Mobile Peer-to-Peer Audio Streaming

Andreas Lüthi
Bachelor Thesis
Computer Science Department
ETH Zürich
8092 Zürich, Switzerland
Email: aluethi@student.ethz.ch

Abstract

A peer-to-peer network has several features over a standard network connection that need to be accounted for while developing a scalable audio codec. There is a multitude of possible ways to implement such an audio codec with tradeoffs ranging from partitioning the network to having computationally complex ways of scaling the compressed signal (which is not useful in a mobile setting where energy efficiency is key). The developed codec scales along the available channels, with a base layer providing a mono signal to more than one layer which enable for example stereo signals. This scaling technique allows for an elegant implementation in the existing Vorbis I codec and provides the necessary properties that are important in a peer-to-peer environment. The codec aims to provide multimedia content (like movie audio or pieces of music) in a way that allows the efficient use of the available bandwidth on a mobile peer-to-peer client.

1 Introduction

While the mobile revolution has already begun with the dawn of devices like the iPhone or Google Android devices, the evolution of audio codecs was more or less stagnant in this area of use.

Currently available scalable audio codec technologies implement usually one of two ways how an audio signal can be encoded. One such way would be to encode an audio signal into different independent channels and each channel with different qualities (e.g. 32kbit/s, 64kbit/s and 128kbit/s) and switching between the channels if the available bandwidth changes (*stream switching*). The other way would be a technique called "Bitrate Peeling" where the sending side for example reduces the resolution in which a signal is sampled. This reduction of resolution can for example be achieved by lowering the number of bits which are needed to represent the quantification values of a signal.

Stream switching usually works well in server based environments where there are central entities in a network which distribute the content – with regard to a peer-to-peer network there are some differences which need to be accounted for. On one hand there would be a need to partition the peer-network into peers of different bitrate capabilities and on the other hand the switching between streams would be extremely costly because of the new peers that need to be found for the new bitrate. [1] This would imply that there could be troubles with seamless stream switching.

This facts led to the idea of developing a scalable audio codec which uses a hierarchical layering technique much like the one available in todays video codecs. For lower bandwidth connections a base layer would suffice to have a reasonably good audio quality and while bandwidth would increase with better connections like third generation mobile networks (LTE, HSDPA, UMTS, etc.) and/or Wireless LAN (WiFi) more layers could be added to improve the overall quality of the audio signal.

1.1 Organization of This Report

In Section 2 we will first discuss what we are going to encode and how the current Vorbis reference implementation works. Section 3 will embrace possible ways on how the Vorbis codec could be modified to implement a scaling. It will also include some reasoning about what facts led to our decisions. In Section 4 we give a short overview on how the prototype implementation has been developed. Section 5 discusses and analyzes our findings while developing a scalable Vorbis codec. The report concludes with Section 6 and a short conclusion which subsumes the report and gives pointers in future directions.

2 Theoretical Background

2.1 What will be encoded

In order to encode an audio signal it has to go through the process of digitalization. This process consists of a sampling and a quantization phase.

To represent a continuous analogue signal in a finite system we have to select and record the ordinate values of the continuous signal in a time dependent manner. Usually we take so called samples of the signal within equally spaced distances with respect to the time axis; i.e. with a signal that is to be sampled with $44.1kHz$ we take the ordinate value of the signal every $\frac{1}{44100}th$ fraction of a second. The resulting discrete signal is often referenced to as a time series of a signal. [2]

The signal, now discrete in time rather than continuous, is nevertheless still an analogue signal because the pulse amplitudes (ordinate values) can still take on every possible value. Therefore we quantize the signal. If such a signal is quantized each pulse is adjusted in amplitude to coincide with the nearest value of a predefined finite set of possible values. This signal is no longer analog but digital in nature and as a consequence can be represented by a finite set of symbols. [2]

The resulting digital format is denoted as a PCM (Pulse Code Modulation) representation of the audio signal. This PCM representation is often regarded as 'raw' or the original reference in digital multimedia signal processing applications. [3] (see figure 1 for an illustration of the process).

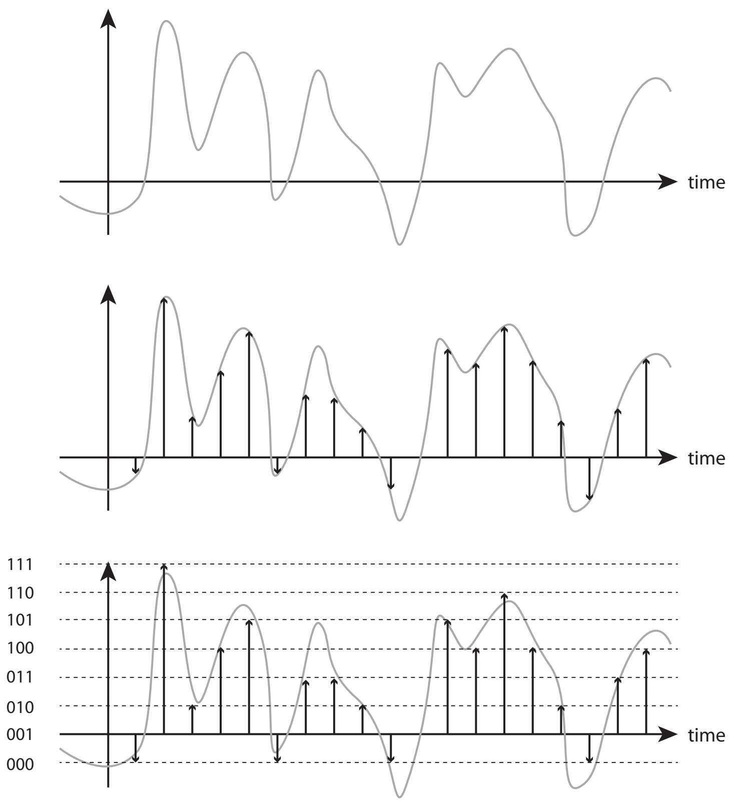


Fig. 1: An illustration of the sampling and quantization process. From top to bottom we see first the analog input signal then the sampled time series (arrows). The sampled time series is the input signal measured at a regular interval. The last part shows the quantization. The quantization represents an approximation of the analog input signal into a finite discrete representation.

2.2 How the existing encoder works

To explain how the Ogg Vorbis codec internally works, we go step by step through the encoding pipeline as it is depicted in figure 2.

