Ogg Vorbis and MP3 Audio Stream charecterization

By:

Ayman Ammoura Franco Carlacci

Instructor: Dr. Ioanis Nikolaidis

Last Compiled: September 22, 2002

Contents

List of Figures

Ogg Vorbis and MP3 Audio Stream Characterization

Ayman Ammoura Franco Carlacci
University of Alberta
Instructor: Dr. Ioanis Nikolaidis

September 22, 2002

Abstract

This paper presents a detailed discussion about the newly emerging multi media CODEC known as Ogg Vorbis. The focus remains on the audio part of Ogg Vorbis. To understand how this format differ in audio quality, an analytical comparison with the "leading" audio CODEC MPEG layer 3 is conducted. This leads to a discussion in which the physical and logical properties of such streams is identified. In this report several avenues are explored in an attempt to characterize Ogg bitstream traffic. Most of the experiments conducted were based on some modifications to the library code provided by the Xiph.org as part of their open source distribution.

1 Introduction

Audio streams make a large percentage of streamed data over the internet. One of the main reason that made this possible, besides fast ethernet connections, is the fact that *audio compression* algorithms are capable of producing file sizes that are a fraction of the original input CD audio tracks. One of the absolute leaders in audio compression is the MPEG layer 3 CODEC known as MP3. The input to an MP3 encoder is a wave file and the output is a compressed version that is stream,

One of the main motivating reasons for developing Ogg Vorbis has to do with patents. Unlike the most popular audio compression CODECs, Ogg Vorbis is fully open source , non-proprietary, patent-free, and royalty-free software. It is a general-purpose compressed audio format for mid to high quality[1] audio and music[4, 10]. One of the best features that made Ogg

[1] That is roughly between 8kHz to 48kHz with at least 16 bit polyphonic encoding.

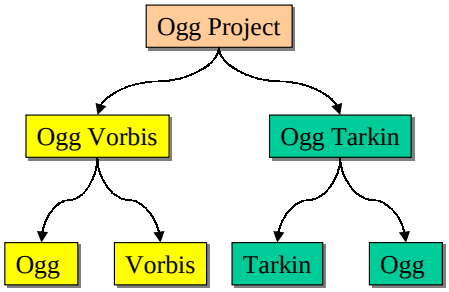

Figure 1: The general Ogg project that include the video project Tarkin.

Vorbis well accepted is the ability to adjust its bitrates according to some quality or average parameter. This is known as *variable bitrate* encoding and *average bitrate* encoding, respectively. This places Vorbis in the same competitive class as audio representations such as MPEG-4 (AAC), and similar to, but higher performance than MPEG-1/2 audio layer 3, MPEG-4 audio (TwinVQ), WMA and PAC [7].

Ogg is the term used to refer to a group of several multimedia and signal processing projects that are underdevelopment by the Xiphophorus organization [9]. Two such project are currently in active development and one of which has already been released, audio Ogg, and the other is still not officially released, video Ogg. The Ogg project video codec is called Tarkin.

This report consists of three segments. First, the formats and definitions that make up the Ogg Vorbis audio is given. This discussion is focused on the formatting and the transformation of audio sound signals into transportable packets (Section 2.2). In the second portion we present some of the characteristics of Ogg Vorbis with respect to the encoded signal itself. The exact details of the mathematics behind the actual encoding of bits and the subsequent appropriate compression is beyond the scope of this paper. Instead, by examining the encoded, decoded, compressed and uncompressed signals, it is possible to observe the effect of the encoding and the characteristics of the resultant bitstream. This is done by taking a close look at the frequency domain (Section 3).

Once the original audio signal has been encoded and appropriately formatted for transporting, it is important to examine the statistical properties that Ogg Vorbis traffic exhibits. This is done in the third portion of the paper. Using code that scans Ogg streams, our initial results suggested that the traffic could be classified as *self-similar*. This notion is exploited further in Section (4).

