio), higher compression can usually be achieved by taking the correlation between channels into account. Methods include mid/side stereo (lossless) and joint stereo (lossy).	E.g. MP3 supports dual channel, mid/side stereo (L+R/L−R) and joint stereo (exploiting the (in-)accuracy of the localization of certain frequencies by the human ear). FLAC uses mid/side stereo to preserve losslessnes.
Predictive modeling	When using PCM the value of each sample is stored directly. To achieve better compression, a predictive model could be used. The model predicts the value of the next sample to come, and only the difference between the prediciton and the actual value is stored. Most of the time, this will require less bits than directly storing the sample values.	Described in Robinson (1994) and Hans and Schafer (2001) and for example used in FLAC. Also used in some more advanced variants of ADPCM.

Table 2: A selection of basic audio compression principles

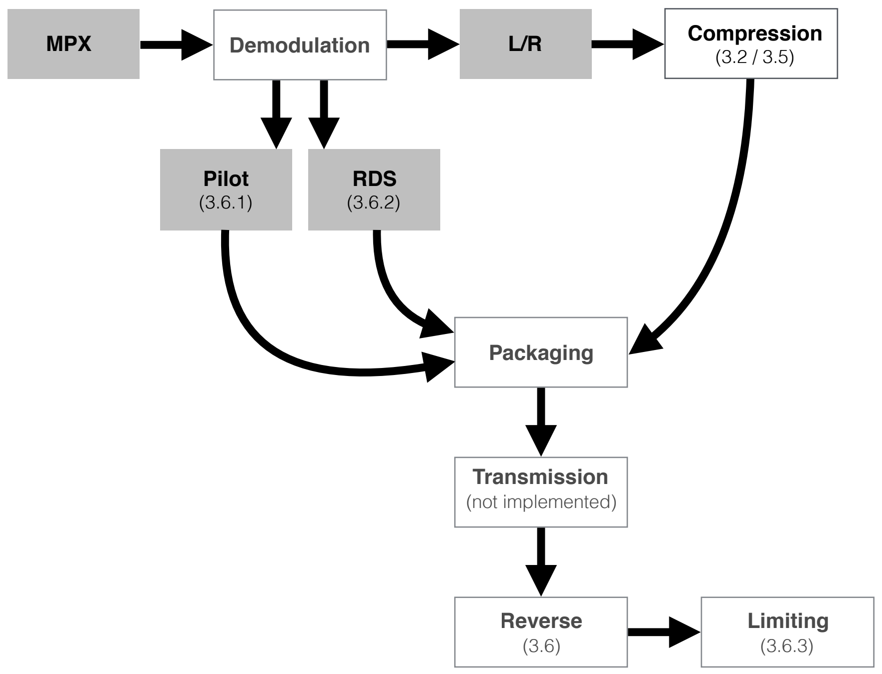


Figure 2

the signal are relevant, and the samplerate does not have a direct influence (though MP3's block size is determined by it).

However, the same property will not hold for our codec. We will store samples, not frequencies. Because of this a file with a lower samplerate will result in a lower bitrate and vice versa. FM audio can contain frequencies up to roughly 16 kHz. Therefore it makes sense to downsample the audio before performing any other type of processing. To store all frequencies accurately and to leave some headroom we will use a samplerate of 36 kHz. This will save data, though a downside is that resampling results in extra latency. How much depends on the method of resampling: just six samples for linear interpolation (0.03 ms at 192 kHz), and up to 4096 samples if a Fourier transform is involved for pre- or post-filtering (21.3 ms at 192 kHz).

3.3 Prediction

A technique used by various lossless compression methods is *predictive modeling*. A model is used to predict the value of the next sample or samples, given a number of previous samples. This technique is also called *Linear Predictive Coding* (e.g., Makhoul, 1975).

Prediction in its simplest form is assuming the next sample has the same value as the last. In order to be able to reproduce the true value of the second sample, only the difference with the first sample is to be saved. For example, if the first sample has value 100, and the second sample has value 120, for the latter only 20 need be saved to get back to the original value of 120. This prediction method can be further generalised to save the differences of differences or even the differences thereof. However at some point the differential value needs more bits to save than the original value. In practice it does not make sense to go beyond 3$^{\text{rd}}$ order differences (Craven & Gerzon, 1996).

A generalisation of the 'previous sample predictor' is the FIR (finite impulse response) predictor. With this type of predictor, each sample is predicted as being a linear combination of past samples:

$$\sum_{i=1}^{N} a_{-i} \cdot s_{-i}$$

where s_{-1} is the previous sample, s_{-2} is the sample before that, etcetera; and a_{-1} is the constant to multiply the previous sample with, a_{-2} the constant to multiply the sample before that with and so forth. Different sets of values for a could be used, choosing a set that best reflects the audio to encode.

The FIR-method is easy to understand, easy to implement and has a low computational cost as only linear calculations are involved. It is used in for example SHORTEN and AudioPAK (Robinson, 1994; Hans & Schafer, 2001). Robinson (1994) claim that in practice higher-order FIR predictors do not predict significantly better than second-order FIR predictors. Keeping computational complexity in mind, they suggest using second-order FIR predictors, as the compression level rises only slightly when using higer level FIR's or IIR's (Infinite Impulse Response).

Craven and Gerzon (1996) propose to use IIR predictors instead, as these are capable of more accurately predicting acoustic signals compared to using a FIR predictor. Especially if the material to compress contains a lot of high frequencies it makes sense to prefer an IIR above a FIR predictor. IIR's can model audio data containing high frequencies and lots of dynamics. FLAC is a popular lossless compression method that uses IIR prediction models (Coalson, 2014).

An IIR predictor is more tricky to implement, as the constants to be used in the calculation will have to be updated "on the fly" in order to keep the prediction accurate. This might also lead to a high computational cost.

Although the claims of Robinson (1994) about the performance of FIR compared to IIR seem to contradict those by Craven and Gerzon (1996), we decided to start with the simpler FIR predictors to find out how they perform ourselves.

3.3.1 The SHORTEN-predictors

In theory, the predictors mentioned by Robinson (1994) and Hans and Schafer (2001) result in a residual signal with a flat frequency response, meaning that only noise remains intact. This makes sense, as noise consists of random samples and thus is unpredictable.

The four predictors of SHORTEN are as follows (s_0 being the prediction value):

$$s_0^1 = 0 \tag{1}$$

$$s_0^2 = s_{-1} \tag{2}$$

$$s_0^3 = 2 \cdot s_{-1} - 1 \cdot s_{-2} \tag{3}$$

$$s_0^4 = 3 \cdot s_{-1} - 3 \cdot s_{-2} + 1 \cdot s_{-3} \tag{4}$$

In both the SHORTEN and AudioPAK algorithms, the input audio is divided into blocks of a certain length. Then, each predictor is used on that block, and the predictor leaving the smallest residual signal is chosen. Then the predictor number is stored with each block, so that upon decoding that same predictor can be used to reconstruct the original signal again. A visualisation of the behaviour of these predictors can be found in figure 3.

For our purpose however, this is an unsuitable approach. First of all, due to the part of the algorithm that divides the audio into chunks, a minimum latency is introduced. This could be solved by reducing the block size to a small number of samples. However, this would cause the required bandwith to increase, as per block the predictor has to be stored, and the smaller the block size becomes, the larger the relative overhead of storing the predictor will be.