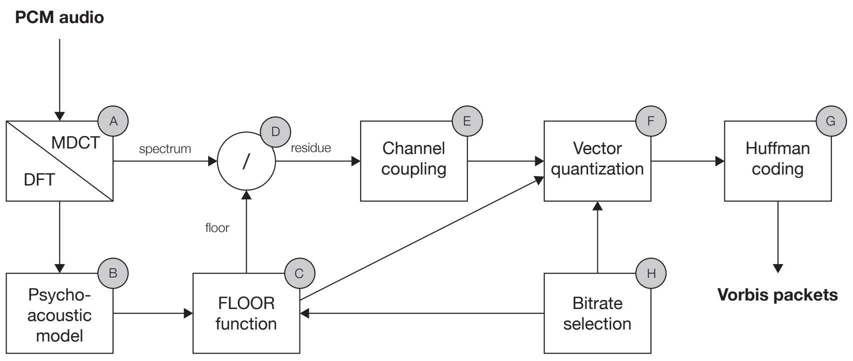


Fig. 2: Encoder pipeline (Source: [4]). The enumeration of the stages is only for referencing purposes and does not reflect the flow of the signal through the pipeline.

As denoted in step A in figure 2 the first task is to take the signal and transform the representation into a frequency domain representation. The reference implementation of the *Vorbis I* encoder does this in two ways. On one side the input signal gets windowed and transformed by an *MDCT* (Modified Discrete Cosine Transform) and on the other side the input signal gets windowed and transformed by a *DFT* (Discrete Fourier Transform). Because it is impossible to encode arbitrarily large signals (for example a radio stream which runs non-stop over several years cannot be transformed at once because of size and time restrictions on a computer) an input signal gets windowed. Windowing is a process which takes a part of a signal, e.g. a 2048 or 512 samples chunk (in Vorbis I only those two window lengths are used – the shorter one even only for critical music signals with sudden changes), applies the requested transformation(s) and returns the encoded signal.

The MDCT transformation works, like the name suggests, in principle like the widely used *DCT* (Discrete Cosine Transform) with some minor changes. To easily understand the differences between the MDCT and the DCT we can look at some of the widely used picture compression algorithms [2]. With those algorithms we take usually a block (window) of a picture and encode it separately. This leads to the often seen blocking artifacts, where we see the edges of a block due to different local attributes within a block. This essentially happens also with audio signals. To reduce this effect we often use the MDCT which provides an overlapping of the windows as it can be seen in figure 3. This overlapping leads to smoother edges and therefore a signal with lesser fast changes (like crackling noises).

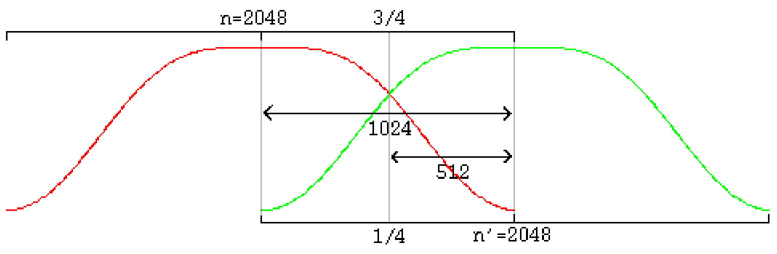


Fig. 3: The visual representation of the overlapping of two MDCT frames. The red and green parts are signal chunks (windows) which overlap to ensure a smooth reconstruction. (Source: [5])

The use of the DFT in the Vorbis I codec is primarily as an instrument to analyze the incoming signal (besides the MDCT). The DFT has the advantages of a better tonal estimation and easier handling and is therefore used as an additional input for the psycho acoustic model besides the MDCT output.

After having the signal transformed from the time domain into the frequency domain the first step of the lossy processing of the signal happens (step B in figure 2). The psycho acoustic model helps to exploit important effects regarding the human hearing system with the use of the masking property of the human hearing [3]. As it can be seen in figure 4 not every frequency can be heard equally well therefore we are able to remove unheard parts from the spectrum to compress the signal in a first step.

The output of the psycho acoustic model is afterwards used to create a low level approximation of the frequency representation of the input signal, the floor (depicted as C in figure 2) shown in figure 5. This floor is afterwards used to remove a large part of the spectrum with the calculated approximation (stage D in the encoding pipeline depicted in figure 2). The remaining part, called residuals (the fine structure of the audio spectrum), are afterwards encoded separately. The floor data itself is independently vector quantized and entropy encoded for packing.

The residuals are in a separate step channel coupled. Channel coupling exploits the high correlation between the encoded channels to further reduce the amount of data that needs to be transmitted. The principle behind channel coupling is to use one channel as a base and encode only the differences to the other channels. After the coupling the residual data is also vector quantized and entropy encoded.

The last step before the final entropy encoding is the vector quantization. Vector quantization is a lossy data compression technique which uses the

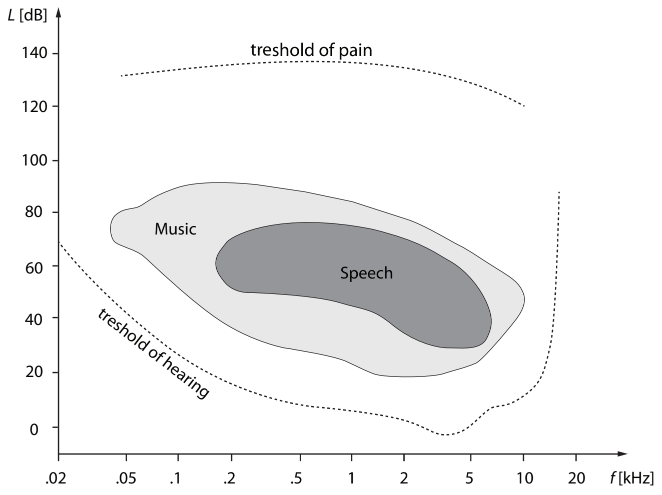


Fig. 4: Parameters of the human hearing depending on sound pressure level. The dark grey region represents the frequencies and sound pressures usually occurring with natural speak. The light grey region represents music. This region map can be used as a base for a psychoacoustic model where frequencies outside of the relevant frequency band are discarded from the recording. (Source: [3])

block code principles. This quantization technique works as follows: we take n input values (for an n-sized block) and place them in the n dimensional vector space. We do this for every n-sized block of our input. Now we can approximate those points to their nearest neighbor and let them be represented by it. This partitions our vector space in so called *encoding regions* which are represented by a single *code vector*. If we do this rounding according to the distribution of the points in the vector space we lose a bit of information but still keep the overall picture. The more approximation points we have the better is the approximation of the input values. An example representation of such a partitioned vector space can be seen in figure 6 as a 2-dimensional vector quantization map. Such a map is also often called *Voronoi diagram*.

As the last step in our pipeline a standard entropy encoding with prefix free codes is used to further reduce the size of the input signal. This entropy coding is based on the idea of *Huffman-Coding* which is a technique that encodes the symbols of an input according to their relative appearance in the whole input sequence. The value that appears the most will be coded with the shortest symbol and the value that appears the least in the whole sequence will be assigned the longest symbol. This can be shown to be an optimal code which approximates the entropy of any given input best. [6].