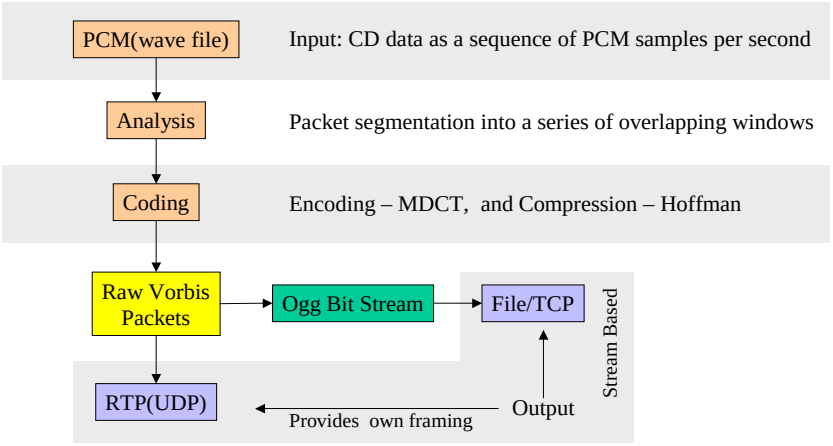

Figure 2: The complete process of input, analysis, encoding, and output.

2 Which Came First, the Ogg or the Vorbis?

It is often the case that the terms "Ogg", "Ogg-Vorbis" and "Vorbis" are used interchangeably without really knowing which of which is which! In simple terms, *Vorbis* is the name of the CODEC that does the encoding, decoding and the compression while *Ogg* is the term used to refer to a Vorbis encoded audio stream. That is, a network carries the Ogg and the client strips the Ogg and decodes the Vorbis. More details as to what this entails is presented in this section.

2.1 Vorbis: The CODEC

Given a wave file or a **PCM** (Pulse Code Modulation) source, the task is to produce an encoding that maintains a reasonable[2] audible quality as well as compresses the output data. This is precisely the task of the Vorbis CODEC. Vorbis encodes short blocks of PCM data into raw bit-packed data packets called *raw Vorbis packets*. Once these raw packets are generated, the following step depends on the intended use. For transport mechanisms that provide their own framing, synchronization and packet separation, such as **RTP** (Real-Time protocol), for the purpose of stream-based storage (files) or transport (TCP) Vorbis uses what is called an Ogg bitstream format (Figure 2.

[2]What does "reasonable" really mean? This is discussed the Section 3.

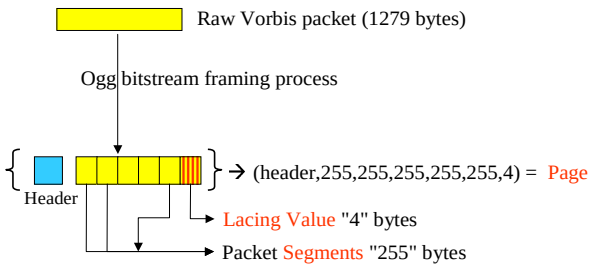

Figure 3: The raw Vorbis packets are logically segmented into a set of segments of length 255 bytes. The last segment is the *lacing value* which must be < 255 and is used as a recapture sequence marker.

2.2 The Ogg Bitstream

The Ogg bitstream format provides the raw Vorbis packets with framing structure, synchronization, synchronization recapture after errors, landmarks for seeking, as well as the information needed to be able to reproduce the packets from the frames without relying on the decoding process for boundary recovery [10].

Figure 3 illustrates how the Ogg framing is used to convert the raw Vorbis packets into an Ogg bitstream. As the name suggests, this is applied when the destination for the encoded audio is intended for streaming, ie. TCP or file storage. There are three terms that are needed to define the Ogg bitstream, these are *page, segment*, and *lacing value*. As indicated earlier there are no restrictions on the Vorbis packet size, which makes it necessary to place a "framing convention." Such convention allows the decoder, ogg123, an easy mechanism to synchronies and recover from errors.

A *logical Ogg bitstream* is an ordered set of pages that belong to one Ogg bitstream. Since it is possible to multiplex more than one Ogg bitstream, the term Ogg *physical bitstream* is used to define a stream that consists of one or more logical bitstreams. To clarify, multiplexing one or more logical Ogg streams does not include stereo audio. What is meant by multiplexing is the ability to include multichannel tracks in a single Ogg media streams. Another example is in Video where audio (single or multichannel track) and video are multiplexed into a single Tarkin stream. Just a side note that worth mention here is that with Ogg Vorbis it is possible to multiplex an MP3 audio track with an Ogg video stream.