The second problem is that chunking the audio may lead to audible distortion. SHORTEN was meant as a lossless codec, so upon decoding the exact same samples as in the original stream are reconstructed. We are not trying to build a lossless codec, and thus the samples will

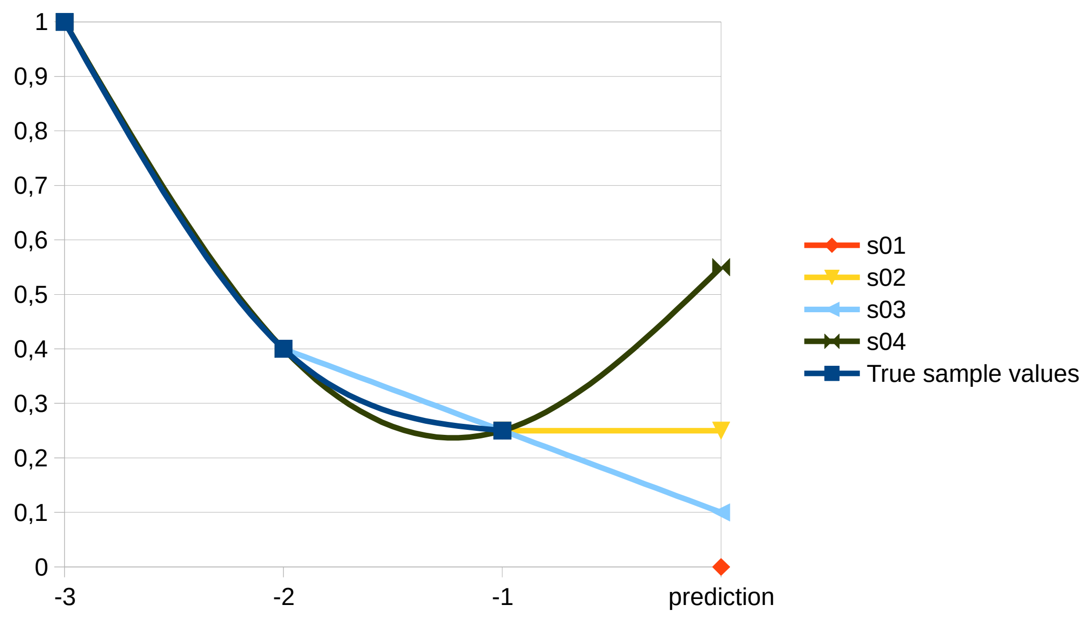


Figure 3: Illustrated behaviour of the four linear predictors.

not be reconstructed exactly: they will be an *approximation* of the original sample values. By choosing different predictors for each block, sample values will be approximated in a different way. Each block may sound differently depending on the predictor chosen. A similar problem occurs in the widely-used codec MP3, where the precision of the audio stored varies per block of 20ms (Brandenburg, 1999). This results in audible artifacts referred to as pre-ringing and post-ringing.

3.3.2 Predicting per sample

To mitigate these problems, we use a different method of predicting samples (but using the same prediction formulae as in SHORTEN). Assume at least four samples have already been encoded by our algorithm. A fifth sample s_5 then enters our system and needs to be encoded. For each predictor, we calculate the predicted value of sample s_4 using the values of previous samples s_3, s_2 and s_1. The predictor that outputs the value closest to s_4 is deemed the best predictor. We then use this predictor to predict the value of s_5, using in our calculation the values of samples s_2, s_3 and s_4. This way, we do not need to store the chosen predictor at all, because in the decoding stage the same deterministic choice can be made as we only choose based on historical data. Furthermore: it is not necessary to do chunking of audio, as we can do this prediction per sample instead of per block.

3.3.3 A better predictor

The SHORTEN prediction method works reasonably well for lower frequencies. On signals containing high frequencies, the prediction becomes less and less accurate. When used in a lossless codec, this may influence the bit rate, but when it is used in a lossy codec, it directly influences sound quality. Ideally, the predictor performs equally well across the whole spectrum, not only on the lower end.

Keiler, Arfib, and Zölzer (2000) present multiple methods to dynamically calculate the coefficients used in a FIR calculation. All methods depend on previous samples, for which a small buffer needs to be maintained. Since in our case the previous samples are rounded based on previous predictions, using a FIR with dynamic coefficients effectively gives us an IIR predictor.

We have opted for the autocorrelation method, as this method has an algorithmic complexity of $O(N)$ with N being the size of the block the prediction is performed on (Keiler et al., 2000).

With this method, for a block of length N an approximation of the autocorrelation sequence is calculated by

$$R(i) = \frac{1}{N} \sum_{n=i+1}^{N} u(n)u(n-i)$$

Where $u(n)$ is sample s_n with a Hamming window applied to it. Then, to obtain the coefficients a_i for $i = 1, \ldots, p$ for a predictor of order p, the following equations are to be solved:

$$\sum_{k=1}^{p} a_k R(i-k) = R(i)$$

for $i = 1, \ldots, p$ (Keiler et al., 2000). The solution to this can be calculated efficiently using e.g. Levinson-Durbin recursion (see e.g. Makhoul, 1975; Levinson, 1949; Durbin, 1960). Then, the next sample is predicted:

$$s_{N+1} = \sum_{N-p+1}^{N} a_k s_k$$

3.3.4 Downside of prediction

A downside of using predictors per sample is that this codec will not be able to (partly) reconstruct packets in case of network packet loss. This can be solved partially by starting each block with three "full" sample values instead of difference values, in the case of the FIR predictor. This way the internal state of the predictors can be safely reset at the beginning of each block. For the IIR predictor this would not work, in this case p full samples would be required to accurately restore all sample values. Or the predictor could be reset at the beginning of each block.

The impact of packet loss can also be diminished by sending the data for the left and right channel in separate packets, so that the decoder is able to temporarily switch to mono audio in case one of the packets is missing or arrives too late. On the downside, this will lead to a higher latency as the decoder has to wait until both the packets for left and for right are received before it is able to reconstruct the signal. We currently have not implemented any of these methods, as our implementation is not yet suitable for online usage.

3.4 Residual coding

After the value of a sample is predicted, the difference between the prediction and the actual value is stored. However, the value will not be stored exactly: only an approximation of the value will be stored. This is where the algorithm becomes lossy. The question is how to determine what a good approximation of a difference value is. The value needs to be accurate enough so that no audible distortion is introduced, but it should not be too precise as then more data would be needed to store the value.

3.4.1 Only noise

The basic assumption is that the prediction step produces a residual signal with a flat frequency response, meaning that every frequency is present with more or less the same power as other frequencies (white noise). Instruments causing noise-like sounds in music are for example crash cymbals and hi-hats.

When rounding difference values we are essentially adding noise to the signal, as each sample will be reconstructed with a slight deviation compared to the original signal. However, we assumed the predictors perform the worst on noise-like sounds and perform the best on clear, pure sounds. The basic idea is thus to store a difference value very accurately when the predictors are giving accurate results (small difference values, pure sound) and to store the value with a rough approximation when the predictors yield results that deviate a lot from the original signal (large difference values, noise-like sound). This way noise will mostly be introduced in sounds that were already noise-like. We expect that this type of noise will not be audible.

3.4.2 Determining the prediction quality

When the deviation between the values predicted and the actual values is small, the predictors produce accurate results. High differential values indicate a poor predictor performance. To estimate this performance, we keep track of the average absolute deviation. Calculating a true average or mean would introduce the need for quite a lot of computational power. Therefore we choose to approximate the root mean square of the differences:

$$\boxed{\begin{gathered} q = \sqrt{\alpha\Delta^2 + \beta q^2} \\ \alpha + \beta = 1 \end{gathered}}$$

We empirically determined that using $\alpha = 0.1$ and $\beta = 0.9$ yields good results.