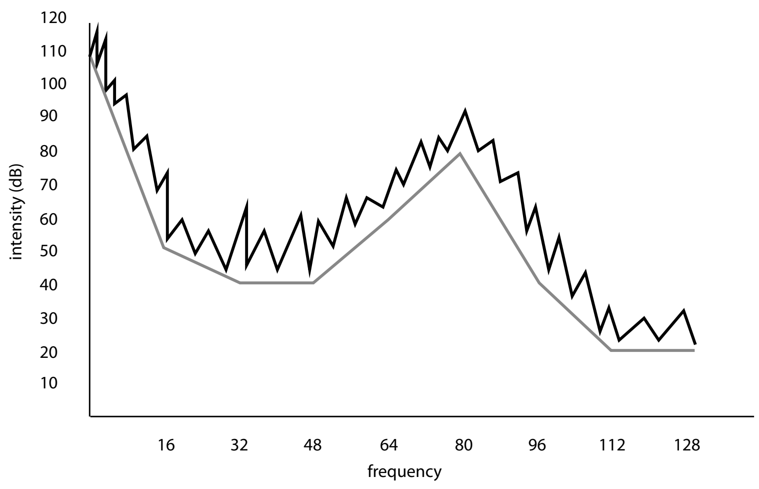


Fig. 5: A visualization of the floor and the residuals. The light grey line is the interpolated floor while the black line is the exact signal. The subtraction of the floor from the exact signal results in the residuals which resemble white noise.

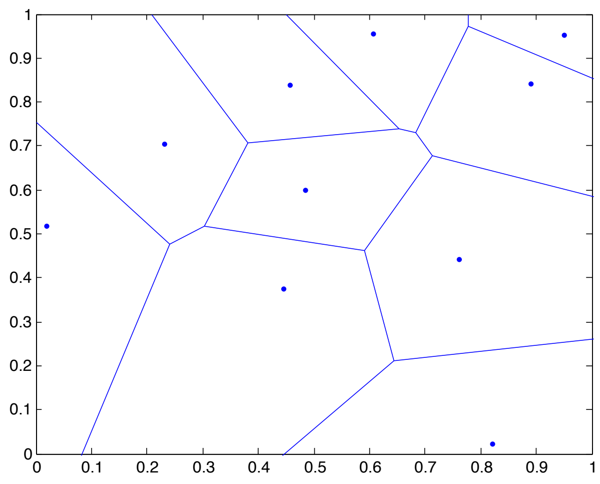


Fig. 6: A Voronoi diagram which illustrates vector quantization. Each two dimensional vector gets rounded to its nearest centroid.

The result after all these steps is an encoded Vorbis packet. This packets can afterwards being stored into an *Ogg container* which would make our signal ready for distribution.

3 Different Possible Implementation Approaches

First we have to introduce the notion of *layer splitting* and what we mean by it. Layer splitting is a technique which is currently often seen with videos in a web setting. An input stream (like an afore mentioned video) is split into several streams. The first stream is called the *base layer*. The base layer accomodates all the necessary information to setup a decoding mechanism for a given stream and carries data for a low quality representation of a signal. Besides the base layer there can be several other layers called the *enhancement layers*. The general idea behind those enhancement layers is to transport additional data to the stream consumer so that the low quality representation can be enhanced. This enhancement data is often just a calculated delta between the data we want to show and the low quality representation of the base layer.

There are several possibilities to scale such a multimedia signal like a scaling of the used bitrate which essentially is a quality scaling, then there is the possibility to scale along the input channels (for example a stereo audio signal can be scaled down to a mono signal). Another scaling would be a sample rate scaling, where the sampling resolution can be adjusted – and there are many more possibilities. We primarily look at bitrate scaling or channel scaling approaches.

To implement such a splitting we identified the following three approaches:

3.1 Provide Stream With Different Bandwidths (stream switching, see [1])

An easy approach would be to encode a stream in several different streams with different quality attributes. This would enable a mobile device to switch between the different streams according to its available bandwidth. The advantage of this approach would be the low complexity of implementation.

Large disadvantages would be the fact that a server would have to provide several different streams with preset different audio qualities. This would increase the outgoing traffic on the server side and make it hard for the mobile devices to make a seamless handover of the streams because of buffering issues and probably different routing paths. In a peer-to-peer setting are even more disadvantages identifiable. If we assume, that not every peer provides all streams (which makes sense due to different bandwidth constraints) we would partition our peer network according to the provided streams by each peer. This has an implication on the switching between the bitrates which would be prohibitively expensive. Peers would have to find new neighbors with each bitrate change and could not necessarily reuse the connections they already have. [1]

3.2 Quantization Approach

The first non-trivial idea was to change the quantization of the floor and in a second step of the residuals. The quantization is changed in a way, that the least significant bits of the values of a quantization vector would be removed. This would result in a coarser grained quantization. The assumption here is, that the coarser quantization values would represent all the values with a finer resolution – therefore all those values can be replaced by one value. The signal would degrade but in a controllable way. This idea would have the following advantages: It could in theory be fairly easily accomplished. We would just reduce the count of the partitions which partition the vector space and therefore reduce the amount of vectors used to represent input values (example for 2-dimensional space in figure 6).

A large disadvantage would be, that we wouldn't consider the position of the quantization vectors in the vector space (the values where all other values are rounded to). It can and will happen, that all the newly created quantization vectors are no longer in an optimal position in the vector space – this would reduce the overall optimality of this approach. Also the modification of the codebooks would not be easily made.

Regarding a peer-to-peer scalable solution this would also imply some difficulties – we would need a protocol that lets a sending site scale the quantization otherwise we would again need to partition the network into peers with different provided qualities.

3.3 Bitrate Peeling Approach

The bitrate peeling approach would be to take a given Vorbis input stream and reduce the bitrate of the stream by just stripping bits out of the stream. This is accomplished by removing bits of, for example, the residual part of a file. If a residual is represented by a 16-bit short value, we would just remove bit by bit from the least significant bit side. This technique would result in a) a smaller representation of the stream and b) a smaller codebook section which would be needed to decode the stream (this results from the idea that due to the bit peeling less symbols are used to represent the stream and therefore a smaller codebook can be used). The bitrate peeling approach is very closely related to the quantization approach. But instead of creating a coarser quantization during encoding we strip the bits out of the final file. The advantages of this approach would be, that we can do this on any Vorbis file without specially encoding it.

In addition to the same disadvantages we would encounter as with the quantization approach we could not take any influence on the maximum bitrate. The maximum bitrate is given by the initial encoding of the file. Other additional disadvantages would be, that we would need to change the codebook entries according to the new values with the bits stripped.

3.4 Floor and Residual Splitting Approach

The fourth approach which has been identified includes serious modifications on the original Vorbis codec. The idea behind this approach is to first preprocess the signal by combining for example combine both channels of a stereo input to a mono channel and a left or right channel (for writing clarity we take the right). The mono channel is calculated by adding the original left and right channel and dividing it by two (so that we get an averaged signal). Now that we have only one channel, we can use the remaining free channel of the Vorbis encoder to encode the right channel. With this two channels – the mono and the right – we are able to calculate the original stereo signal back by multiplying the mono channel by two and subtracting the right channel (this gives us the left channel back). With this approach we exploit the channel coupling of the Vorbis encoder (because the channels are still highly correlated). Now the layering can no longer happen via bitrate peeling or a different quantization because the signal would further degrade. The idea now is to split the signal into a mono layer (which transports the setup data like the codebook and packet information) and a stereo enhancement layer. This splitting would enable the playback of the mono layer by a normal Vorbis implementation and a specially designed decoder could also take the stereo enhancement layer to further improve and calculate a stereo signal.