In the example given in Figure 3, assume that a packet of size 1279 bytes has arrived to be incorporated into an Ogg bitstream. To construct

one page, a header is needed to store the number of logical segments that are going to be generated. The convention here is that every segment except the last must be exactly 255 bytes long. Values that are less than 255 are reserved for a special type of segment that is called the lacing value. For instance, upon the receipt of a page, if the some of the data was lost,then the decoder will seek the lacing value to recover and re-synchronies. It should be noted here that it is indeed possible to have an Ogg stream that consists of a *single* page.

In in an effort to avoid confusion, it is important to draw a distinction between three types of bitstreams. The scarce bits-and-pieces of literature available about Ogg Vorbis seem to describe three bitstreams; Ogg logical, Ogg Physical and *Vorbis bitstream*. In the above discussion the Ogg bitstreams have been defined as a collection of raw Vorbis packets that have been formatted used Ogg framing. Now, what if the Ogg framing is not needed? For this purpose, the Vorbis bitstream is used. More details is given next.

2.3 Using RTP Over UDP: Vorbis Without The Ogg

As indicated in Section 2.1, once the raw Vorbis packets are obtained, the next step is determined by the method by which these packets are to be consumed. If these packets are to be streamed, then the Ogg formatting is needed for synchronization and framing; however, if the transport mechanism used does provide its own framing, then the Ogg formatting is unnecessary. This subsection includes a description of how Vorbis encoded audio may be formatted for use as an RTP payload type [8]. Applications typically run RTP on top of UDP (User Datagram Protocol)as part of the transport layer protocol UDP provides a connectionless service to application-level procedures [6, 12], this means that this is unreliable service. The natural question to pose then is how can this be use for the transportation of audio media?

To use UDP as a transport protocol for real-time traffic, some functionality has to be added. Functionality that is needed for many real-time applications is combined into RTP. The services that RTP provides include timestamping, sequence numbering, payload identification and source identification [11]. As far as the transportation of the Vorbis packets, the most significant information in the RTP header is the *timing information*. The sender *timestamps* each RTP packet with the point in time the first sample in the packet was encoded. The receiver then uses these timestamps to reconstruct the original timing before the audio can be played back (decoded

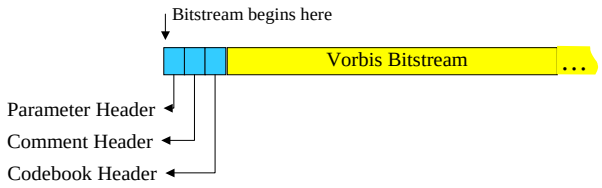

Figure 4: This is the Vorbis bitstream structure. It is designed to be used either inside an Ogg bitstream (TCP and files) or as-is (RTP over UDP).

- ogg123).

There are three header packets that must begin any Vorbis bitstream; these are parameter header packet, comment header packet and codebook header packet. The comment header is used to specify information such as the stream type, version, number of channels, sample rate, nominal bitrate (ABR), minimum and maximum bitrates (VBR) etc. The text comment header is the second (of three) header packets that begin a Vorbis bitstream. It is meant for short, text comments, not arbitrary metadata; arbitrary metadata belongs in a metadata stream (usually an XML stream type) [13].

Vorbis uses codebooks, the third header, to perform Huffman encoding on vectors of floating point values. These values are stored in the analysis packets that are also called the "raw Vorbis" packets. Technically, a codebook is a one-to-one mapping between a set of these vectors and a set of Huffman codes, which are bitstrings of varying lengths [13]. To be a little more specific, the process of creating Vorbis packets is called *Bitpacking* which arranges the variable sized words of the back-end coding into a vector of octets without wasting space. The octets produced by coding a single short-time audio segment is one raw Vorbis packet.

2.3.1 Codebook Transmission

In order to decode a Vorbis stream, a set of codebooks[3] are needed. The RTP takes care of error recovery but without the codebooks the Vorbis player (decoder) will not be able to use any of the packets transmitted. These codebooks may vary for for each logical bitstream, for example n users listening to Vorbis net-radio station. This gives rise to an important issue that remains an open issue [8].

A client requesting a connection to a multicast RTP Vorbis session needs to get the first set of codebooks *intact* in some manner. How can this be

[3]The Vorbis codebooks vary between 4 kilobytes and 8 kilobytes

done? As of the date of this writing, there has been no final solution in which this requirement is guaranteed. Some of the possibilities proposed by the Xiph.org foundation include:

1. Use the session description protocol (SDP) to include the first set of codebooks.
2. Use multicast to broadcast a second Vorbis stream that contains including the codebooks.
3. Define a method that a client can use to request the codebooks via RTCP.
4. Periodic retransmission of the headers.