3.4.3 Calculating the step size

The differential values are rounded to a nearby value that can be efficiently stored. To determine the value to round to, first a step size δ is calculated. The value q obtained in the previous step is first multiplied with a scaling parameter w. The smaller the value of this parameter, the more precise the sample difference values can be stored because less rounding occurs. Choosing a higher value for this parameter leads to more rounding (and thus probably to a lower audio quality), but might reduce the bitrate because the number of possible results is smaller.

$$\boxed{\delta = q \cdot w}$$

Then the differential value will be rounded to $n \cdot \delta$ with $n \in \mathbb{Z}$ chosen such that the rounded value lies as close as possible to the original value. The value that is then actually stored is n.

Important to note is that this same method can be used on the decoding side without the need for sending extra information. The process entirely relies on the prediction quality of past samples which is known on both sides. Because of this the approximated value can be restored from n.

In our implementation we choose w for each block such that the bit rate remains roughly constant. If the size of the previous block is too large (or if it is very close to the maximum), we can use w to temporarily lower the audio quality in order to make sure the current block will be smaller than the allowed maximum. If constantness of bit rate would not be a problem (in case the coding is done offline) then w could be fixed, which would result in a more constant audio quality.

3.5 Entropy coding

The resulting rounded values from the previous step are entropy coded in order to save bits. Although in a worst-case scenario n can still be quite large (in theory it could still require 16 bits to store), this is very unlikely. Due to the way n is calculated, it is likely to be a small number (if the predictors would always predict the next sample with an accuracy of 100%, n would always be 0). Therefore we can save bits by using a short bitcode for small and likely numbers and a long bitcode for large and unlikely numbers.

To achieve this we implemented Huffman coding (Huffman et al., 1952). In order to reach efficient compression with Huffman, it is important to choose the weights assigned to each entry in the tree correctly or a sub-optimal coding tree is generated. One way to determine these weights would be to keep track of a histogram and to directly use the sample occurence of the previous block as input to the Huffman algorithm. This way however sudden changes in values cannot be coded efficiently (which occur for example when for some reason the predictors do not predict accurately). The size of the block is constant. We used a size of 128 samples per channel per block.

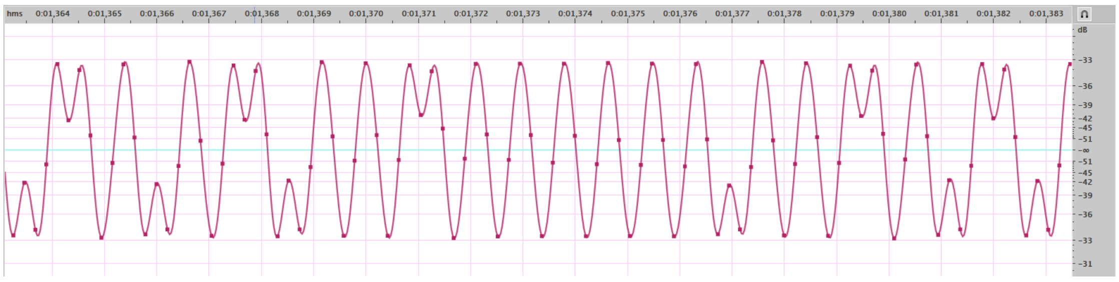


Figure 4: Demodulated RDS signal, resampled to 4750 Hz ($\frac{19}{4}$ kHz)

To solve this we have chosen to use an approximated histogram of the *current* block, where the approximation is sent along with the other data towards the decoding site. The approximation consists of the first 2^b entries of the histogram, where each entry $-x$ is summed together with entry x (the entries after entry 2^b are assumed to have a very small chance of occuring). Parameter b is fixed and can be used to make a trade-off between bit rate and precision of the Huffman histogram. With a total bit rate of 320 kb/s, we used $b = 4$. The entries in the histogram are scaled to fit in 4 bits, leaving a relatively small overhead while still a good approximation of the optimal Huffman code is obtained.

3.6 Back to MPX

Thus far we only described the coding method to code 'normal' audio. In order to be able to generate a full FM MPX signal upon decoding, more information is needed than just the audio signal. Both the stereo pilot and RDS data are needed to reconstruct the full signal.

3.6.1 Stereo pilot

The stereo pilot is a tone with a frequency of 19 kHz, $\pm$ 2 Hz according to the standard (European Committee for Electrotechnical Standardization, 1998). Because of this small deviation allowed, we cannot assume that the tone has an exact frequency of 19 kHz. The easiest way would be to transfer the pilot as samples, but this would require a sample rate of 38 kHz. Because the samples will be highly predictable, these samples can probably be compressed quite efficiently.

However the most efficient way to transfer the pilot correctly would be to send one number providing information on the phase of the pilot. This is the method we have chosen to implement. Using a phase locked loop or PLL (see e.g. Hsieh & Hung, 1996) the phase of the pilot is tracked at each input sample. The phase of the last sample in a block is stored and transmitted to the decoder. The decoder assumes a pure tone of 19 kHz, but makes small adjustments to reach the correct phase at the end of the block. This way the pilot can be reconstructed accurately without sending large amounts of data.

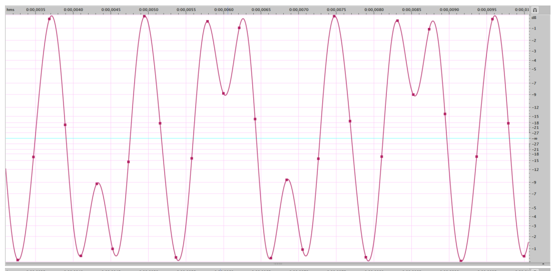
3.6.2 RDS

Multiple methods can be thought of to compress RDS data. One way would be to decode the signal to obtain a bitstream, which can then be turned into a waveform again upon decoding. However different RDS encoders may cause different signal characteristics depending on how the RDS standard was interpreted. When the data is decoded to a bitstream, these characteristics are not reconstructible by the decoder. The result may be that the RDS waveform is slightly different compared to the original, which may cause large deviations in the MPX waveform.

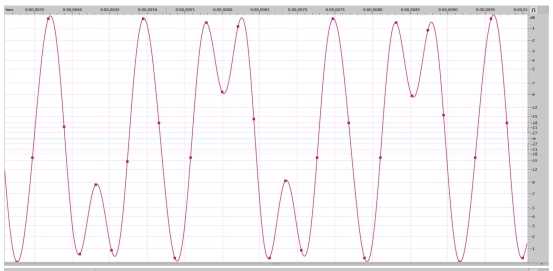
When RDS is demodulated from the MPX data, what remains is a highly repetitive signal (see figure 4). A similar approach as we use to compress audio could be applied to the RDS stream, as the signal is easily predictable (when downsampled to exactly $\frac{19{,}000}{4}$ Hz the number of different sample values is relatively low and thus they can be stored precisely with just a few bits).

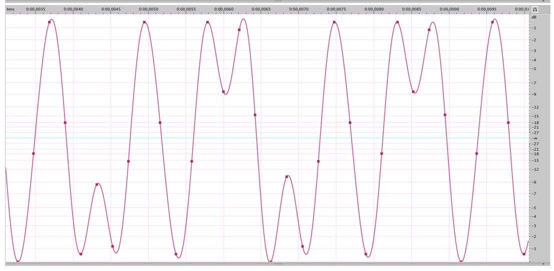
However an even simplier method is possible. The volume of the signal is relatively low: usually it equals around 4.5% of the total MPX volume. Since the maximum volume is known,

(a) Original RDS, demodulated from MPX, resampled to 4750 Hz and normalized to 0dBFS

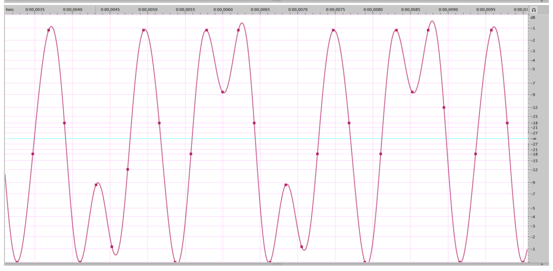


(b) Original RDS, downscaled to 6 bit





(c) Original RDS, downscaled to 5 bit



(d) Original RDS, downscaled to 4 bit

Figure 5: Comparison of RDS signal using different bit depths.

a smaller number of bits can be used to store the samples of the data signal. We expect some rounding of sample values will not hurt the signal quality, as the phase remains intact. As can be seen in figure 5, at 4 bits per sample the waveform starts to get more distorted. Though this still might be good enough to guarantuee good signal reception, to be on the safe side we choose to use 5 bits per sample.