With this approach we would also partition our peer-to-peer network into peers with different provided layers but one peer would receive its signal from multiple other peers. A client peer could have a peer A which provides it with the base layer and a peer B and/or C which provide enhancement layers. This complicates the peer handling a bit because we have multiple connections to manage (we need to find peers for every layer) but we could switch seamlessly between different quality levels and wouldn't need to begin a bootstrapping by switching to another quality level.

4 Implementation

While looking at the different approaches on how to implement such a layering of an audio signal we came to the following conclusion. The first approach (stream switching) would not consider our requirement on keeping outgoing traffic on a minimum. The quantization approach would not be easily feasible because the mentioned disadvantage that the created quantization vectors would be no longer in an optimal position and we would have to somehow transfer a large codebook or create different streams like we have to do with the stream switching approach. The bitrate peeling approach has not yet been implemented but it is a topic brought up time and time again in the Ogg Vorbis community. There is currently a bounty set to implement bitrate peeling in Vorbis.

The problem all of the previous mentioned approaches suffer is the partitioning of the peer network in a peer-to-peer case.

So we decided to go with the fourth approach to develop a layered Vorbis codec and split the floor and residues according to the two available channels.

We used the Xiph.org reference implementation of the Vorbis codec as a basis for our implementation. This way we didn't need to implement things like the MDCT or DFT.

4.1 Implementation Overview

In order to accomplish the layering we had to adjust the codec on several levels. At first the input signal needed to be converted into a format, which lets us easily restore it and provide an audio source for a client in a transparent fashion (this means, that the client does not necessarily know if there are one or more layers available). To accomplish this, the audio consumer needs to always receive all channels (in an extreme case, all channels will carry the same signal).

The next step was to make sure that all the necessary codec setup data was available independently of how many layers were received by the client, and that the base signal (floor) was distributed correctly among those different layers. The remaining data (the residuals) needed to be processed and split carefully to make sure that the important part of a Vorbis sound signal gets transferred completely.

Finally the decoder part of the codec needed to be able to interpret the incoming signal and switch between layer modes depending on how many layers it was receiving and provide the afore mentioned transparent signal to a client.

4.2 Input Signal Preparation

The incoming PCM signal was at first adjusted in a way that we would make it possible to a) reconstruct the source signal and b) to calculate a pseudo stereo signal from one channel.

To accomplish this we took the average of the incoming channels by adding them, dividing them by two and interlacing the calculated signal with the signal of either the left or the right stereo channel. This way we are able to reconstruct the input signal with almost no error by just multiplying the calculated layer by two and subtracting the other channel.

If we now only receive one channel (the one with the averaged signal) we can reconstruct two channels from it – the main information we lose is the spatial information. Besides keeping enough information for just one layer this technique has another advantage that it does not compromise the channel correlation and coupling.

After the input signal has been prepared we can encode it by feeding it into the Vorbis codec. To layer the Vorbis output signal we have to modify how Vorbis encodes and decodes the signal. After analyzing how a general audio signal gets encoded we were able to identify the relevant sections where modifications were needed.

4.3 Splitting the Headers and Floors

In a first step we had to change some of the initialization code in `mapping0_forward()` to ensure that there are enough buffers to accomodate for the multiple output layers (the standard codec provides only one output buffer). This includes the creation of multiple `oggpack_buffer` output buffers – one for each layer. The first `oggpack_buffer` (layer 0) includes in addition to the encoded signal also the header structure with information about the layer count, type of packet, packet number, etc.

Afterwards the input channels are analyzed by a MDCT and a DFT. Because the DFT provides a better tonal estimation it is better suited as an input fort he psychoaccoustic model. The psychoaccoustic model ist he part oft he codec, where for example frequencies outside the audible range are deleted. Up to the psychoacoustic model nothing had to be changed – which is very good, because a good psychoacoustic model needs to be thoroughly implemented and tested with real test subjects in a special test setting. This originates from the properties of the human hearing system where some frequencies are better heard than others or are even masked and shadowed.

The next step in `mapping0_forward()` includes the interpolation of the floors for both channels and the packing of them into the output stream. Vorbis encodes each PCM input channel into a low-resolution spectral floor which is used as a whitening filter for the input signals [5]. These floors are afterwards saved in their respective `oggpack_buffer` for each layer. If layering is off, the floors are stored sequentially in one `oggpack_buffer` as it is done with the standard Vorbis codec. This sequential storing helps in a later entropy encoding phase to further compress the floors which is unfortunately not the case with the scalable version.

4.4 Encoding the Residuals

The remaining data after the removal of the floor represents the fine structure of the audio spectrum (the residuals) [5]. These resemble white noise with one main difference: the channels are usually highly correlated to each other. The residue format of the standard Vorbis codec partitions each channel into a bundle of chunks, classifies each chunk encodes the chunks classification and finally encodes the chunk itself with a specified vector quantization arrangement (see figure 7).

The standard Vorbis codec implements three different modes on how this classification works. Mode 0 and 1 encode each channel separate and differ only on how the partitions are encoded (mode 0 interleaves this partitions and mode 1 not). Mode 2, which is usually chosen by the encoder itself to encode a 2 (or more) channel signal, interleaves the channels first (flattens them into one long vector) and encodes the signal afterwards like mode 1.