3 Experimental Methodology

The general motivation for this project is to study the statistical properties of either the MP3 audio compression algorithm or the Ogg Vorbis. Having said that, it is important to take a closer look at both to examine some of the differences. The final analysis of an studio stream was performed on the Ogg Vorbis stream (Section 4). This choice was not based on a particular reason it is just the fact that it is new, and the dynamic range provided by the Vorbis encoder seem to be better than that of the MPEG as well as the fact that Ogg is an open source project.

It is difficult to isolate the methodology of the research from the results obtained. For the most part, the interest was to try and locate interesting features of the encoded streams, either from the encoding perspective or from the stream perspective. As a result, the remainder of this paper is divided into two main parte. We first present the methods used to analyse a the Ogg and the MP3 sample pieces. In the second segment, the discussion is focused on the statistical properties of an Ogg Vorbis stream, including the multiplexing of more than one stream over a network.

The original PCM files were captured, "ripped," from audio CD's into wave (WAV) files using creative studio. From these files, LAME version 3.2 and Oggenc (the Ogg encoder) were used to produce the encoding into the MP3 an Ogg respectively. Both encoders are capable of generating the variable file format that is desired for this study.

3.1 Obtaining Sample Pieces

Each wave sample obtained was converted into Ogg and into MP3. The sample length for each of the samples was 20 seconds[4]. There are a total of 23 samples in the corresponding we page for this project [1]. The following are the two commands used for generating the files:

```
lame -v -V 2 sample.wav sample.mp3
oggenc -q 2 sample.wav -o sample.ogg
```

Once these samples have been encoded, a closer look would easily reveal that this would *not* result in samples that are compatible with one another, the reason for this is the fact that both encoders have their own interpretation of the quality parameter. That is, level "2" quality in Lame does not necessarily mean the same bandwidth produced by the Vorbis encoder.

3.2 Waveform: Ogg Verses MP3

There are two ways by which an encoder can be studied; by exploring the back-end algorithms or by listening to its output. Those two methods define the *objective fidelity* and the *subjective fidelity* of the analysis respectively. For this project, the latter seemed more appropriate due to time considerations and the fact that the output of the encoders is very enjoyable. As a matter of fact, the real reason why the subjective fidelity is very important when discussing Ogg Vorbis is the fact that Vorbis compresses audio by eliminating inaudible or undesirable audio frequencies based on spatial psychoacoustic models. What does that mean? It means that the Vorbis developers used experiments that illustrated how humans recognize various sound frequencies and to what degree. This information was then used so that certain frequencies and/or certain frequency changes are not encoded.

Initially in the project, we have collected several software pieces, plugs and demo packages in an attempt to find a code that would plot and analyse samples of both Ogg and MP3 input. after obtaining such tools the validity of this was questioned. What is to be answered here is what is being looked at and compared here? Looking at the wave form of an Ogg sample, basically assumes that we have agreed on some tacit interpretation of the encoded sound that is done by the plugin. That is, what is of importance is to know what was changed, and what was removed[5]. As a result, it was decided that

[4]No particular reason for choosing this time duration, but 20 seconds seems to be long enough to audibly identify the sound quality and visually examine the behaviour of the encoded samples.

[5]Both MP3 and Ogg Vorbis are forms of lossy compression.

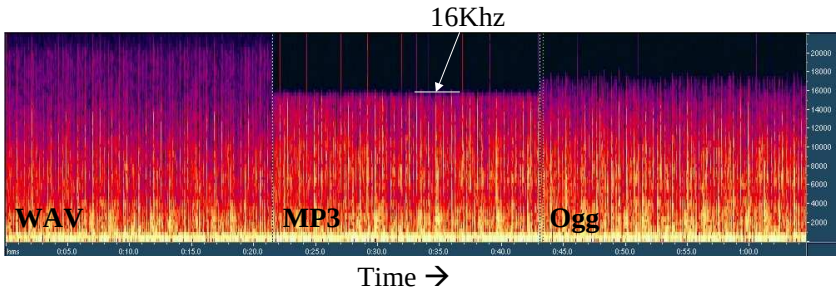

Figure 5: Three 20-second samples appended into one file. Left is the original wave sample, followed by the corresponding MP3 and Ogg Vorbis encoding.

to obtain objective results, the following must take place:

1. Take a 20 second wave sample.
2. Encode both samples into Ogg Vorbis and MP3 using ABR with parameters that are least biased.
3. Convert the two pieces back to wave formate.
4. Compare the three wave pieces side-by-side.