Furthermore the recommended bandwidth for RDS is $\sim$ 2.0 kHz (European Committee for Electrotechnical Standardization, 1998), which means we need a minimum sample rate of $\sim$ 4.0 kHz. Because the RDS data is synced with the 19 kHz pilot tone, it makes sense to choose $\frac{19000}{4} = 4750$ Hz as sample rate for the data signal. Sending a signal with 4750 samples per seconds and 5 bits per sample leads to an extra data stream of $\sim$ 23 kb/s, which we think is acceptable.

3.6.3 Clipping the composite signal

So far the algorithm still cannot guarantuee no peaks will occur in the final signal. Small deviations in sample values for the left or right channel might cause relatively large peaks in the composite signal. For example, assume a sample value of 0.5 for the left channel and a value of 0.5 for the right channel, leading to $L + R = 1.0$ and $L - R = 0$ and $(L + R) + (L - R) = 1.0$. Say these values are then rounded to 0.6 and 0.4, respectively (the actual rounding depends of course on the internal state of the predictors). $L + R$ then still is 1.0, but $L - R$ now has the value 0.2 and $(L + R) + (L - R)$ is now 1.2 which is a disallowed peak. In practice these peaks might be very small or very large, depending on the point in time as $L - R$ is multiplied with a 38 kHz tone.

Sample values change just a bit when they were part of a sound that was easy to predict. When the prediction starts getting less accurate, sample values change more and more. The predictors are – at least in theory – capable of predicting pure tones very precisely, but start performing worse when the sound gets more noise-like. Therefore sample values get rounded more and thus cause peaks when the samples are part of noise. However especially in sounds that are already noisy, a bit more noise is probably not easily detectable by the human ear.

The signal can thus be fixed by simple clipping of the samples outside the allowed range, without audible distortion. To completely eliminate possible distortion in the spectrum that clipping might cause (which also might clutter the frequency spectrum and might thereby harm reception quality), we implemented a simple limiter. The limiter holds a small buffer and starts

Step	Description	Delay
Demodulation	Fourier transform – for best precision with block size of 4096 samples	$\frac{4096}{192000}s = 21.33ms$
Downsampling	Depends on resampling method – if filtering is applied, the same block as in the previous step can be used, giving no extra latency. However at least six samples are needed at 192000 kHz before the next sample at 36000 kHz can be calculated	$\frac{6}{192000}s = 0.03ms$
Prediction	Depends on predictor order – we used 15	$\frac{15}{192000}s = 0.08ms$
Packaging	Depends on the block size chosen - the smaller the block size, the larger the relative overhead. We used 128	$\frac{128}{192000}s = 0.66ms$
Limiting	Depends on limiter look-ahead – we used 96	$\frac{96}{192000}s = 0.50ms$
	Total	$22.61ms$

Table 3: Estimated minimum delay of the coding method

to temporarily lower the volume (both backwards and forwards) when a sample lies outside the allowed range. The result is a clean signal that conforms to the specification without having introduced extra audible noise.

4 Evaluation

4.1 Technical evaluation

4.1.1 Prediction

We performed a small-scale technical evaluation, comparing the two types of predictors: the SHORTEN FIR-predictors, and the more advanced prediction method by Keiler et al. (2000). We compared the predictors by feeding them a 20 second logarithmic sweep from 20 Hz to 15 kHz. We measured the absolute difference between the predicted value and the actual value. The results can be found in figure 6. The curve displays the absolute deviation between predictor and true value, measured in dBFS. As we suspected earlier, the more advanced IIR predictor is more stable, in the sense that its performance across the spectrum is more constant than that of the FIR predictor.

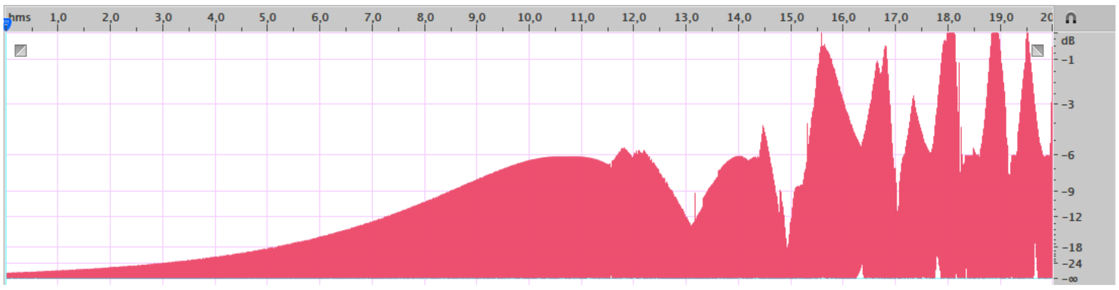
4.1.2 Delay

With our current implementation, the codec is not suitable for online use. We have used libraries for e.g. resampling that were meant to be used offline. These libraries use relatively large buffers internally, causing a high delay when used online. The total delay of our current implementation is close to 5 seconds. However, the coding method does not impose a minimum delay on its own – the delay is mainly caused by these libraries. Replacing them with a version suitable for online usage should be easy and may reduce the delay drastically. An estimation of the theoretical minimum delay can be found in table 3.

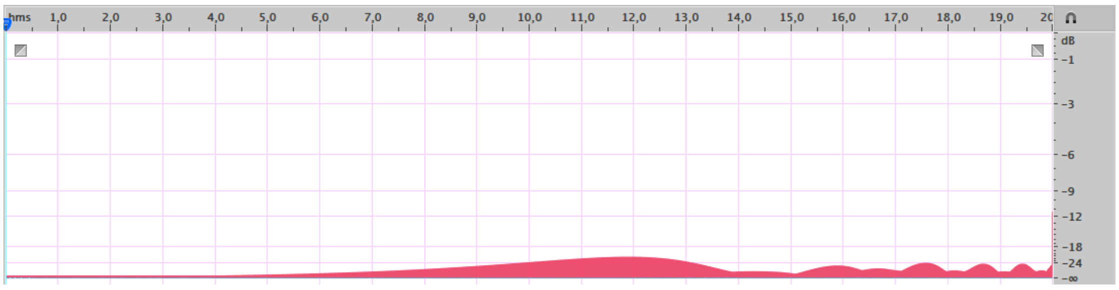
The delay is mainly caused by the demodulation and resampling stages. The demodulation stage could be replaced with one that does not require fourier transforms (a method emulating the way analog car radio's demodulate the signal could be used). The delay of the resampling stage depends on the method of resampling. A lower quality resampler might still give acceptable results but less delay.