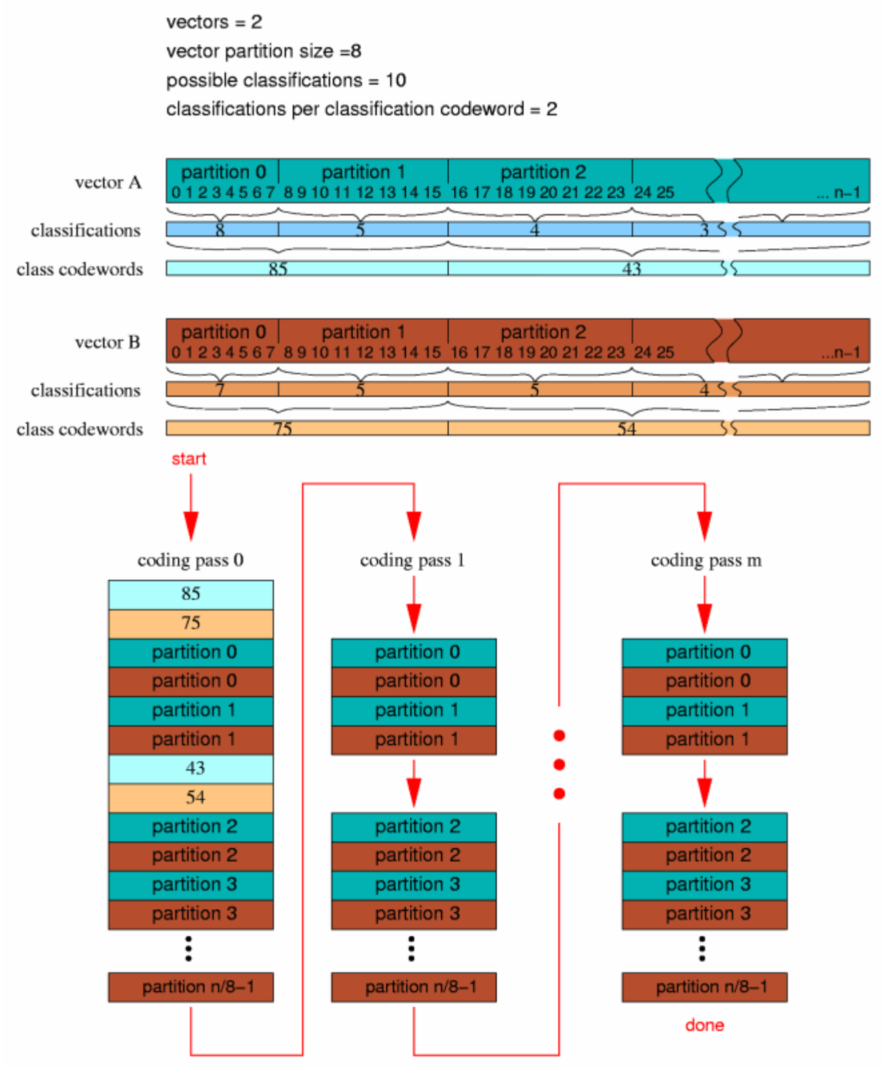


Fig. 7: Illustration of the residue encoding. In a first step a partition is classified and multiple classification values are concatenated to a codeword. Afterwards the classification codewords are put into a buffer followed by the partition data itself and interleaved between the different channels. (Source: [5])

This enables the codec to further exploit the high correlation between channels of the same audio signal with special channel coupling techniques.

With our implementation we had to make sure that this flattening which occurs in mode 2 does not happen because we could not reconstruct the signal with just one channel where not all information would be available (see figure 8). This requirement results from the way we split the channels – we separate the data chunks to have a chunk which originates from channel 0 being saved in our layer 0, a chunk from channel 1 in layer 1 and so on. With the data of each chunk being interlaced we cannot distribute the chunks over the different layers because we would always have information with us that does not belong to the current layer and information that is missing because it is in the next chunk.

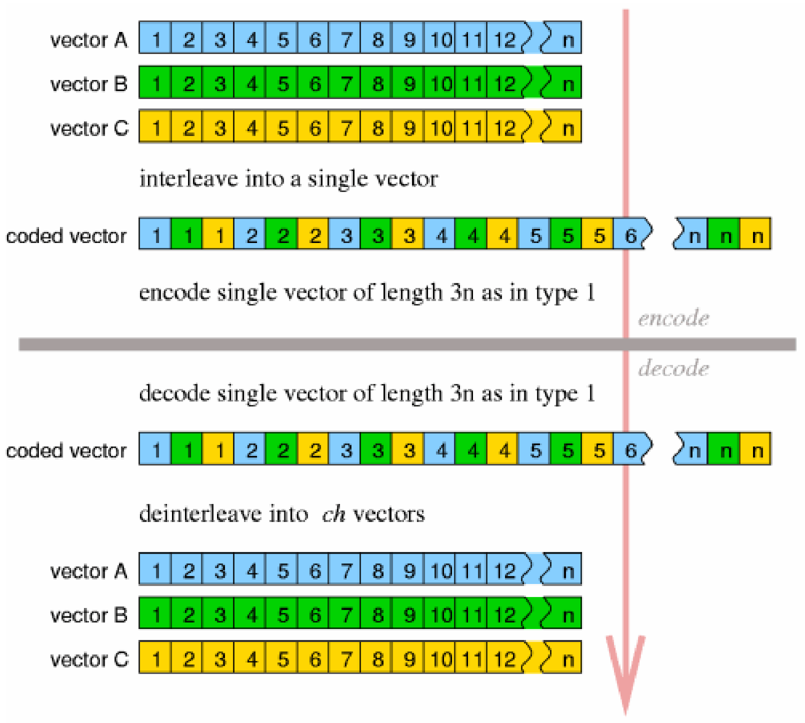

vector A
vector B
vector C
interleave into a single vector
coded vector
encode single vector of length 3n as in type 1
encode
decode
decode single vector of length 3n as in type 1
coded vector
deinterleave into ch vectors
vector A
vector B
vector C

Fig. 8: Illustration of residue mode 2. This is a variant of the residue mode 1 where the channel vectors are interleaved into one large vector. This vector is afterwards encoded with the same process that can be seen in figure 7. (Source: [5])

In the last part of the `mapping0_forward()` function we see the call to the residual encoding which does the previously described steps. The way it is implemented now is that if the `channel_layering` flag is set, the residual encoding mode is fixed to 1. This avoids the discussed flattening of the input channels and lets us easily separate the input signal into multiple layers. This separation is done in the function `_01forwardsc()` in the file `res0.c` of the codec. The main modification is that we call this function with an array of `oggpack_buffer` pointers as an argument, which represent the separate layers and afterwards make a case differentiation on the `channel_layering` flag.

If we are not in channel layering mode we only use the first pointer (only this one is allocated) and safe our output as the standard Vorbis implementation does. If channel layering is active we first encode the partition codewords for each channel and safe them into their respective layer's `oggpack_buffer`. In the next step we encode the residual values for each partition. Standard Vorbis would now interleave the input channels on each partition word to optimize for the later occurring entropy encoding stage. With channel layering enabled we do not interleave but instead split the signal into the different Ogg buffers therefore we need more space because we cannot really use any entropy encoding effects.

4.5 The Decoder

The rest of the encoding process remained the same like in the reference implementation of the Vorbis codec from Xiph.org.

The decoding process is generally speaking the inverse function to the described encoding functions. The main difference is, that for the encoding the primary unit of work is not the `oggpack_buffer` but the `vorbis_block`. After the encoding process we receive an `ogg_packet` from the encoder which is a unit that encloses an `oggpack_buffer` and augments it with additional information like packet sequence number, size, etc (usually values that came with the packet header). The `vorbis_block` structure represents an unpacked `ogg_packet` structure with additional information like frame size (for the MDCT windowing) and additional buffer pointers for the PCM signal which is going to be decoded.

In order to decode the signal correctly we need to know two additional things about the stream than just the stream data itself. First we need to know whether the current input is channel layered input or not and if so how many channels are delivered for decoding (starting with the base layer). To get the first information we had to extend the Vorbis header by an additional bit flag, which represents the channel layering mode. This change unfortunately broke compatibility with the standard Vorbis codec. This change has been done as a convenience for the developer using the codec. In principle this setting could also be preset externally by calling a function that sets the codec into scalable mode – but in order to easily test and implement the prototype this breaking of compatibility has been accepted. The second change is determined by the client software which uses the codec (this depends for example on network parameters like how many peers are available and what layers do they provide). This parameter needs to be given to the decoder as an argument while decoding the receiving signal. Afterwards the signal needs to be fed into the decoder as an array of `vorbis_block`s.

With this information in place the decoder can allocate the needed space to decode the final audio signal and loop through each available layer to decode the signal. This decoding works exactly the inverse way of the description given on the encoding part above.

5 Findings and Analysis

To test our prototype version of the scalable Vorbis codec we used several artificially generated input signals as well as a sample of a movie and an excerpt of a song to get some intuition on how the codec will perform in a real live setting where mainly movies and/or music is streamed.

The artificial signals (figure 9) have been generated with Matlab and try to cover the human hearing range (figure 4) to see the potential effects different frequencies have under the influence of the psychoacoustic model of Vorbis. To see how the compression behaves we also generated a random signal (white noise).

The idea of testing sine waves was to have some kind of reference on how the codec performs and the random signal represents an extreme corner case where the two channels do not correlate to each other.

In order to test the codec, a benchmark has been created where we could compare the different signals under the different modes for the codec. The different modes that have been tested are the following: a) the scaling is turned off (we use the standard compression of Vorbis), b) the scaling is turned on and two layers are received, c) the scaling is on but only one layer is received. We tested the codec under this premises with different sample sizes (from 2^{13} to 2^{22}, which equal 2^{14} to 2^{23} bytes of data because of the two channels) to see if the size of the signal has any influence on how it is compressed (larger codebook size, different codebook use, etc.).

A measure often used in signal processing to test whether a lossy compression codec provides the necessary quality while reducing the size of the data to a reasonable size is the PSNR value (Peak Signal-to-Noise Ratio). While it helps to test for example video compression algorithms it is not very useful with audio compression because of the very special characteristics of the human hearing system. These characteristics include the non-linear distribution of the perceived power over the audible frequency bandwidth as well as effects of different frequencies that can shadow or mask each other while occurring at the same time. Such measures are therefore not used in audio codec development – that is why we also did not use them.

If we start with comparing the artificially generated sine wave signals we see that with our modified Vorbis solution we need about a factor of 2.37 (in figure 9(c)) – 2.72 (in figure 9(f)) times more space for the same signal than with the standard Vorbis compression method. These extreme factors are caused by the fact, that the standard implementation uses residual mode 2 with a signal where each channel has a 1:1 correlation to the other. Another aspect that can be seen here is, that the psychoacoustic model has some optimizations with this residual mode 2 which obviously can be exploited. This is reflected in the fact that the higher compression factors are achieved at the borders of the tested ranges ($100Hz$ and $15kHz$ where much of the signal gets discarded) and the smaller ones near the center of the human hearing spectrum ($1kHz$). A very interesting property of this sine waves is the fact that the standard Vorbis compression works even better than the single layer compression. This is also related to the better use of the psychoacoustic model.

Now that we have established a base with pure sine signals we can test the other end of the spectrum with a completely chaotic signal. White noise is ideal in this case because it is generated by a random process and it has a near flat power spectrum (see figure 10(b)). This means, that the signal contains each frequency in equal power parts within our measured spectrum. In addition to this fact we also have almost no correlation between the two channels. If we compare now the standard compression with the compression of our two layer variant we see that the factor is only about 1.1 times (see figure 10(a)). Regarding the one layer variant we now even have a gain. One layer compressed needs only about 65% of the space of the standard compression mode where scaling is turned off.

After having seen the extreme cases of high and almost no correlation we can now compare signals of a piece of music and a movie excerpt.

If we first consider the music signal (figure 11) we see that the correlation of the channels is relatively high and the use of the residual mode 2, which is used by the standard codec on this kind of input signal, has a high impact on the compression. While the layered modes use the afore mentioned residual mode 1 we can (at least at the moment) no further exploit any correlation between the two channels (even the interleaving of partition words is not available, therefore even entropy encoding can not work as effectively as it can in other modes). The overall increase of the size while compressing the music stream with our codec instead of the standard Vorbis codec settles at about a 25% increase over the output of the standard encoder. If we only use one layer we are still with about 90% of the standard output size which provides two channels.

A rather interesting case is the compression of audio streams from a movie. In figure 12 we clearly see, that this is an ideal use case for this kind of scalable codec. With only about 15% more space required we achieve with our two layer scalable codec the same output quality as with the standard codec. This good result can be explained with the sophisticated mixing of movie sounds. While there usually are different audio sources placed in a virtual room around the audience the correlation between the different channels gets reduced (e.g. a shot of a firearm from the right side mainly affects the right channel). This leads to a minimized effect of the entropy encoding on the overall signal and therefore to a better relative performance of the scalable Vorbis codec against its standard implementation. If there is only one layer available we reduce the space of our transmitted signal to about 77% of the standard encoding. It seems as if this scaling with respect to the channels fits the use-case of movie streaming very well, especially if we consider the possible use of more than only two channels e.g. for 5.1 surround sound (with 6 channels). This use of more channels would even help to reduce the impact of the overhead we carry in our first layer for the set up of the codec (codebooks, etc.).

In general we see, that although we loose the ability to compress such a signal as good as possible because we are no longer able to exploit the high correlation of the input channels in the entropy encoding stage, we are able to do a reasonable compression while enabling a scaling along the different available channels. This result is very good especially if we consider the fact that the primary use of this solution lies on a peer-to-peer network where we should minimize the need to search for new peers (and therefore interrupt the stream) every time bandwidth is not sufficient enough to sustain a high quality stream.

6 Future Work

While implementing a scalable version of the Vorbis codec we found several possibilities on where to further improve the scaling. With the current codec there are not much optimizations feasible on a local scale. There are possible ways of further degrading for example the base layer signal and increase the enhancement layers to get a larger spread between the layer bit rates but this would not change the maximal bitrate.

A better way to further improve the codec would be to cascade multiple instances of the codec. One way could create several codec instances which work synchronously on the same input signal but are configured to create different quality level output. The lowest level output would represent the base layer and the better quality instances of the codec would use the information generated by the base layer instance for their encoding. This could for example be the residuals which are calculated in the base layer, afterwards inverted and fed into the next quality step encoder. The mentioned encoder on the next stage would only need to encode the difference of its quality level to the previously calculated quality level. Our idea is that such a configuration could create more different quality levels and would better exploit the similarities in the different quality signals. A problem with this approach could be the different MDCT window sizes and the alignment of the packets (that the packet numbering is consistent). An advantage would be the good quality across all layers but with a toll that has to be paid for with higher overheads.