4.2 Subjective evaluation

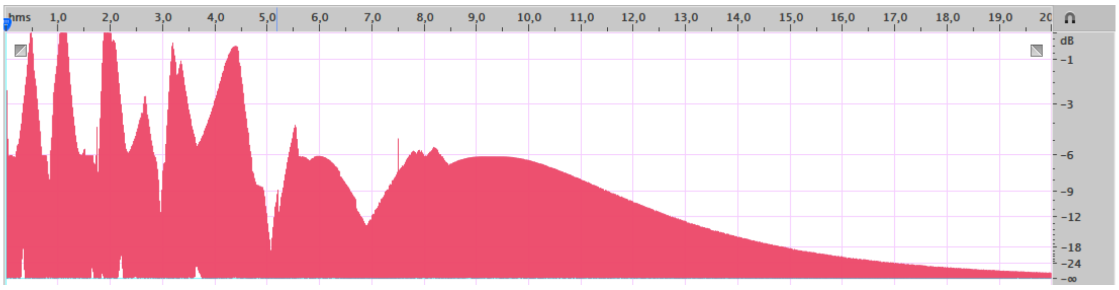
Of course we can analyse the performance of our codec in various technical ways (measuring the amount of noise introduced, for example). But this does not directly tell us anything on how the result of the codec is perceived by humans. If a lot of noise is introduced, but a sufficient amount of other sounds is present, it might be the case that the noise is not perceived, or the



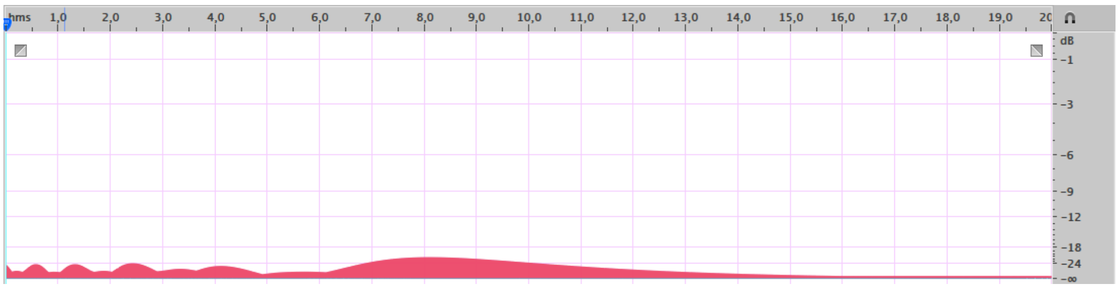
(a) Absolute deviation between log sweep and value predicted by FIR



(b) Absolute deviation between log sweep and value predicted by autocorrelation-IIR



(c) Absolute deviation between reversed log sweep and value predicted by FIR



(d) Absolute deviation between reversed log sweep and value predicted by autocorrelation-IIR

Figure 6: Comparison of predictor performance on a log sweep. A lower value indicates a smaller deviation from the real sample value. The X axis represents time.

noise *is* perceived but is not annoying. To find out how well the codec performs subjectively and to find at which bitrate it becomes 'transparent', we will perform listener tests.

4.2.1 Test method

The subjective evaluation should be double-blind so that neither the test subject nor the researcher know in advance which sample (as in fragment, not as in measurement of the sound wave) the subject is listening to. An often used method is the so called A/B/X test (Clark, 1982). In an A/B/X test, a listener is presented three samples: A, B and X. One of the samples A or B is the sample that was processed with our codec, the other one is the original, lossless signal. The third sample X is randomly chosen to be the exact same as A or to be the exact same as B. The listener is then asked to make the decision "X is A" or "X is B". If the test subject can truly distinguish A and B, they will anser the question correctly most of the time. If the subject is guessing (they cannot hear the difference between A and B), we expect they answer 50% of the samples correctly. In our test, we have chosen to fix A to be the unprocessed sample, where B was the processed sample.

4.2.2 Hypothesis

Because of the limited number of trials per subject and per track, we can not say anything useful about the results of one particular subject, nor of the results of one particular track. However to get a first impression of how well the compression method performs in terms of subjective quality, we state the following:

Hypothesis. *Less than 25% of the test subjects can perceive a difference between the unprocessed, received FM signal and the processed, received FM signal.*

4.2.3 Samples used

We picked music samples semi-randomly from music broadcast by customers of *Stereo Tool*:

- Arrow Classic Rock (Dutch internet / DAB radio station)
- N1 (Dutch public local radio station)
- Sky Radio Hits (Dutch internet radio station)
- Sky Radio 80s (Dutch internet radio station)

The samples were selected such as to cover a wide set of musical genres. A full list can be found in table 4.

Aditionally, we used a recording of the *NOS Journaal*, which is a Dutch news bulletin provided by national news organization and broadcaster *NOS*. All samples were created from lossless sources and thus are of CD-quality originally. They where then processed for FM broadcasting, using Stereo Tool with preset *Analog Pleasure FM* (as used by N1). An RDS stream was present in all FM samples, but was not used in the test.

We then compressed the samples using two different bit rates: 256 kb/s and 320 kb/s. The compressed files were decompressed again to obtain a file suitable for FM broadcasting. Then all files (including the 'original' FM samples) were fed to an FM transmitter running a low power setting. The signal was recorded using an USB-based receiver and the tool SDR#. The distance between the transmitter and the receiver was roughly 2 meters, allowing for good reception despite the low power setting of the transmitter. A picture of the recording setup is found in figure 7.

For usage in our A/B/X tool, the samples needed to be perfectly aligned to allow for seamless switching between A, B and X. To achieve this we converted the pieces to a high samplerate (192 kHz) using Adobe Audition set to the highest quality resampler available. They were then aligned and converted back to 44.1 kHz, again using Adobe Audition with the highest quality resampler.

Artist	**Title**	**Genre**
AC/DC	Highway To Hell	Hardrock, Bluesrock, Heavy metal
Axwell & Ingrosso	Sun is Shining	Progressive House
Bee Gees	Night Fever	Pop, Disco
Ben Gold	Where Life Takes Us	Trance
Creedence Clearwater Revival	Down On The Corner	Roots rock
Daft Punk	Get Lucky	Disco, Funk
Dire Straits	Sultans of Swing	Roots rock, Pub rock, Blues rock
Ed Sheeran	Thinking Out Loud	Soft rock, Blue-eyed soul
Fleetwood Mac	Everywhere	Soft rock
Jay-Z	Empire State of Mind	Hip hop
Klangkarussell	Sonnentanz	Nu-jazz, Deep house
Mark Knopfler	What It Is	Roots rock, Folk rock, Blues
Muse	Starlight	Alternative rock, Space rock, Progressive rock
NOS	Journaal	News bulletin
Owl City	Fireflies	Synthpop
Shaggy	Angel	Reggae fusion
DNA & Suzanne Vega	Tom's Diner	Trip hop, Electronica, New jack swing
The Script	Superheroes	Pop
Toto	Africa	Soft rock
Van Morrison	Brown Eyed Girl	Rock, British R&B

Table 4: List of fragments used in the A/B/X test. Genres according to Wikipedia.