Another idea would be, instead of using multiple instances, to use only one encoder but change the quantization of this encoder to calculate many different quality levels. To accomplish this an encoder would encode a low quality residual afterwards decode it encode the same signal again with a higher quality and subtract the low quality residuals. The signal difference can then be used for a higher quality layer. The reconstruction of the signal would therefore use the low quality layer for a low bandwidth output signal and this low quality layer with the higher quality layer (the calculated difference) added on top to decode a high quality signal. An advantage would be, that the best signal would only use about as much space as the equivalent standard Vorbis signal. One problem could be, that the low quality layers would carry a really bad quality signal.

Both of those approaches could suffer from rounding errors. Each subtracted quality step could introduce a small error which would eventually degrade the signal in way that its audible.

A different approach could be if we exploit the high correlation of the partition words. That means we would only encode the difference of the base layer to the enhancement layer. This would reduce the possible values in a partition word and therefore help the entropy encoding stage to better compress the stream.

This approaches seem to be worthwhile to explore in the future especially regarding the current mobile market and the ever increasing demands in transmitting multimedia data over the air. This could on one hand decrease the burden on the networks of mobile providers as well as enable the transmission of multimedia data over poor network connections which is non-satisfying or even non-feasible at the moment. Regarding our peer-to-peer approach such a codec could include mobile devices to help spread a stream by for example just providing enhancement layers (which are significantly smaller than the base layer) with their low-bandwidth connections. Peers could therefore receive the base-layer from a server and the enhancement layers from devices nearby.

7 Conclusion

Concluding this work we see a tremendous potential in developing scalable audio codecs. While the main focus of the research has been on optimizing the compression of the codecs further almost no progress has been made in the development of scalable codecs. Today, if a scalable audio codec is needed people often just use an established video compression codec – even if the just use the audio part. With the use of a scalable audio codec, in particular considering the use-case of a peer-to-peer multimedia network, much improvement could be made. With the ever growing market for mobile solutions and todays information and entertainment need there seems to be a huge potential to lessen the burden on network and content providers with offloading some of the distribution work into network itself – and multimedia codecs like this scalable Vorbis version enable this process in a very efficient way.

8 References

[1] R. Meier, *Toward Structured and Time-Constraint Content Delivery Systems.* 2011.

[2] I. Glover and P. Grant, *Digital Communications.* Prentice Hall, 2009.

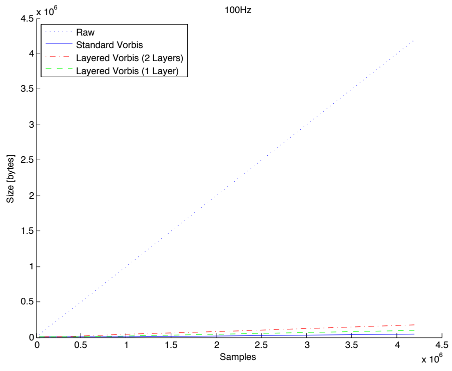
[3] J. Ohm, *Multimedia communication technology: representation, transmission and identification of multimedia signals.* Signals and communication technology, Springer, 2004.

[4] J. Svitek, "Ogg vorbis: Subjective assessment of sound quality at very low bit rates," Technical Report 27, CESNET, 2006.

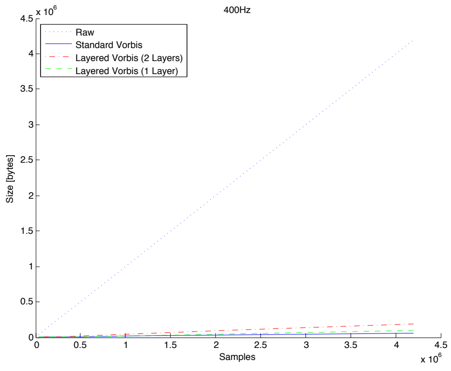
[5] *Vorbis I specification*, August 2011.

[6] T. Cover and J. Thomas, *Elements of information theory.* Wiley Series in Telecommunications and Signal Processing, Wiley-Interscience, 2006.

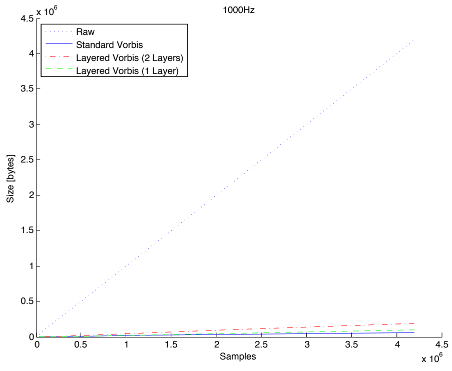
[7] "Vorbis bounty on bitrate peeling." http://wiki.xiph.org/Bounties#Ogg_Vorbis_Bitrate_Peeling, January 2012.



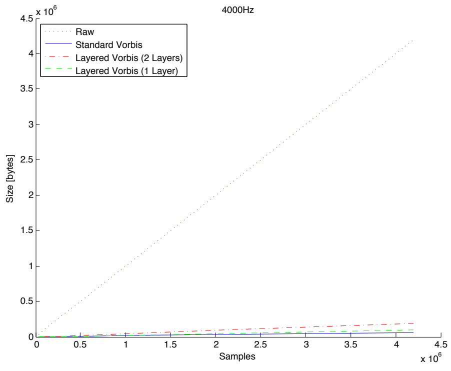
(a) A $100Hz$ sine wave.



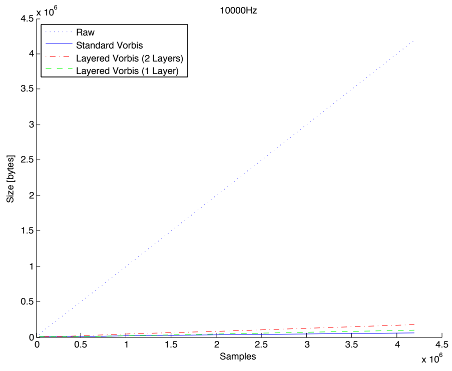
(b) A $400Hz$ sine wave.



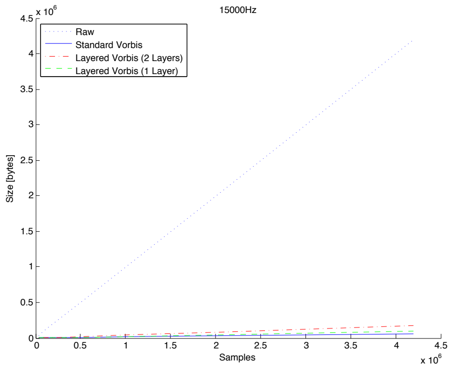
(c) A $1kHz$ sine wave.



(d) A $4kHz$ sine wave.