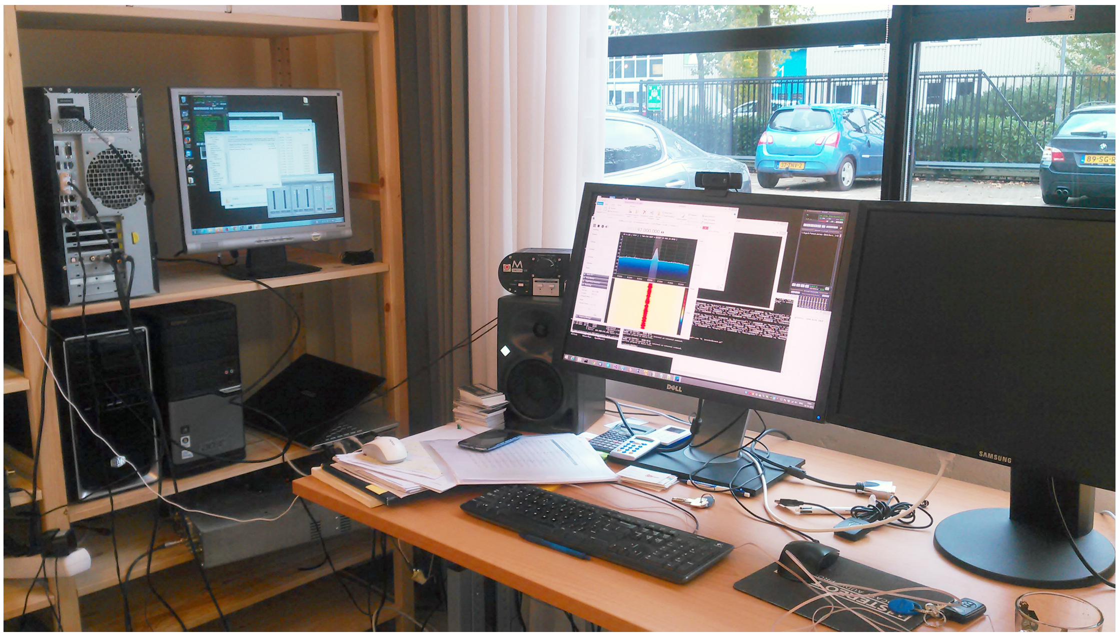


Figure 7: The test generation setup

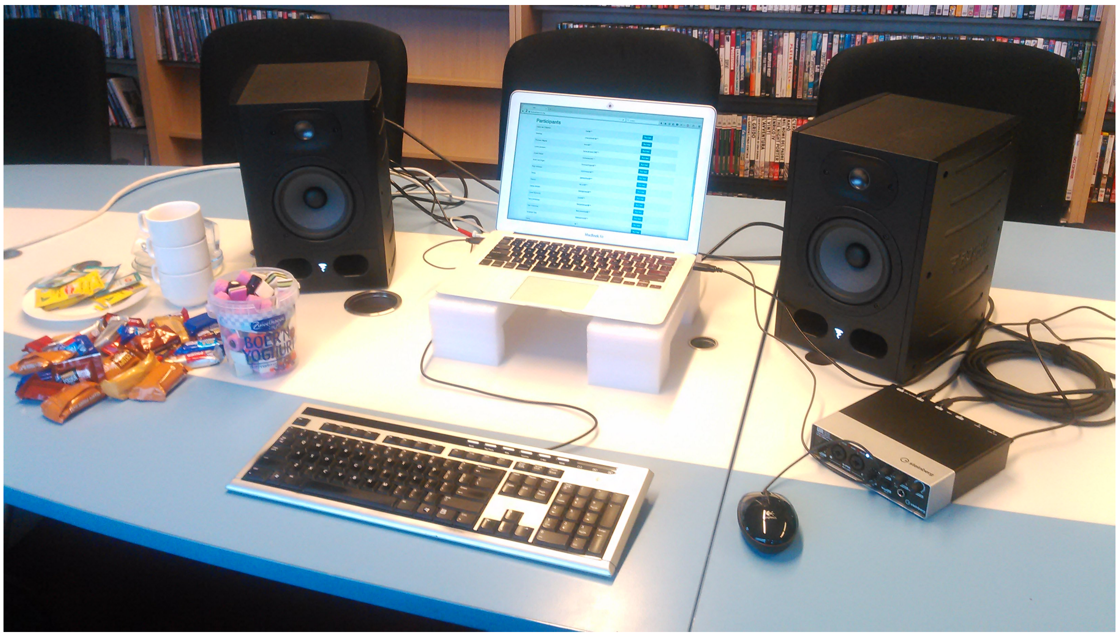


Figure 8: The test setup

The last stage of the decompression involves a simple limiter to ensure the signal complies with the FM standard. The limiter is easily replaced with one of better quality. Since we want the limiter to influence the result as little as possible, the samples were normalized to -15LUFS using the ITU-R BS.1770-2 loudness standard. This prevents the user from recognizing a sample X as being A or B based on just a difference in perceived loudness.

4.2.4 Test subjects

To obtain proper test results a random set of test subjects should be used. However given the limited time available for a master's thesis, we did not use an aselect group of subjects. The subjects consisted mostly of Computing Science students and employees of a public local radio station (N1). Further research should be conducted with a more varied set of subjects. In total 18 subjects took part in the test.

4.2.5 Equipment

The tests were performed in an office space with a carpeted floor and a soft ceiling, providing reasonable acoustics. We equipped the room with a pair of studio monitor speakers (Focal Alpha 50), connected to a laptop through a USB soundcard (Steinberg UR22). The speakers were aimed towards the subject, at a distance of 1 m. A picture of the test setup can be found in figure 8.

4.2.6 Subject preparation

We instructed the subject on the test. First we explained what the test was about: comparing compressed audio samples with potentially lower audio quality and the original signal. We then presented the subject with a sample that was compressed using a very low bitrate (192 kb/s), as to make the artifacts introducted by the codec clear. This prevents the subject from learning while performing the test (which is necessary to obtain independent test results), and provides more useful results as the subject knows what to listen for (Clark, 1982). We told the subjects sample A was unprocessed and that sample B was processed, but that X could be either of the two, randomly.

The subjects were told they had a time limit of 50 minutes, but that it would be no problem if they didn't finish the test completely.

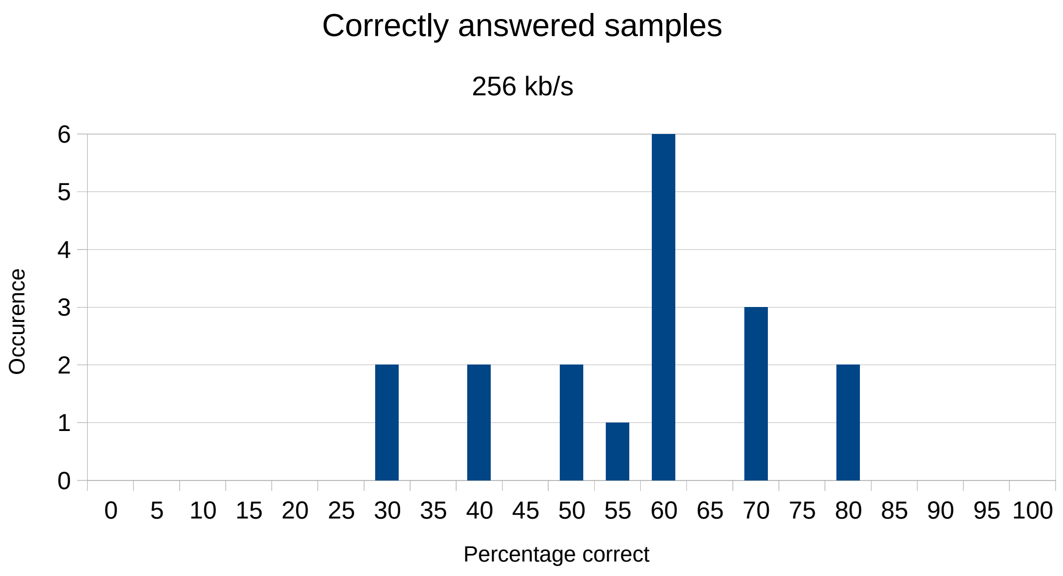


Figure 9: Distribution of percentage answers correct at 256 kb/s

4.2.7 The test

The test was performed using a web-based A/B/X tool[2]. Subjects could operate the tool using either a mouse or a keyboard (a printed list of shortcuts was made available to the subject). The tool allowed for seamless switching between A, B and X. Controls for seeking inside the sample were available as well (allowing the subject to rewind the sample as often as desired).

The subject was able to change the volume at all times using a physical knob. The subject was instructed on what volume setting to use: not too high because it makes it harder to perceive subtle differences, and not too low for the same reason.

Samples were randomised in order and in bit rate. Each subject was confronted with 10 samples at 256 kb/s and 10 samples at 320 kb/s. The order of the samples was randomised, so that bit rates were mixed and so that no two subjects had the same order of test material. In total 173 samples were tested at 256 kb/s, 172 samples were tested at 320 kb/s.

At the end of the test we asked each subject to fill in a small questionnaire. Subjects were asked how confident they were about their answers, indicated on a 1–10 scale (1 being the least confident and 10 being the most). We also asked participants to answer how they would describe the difference (if perceived at all) and if one of the versions sounded better or more annoying than the other one.

4.2.8 Test results

The scores achieved by the participants and the results of their filled in questionnaire can be found in table 5. The score distribution of the two bit rates can be found in figure 9 and 10, respectively.

4.2.9 Hypothesis evaluation

Earlier we stated our hypothesis in section 4.2.2. We will now verify this hypothesis for each of the two bit rates. We assume all tests performed to be independent.

Null hypothesis. *25% of the test subjects can perceive the difference between A and B.*

From the null hypothesis follows:

[2]Available from `http://abx.digitalfeed.net`

N_t	N_c	$\%_{all}$	$\%_{256}$	$\%_{320}$	C	Remarks
18	10	56%	56%	56%	2	Niet storend, verschillen echt alleen na goed zoeken en dan "zou dit het wel eens kunnen zijn", af en toe iets 'blikkeriger'?
20	12	60%	60%	60%	7	De een klonk 'warmer'. (A)
20	12	60%	60%	60%	3	Niet noemenswaardig.
20	11	55%	70%	40%	4	Zo nu en dan is B iets minder zuiver, zeker bij verschillen tussen hoog en laag (na elkaar) zijn zo nu en dan te horen. Ook bij stemmen zijn wel wat verschilletjes te horen. Echter is dit wel erg lastig.
20	9	45%	30%	60%	9	Soms klonk de ene versie (A) wat voller dan de andere. Soms waren achtergrondinstrumenten duidelijker te horen.
16	12	75%	78%	71%	3 / 10*	M.n. "S" klanken (zelfde effect als nieuws). Soms ook stemmen (zelfde effect als Tom's Diner start). Iets minder hoog in B bij Toto?
20	12	60%	60%	60%	3	Af en toe klein verschil, maar dan was 1 fragment niet overduidelijk beter of slechter.
20	14	70%	80%	60%	4	Nee, ik kon geen verschil horen (gegokt)
20	8	40%	50%	30%	10	Geen verschil gemerkt
11	7	64%	60%	67%	4	Muziek A klonk wat voller / helderder soms
20	13	65%	60%	70%	8	Heldere tonen (clean gitaar, gerinkel, etc) minder helder. Niet fijner / minder fijn, nauwelijks verschil. Alleen merkbaar met heel vaak opnieuw luisteren.
20	13	65%	70%	60%	8	Bij geen enkele track had ik het gevoel dat het vervelend in het gehoor lag. Ik moest mijn best doen om de verschillen goed te horen.
20	11	55%	70%	40%	3	Op sommige punten dacht ik in de gecomprimeerde versie iets meer ruis / lichte kraakjes te horen. Qua luisterplezier was er voor mij geen verschil.
20	7	35%	40%	30%	3	Amper te horen.
20	8	40%	30%	50%	6	Ja, soms is het niet verklaarbaar, maar voelt een van de versies (A) prettiger aan.
20	11	55%	60%	50%	3	Verschil minimaal voor mijn gehoor elke versie prima
20	6	30%	40%	20%	3	Haast geen verschil, bij twee nummers dacht ik het te horen (duidelijk). In de lage tonen anders.
20	10	50%	50%	50%	-	-
345	**186**	**Avg 53.9%**	**Avg 56.6%**	**Avg 51.2%**	**Avg 5.0**	*(averages weighted by N_t, N_{256} and N_{320}, respectively.)*
					Med 4.0	

Table 5: Responses of test subjects. N_t is total number of samples tested, N_c number of samples correct, $\%_{all}$ is percentage correct, $\%_{256}$ is percentage correct at 256 kb/s and $\%_{320}$ is percentage correct at 320 kb/s. C is confidence subject indicated on their answers.
**Answered inconsistently, not taken into account for averaging.*

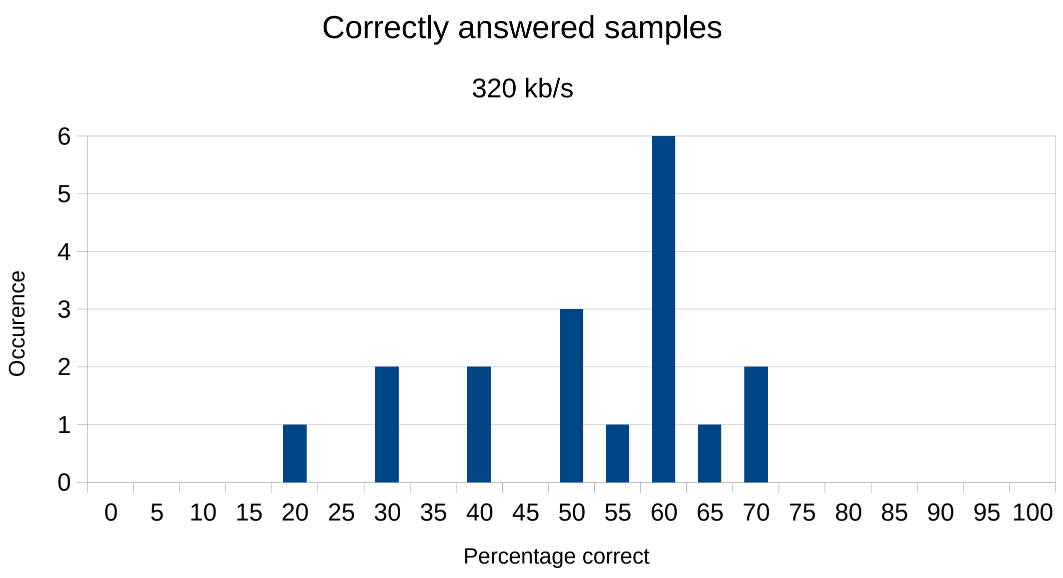


Figure 10: Distribution of percentage answers correct at 320 kb/s

	Expected correctness	**Total correctness**
75% subjects are guessing	50% correct	$0.75 \cdot 0.5 = 37.5\%$ correct
25% subjects perceive difference	100% correct	$0.25 \cdot 1.0 = 25.0\%$ correct
100% subjects		$p = 62.5\%$ correct

256 kb/s. *173 tests were taken, 98 samples were answered correctly.*

We approximate the binomial distribution with $\mu = p \cdot N = 0.625 \cdot 173 = 108.125$ and $\sigma = \sqrt{p(1-p) \cdot N} = \sqrt{0.625 * 0.375 * 173} \approx 6.37$ by a normal distribution $N(\mu, \sigma)$. We can reject the null hypothesis with confidence 95% if at most $k = \Phi^{-1}(0.05) \cdot \sigma + \mu = -1.64 \cdot 6.37 + 108.125 = 97.68$ samples are correct.

98 samples were answered correctly. This means we cannot conclude with confidence 95% that less than 25% of the test subjects perceives the difference between A and B, because $98 \not\leq 97.68$.