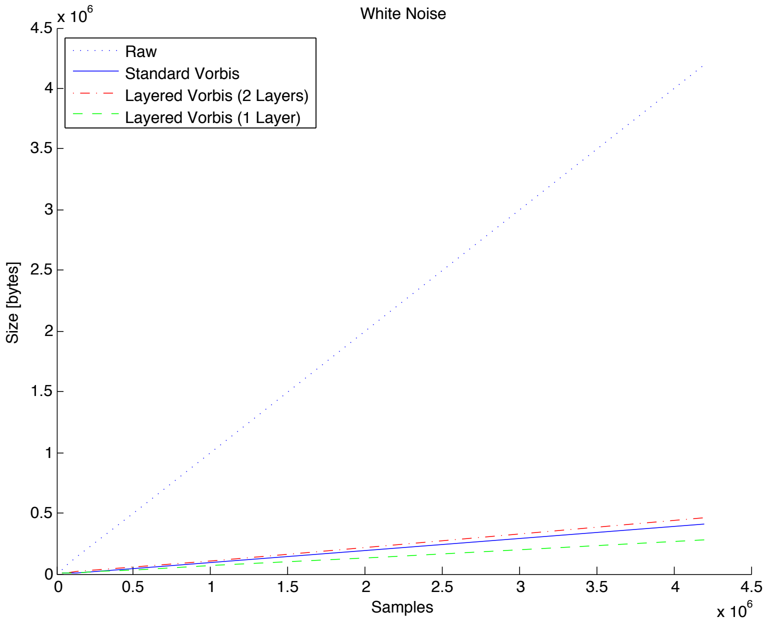


(e) A $10kHz$ sine wave.

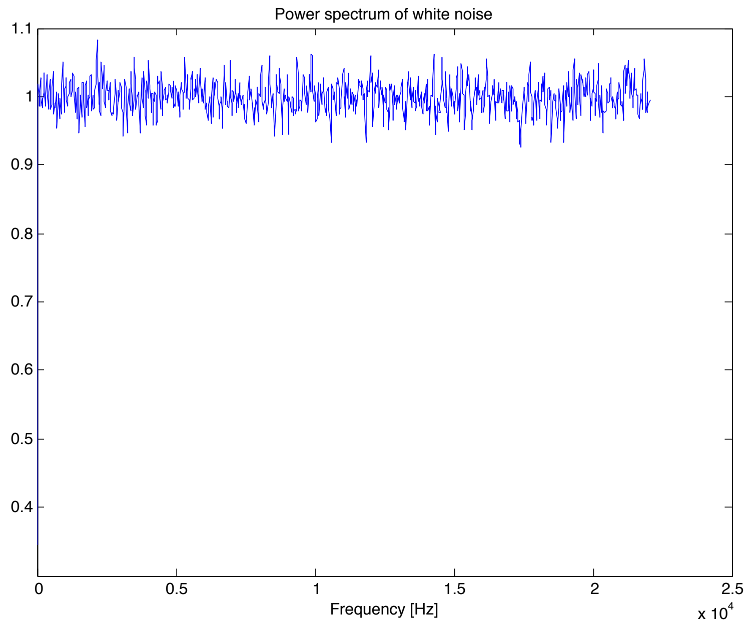


(f) A $15kHz$ sine wave.

Fig. 9: Illustration of the data sizes achieved with different version of the Vorbis codec. We compare the standard Vorbis I codec (blue solid line) with the two channel layered version (red dashed line) and the one channel layered version (green dashed line). The frequencies used are samples across the human audible spectrum.



(a) Size properties of the white noise signal.



(b) The power spectrum of white noise. This type of signal has equal power in any frequency band of a given bandwidth. The sampling frequency used here was $44.1kHz$.

Fig. 10: White noise

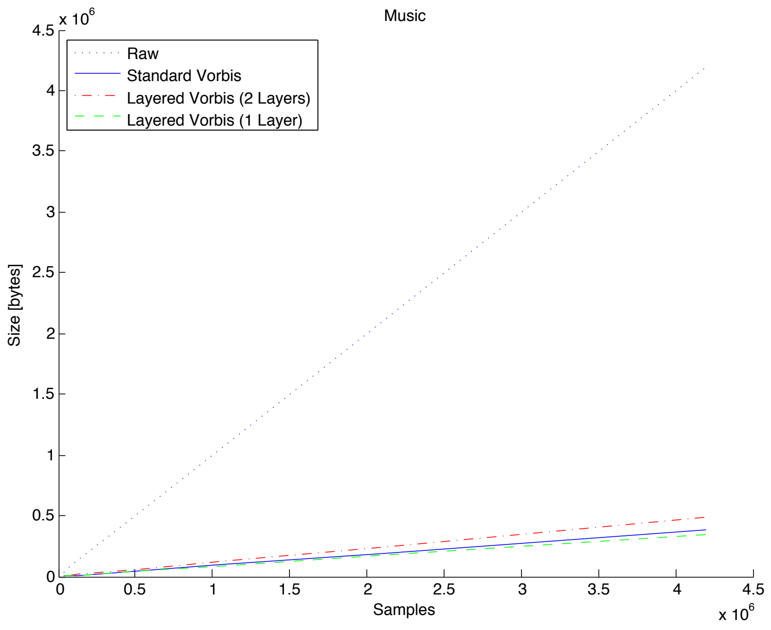


Fig. 11: Size properties of a music signal.

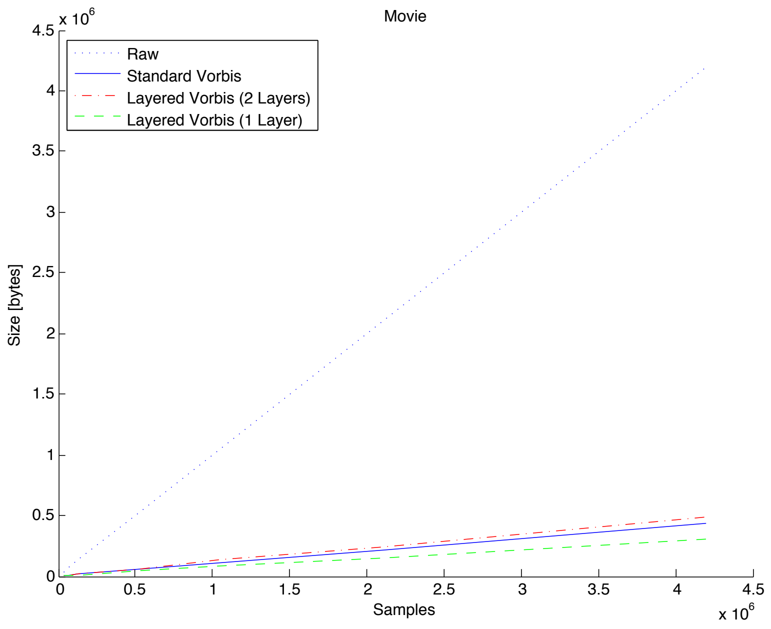


Fig. 12: Size properties of an audio excerpt from a movie.