320 kb/s. *172 tests were taken, 88 samples were answered correctly.*

We approximate the binomial distribution with $\mu = p \cdot N = 0.625 \cdot 172 = 107.5$ and $\sigma = \sqrt{p(1-p) \cdot N} = \sqrt{0.625 * 0.375 * 172} \approx 6.35$ by a normal distribution $N(\mu, \sigma)$. We can reject the null hypothesis with confidence 95% if at most $k = \Phi^{-1}(0.05) \cdot \sigma + \mu = -1.64 \cdot 6.35 + 107.5 = 97,09$ samples are correct.

88 samples were answered correctly. This means we can reject the null hypothesis with significance $\alpha = 5\%$, since $88 \leq 97.09$, and we accept the main hypothesis, namely less than 25% of test subjects can perceive a difference between A and B.

4.3 Conclusion & outlook

In this thesis we have proposed a new compression method, composed of various techniques found in literature. Our codec is unique in the sense that it is the first known (or at least the first written about) that is aware of the structure of FM MPX, thereby able to achieve a relatively high compression ratio compared to lossless coding, while abiding by the FM specification and while making sure the effect on the audio quality is not perceivable.

Some subjects indicated that they were able to spot minor differences between sample A and sample B, but then had a hard time indicating which of the two was equal to X. Therefore it may be that in some cases a difference *can* be perceived, but will not likely be when a reference sample is absent.

Looking at the results, a bit rate of 320 kb/s provides such sound quality that no, or only a small difference is perceived while the bit rate is still acceptably low. For example Dutch radio station 538 uses a satellite connection of 384 kb/s, on which only audio is transported, not full MPX. They are an example of a station that could use our codec to transport their signal, eliminating the need for extra sound processors at each transmitter site.

The codec created for this thesis is still very much a prototype. No streaming functionality was implemented, only files can be processed. If a streaming option was implemented directly, the delay between input and output would be very large (around 5 seconds). This is partly due to the use of offline libraries for e.g. resampling and filtering. It is also partly due to buffers present in the code, which are there to make things work, but could possibly be removed if the whole codebase was re-designed from scratch.

There is more that should be improved on this codec. For example, while the bit rate is relatively stable, sometimes short spikes occur. We experienced this problem, but as the codec was used in an offline way this posed no real problem. This is undesirable when streaming however, because it could cause short hiccups in the audio as it takes too long to submit one data packet. A mechanism should be put in place to protect against this.

Furthermore, a number of constants is used in the code (mentioned earlier in the description of the algorithm). For these constants, we picked values determined empirically, by making a rough guess of a good value followed by iteratively testing and updating of the constant. This yielded good results, but it is possible the constants are still far from optimal (they were never updated and tested in combination, for instance).

At the moment no mechanism is present to take stereo correlation into account. When coding a stereo signal, actually two full mono streams are stored. This could be made more efficient, because in practice the correlation between the left and right channel is often high. The bit rate could be lowered or the audio quality could be improved by making use of this.

Some technicians do not want to put pc's at their transmitter sites because pc's "feel" less stable than hardware-based processors. Because our implementation is fully software-based, a pc of some kind (everything between a server and a raspberry pi) is needed at the transmitter site. It may be that these technicians will also distrust the use of a computer used for this codec.

Currently, we have only considered FM MPX with stereo audio and RDS. However, it is possible to store more than that in MPX. For example RDS2, a standard currently under consideration, uses more than one subcarrier. Our codec is not directly suitable for this. For every additional part of the multiplexed signal, a way to compress and store it has to be designed in order to guarantuee good quality and an acceptable bit rate.

Also, our codec currently assumes the mirrored L–R part is perfectly symmetrical tot the original L–R. While this does indeed hold for the output of some sound processors (e.g. Stereo Tool), it is not always the case. To be able to reconstruct the original MPX signal, this possible asymmetry should be coped with as well. This should not have a big impact on bit rate, as the asymmetry is probably only minor and thus needs only a small amount of data to store (an exception to this is single side band (SSB), were the mirrored L–R part is absent, leaving a lot of asymmetry – but this could be stored efficiently by using one bit indicating the use of SSB).

So, a lot is to be improved before this codec can finally be put into practice. However when this is done, the idea of having such a codec might appeal to a lot of radio stations who only have access to slow connections and are on a tight budget. Eliminating the need of separate sound processors for each transmitter site, the ability to stream full MPX directly (even over slow connections) might be very useful.

References

Benvenuto, N., Bertocci, G., Daumer, W. R., & Sparrell, D. K. (1986). Report: The 32-kb/s ADPCM coding standard. *AT&T technical journal*, *65*(5), 12–22.

Brandenburg, K. (1999). MP3 and AAC explained. In *Audio engineering society conference: 17th international conference: High-quality audio coding.*

Clark, D. (1982). High-resolution subjective testing using a double-blind comparator. *Journal of the Audio Engineering Society*, *30*(5), 330–338.

Coalson, J. (2014). The FLAC format. *Xiph foundation.* Retrieved March 2015, from `https://xiph.org/flac/documentation_format_overview.html`

Craven, P. G., & Gerzon, M. A. (1996). Lossless coding for audio discs. *Journal of the Audio Engineering Society*, *44*(9), 706–720.

Durbin, J. (1960). The fitting of time-series models. *Revue de l'Institut International de Statistique*, 233–244.

European Committee for Electrotechnical Standardization. (1998). *Specification of the radio data system (RDS) for VHF/FM sound broadcasting in the frequency range from 87,5 to 108,0 MHz.*

Hans, M., & Schafer, R. W. (2001). Lossless compression of digital audio. *Signal Processing Magazine, IEEE*, *18*(4), 21–32.

Hsieh, G.-C., & Hung, J. C. (1996). Phase-locked loop techniques. a survey. *Industrial Electronics, IEEE Transactions on*, *43*(6), 609–615.

Huffman, D. A., et al. (1952). A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, *40*(9), 1098–1101.

Keiler, F., Arfib, D., & Zölzer, U. (2000). Efficient linear prediction for digital audio effects..

Kleinsteuber, H. J. (2006). A great future? Digital radio in Europe. *Recherches en Communication*, *26*(26), 135–144.

Lax, S., Ala-Fossi, M., Jauert, P., & Shaw, H. (2008). DAB, the future of radio? the development of digital radio in four European countries. *Media, Culture & Society*, *30*(2), 151–166.

Levinson, N. (1949). The Wiener RMS error criterion in filter design and prediction, Appendix B of Wiener, n.(1949). *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*.

Makhoul, J. (1975). Linear prediction: A tutorial review. *Proceedings of the IEEE*, *63*(4), 561–580.

Moffat, A., Neal, R. M., & Witten, I. H. (1998). Arithmetic coding revisited. *ACM Transactions on Information Systems (TOIS)*, *16*(3), 256–294.

Montgomery, C. (2015). *Vorbis I specification.*

Pan, D. (1995). A tutorial on MPEG/audio compression. *IEEE multimedia*, *2*(2), 60–74.

Princen, J., & Johnston, J. D. (1995). Audio coding with signal adaptive filterbanks. In *Acoustics, speech, and signal processing, 1995. icassp-95., 1995 international conference on* (Vol. 5, pp. 3071–3074).

Robinson, T. (1994). *SHORTEN: Simple lossless and near-lossless waveform compression.*

Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, *37*(1), 10–21.

Sinha, D., & Johnston, J. (1996). Audio compression at low bit rates using a signal adaptive switched filterbank. In *Proceedings of the acoustics, speech, and signal processing, 1996. on conference proceedings., 1996 ieee international conference-volume 02* (pp. 1053–1056).

Sinha, D., & Tewfik, A. H. (1993). Low bit rate transparent audio compression using adapted wavelets. *IEEE Transactions on Signal Processing*, *41*(12), 3463–3479.