The main observation here is to observe what happens what to the high frequency frequency ranges upon encoding. It seems that frequencies above the 16KHz seem to have been lost after the compression process in both the Mp3 and the Ogg Vorbis (indicated by the black region - no energy). However, from Figure 5 it is observed that the cutoff is not as sever in the Ogg sample as it is in the MP3 sample. This indicates that the Ogg algorithm is more capably of encoding some dynamic range in the upper frequencies. A look at the close-ups in Figures 12 and 11 easily confirms this observation.

The way that both encoders treat high frequencies can be confirmed using Figure 6. What is important in this figure is the ability to examine the entire 20 second sample, record the power (decibels) of the frequency for every millisecond, and then *averaging* over the entire period[6]. The result of averaging both the Mp3 and the Ogg Vorbis samples is seen in this figure. It should be clear here that the Ogg Vorbis does indeed maintain, although diminishing, some of the higher frequencies. In contrast, the plot representing the Mp3 sample experiences a sudden large drop above the 15KHz

[6]This was produced using baudline freeware.

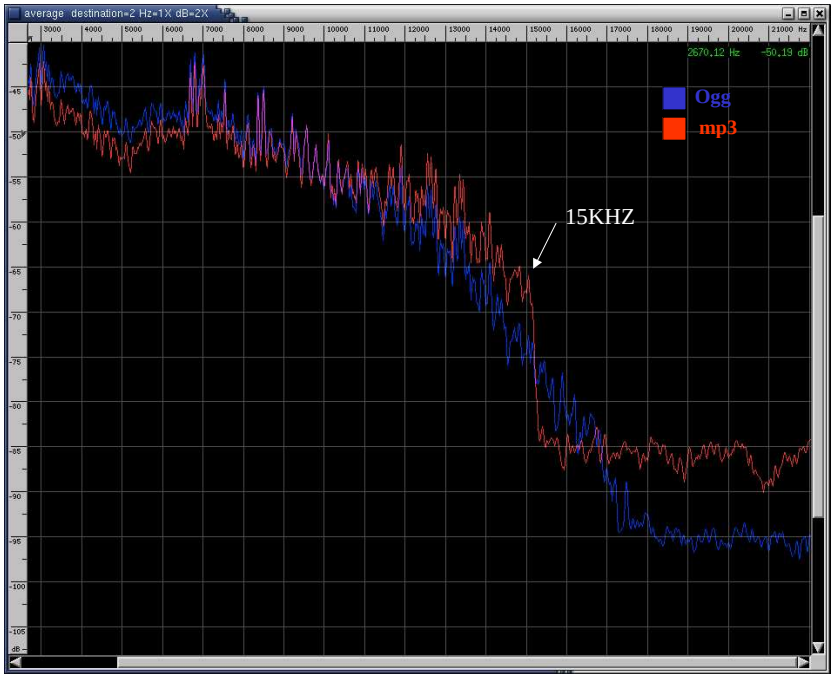

Figure 6: An average frequency graph for Mp3 and Ogg Vorbis. Sample is a 20-second piece from Bob Marley.

Figure 7: This is the statistics over the 20 second sample (Wu Tang) track. From the left, Mp3, Ogg and Wav formats.

mark. This corresponds to the black "strip" seen on top of the mp3 sample in Figure 5.

One feature that does exist in both encoders and illustrates how psychoacoustics are used in encoding music can be seen from the results in Figure 7. The *average power* is measured in decibels (dB). Decibels are defined in Terms of power per unit surface area on a scale from the threshold of human hearing, which is set at 0 dB, up until the level of pain, set about 120-140 dB. This indicates that the Ogg Vorbis encoding does "add" more power to the signals it encodes and both do add more than the original encoded sample.

3.3 Sound Quality

In order to comment the subjective fidelity of both encoders, various samples of varying attributes were listened to and compared. During the sampling process some attributes were noted. In the hope to further understand or prove these observed sound attribute, the following frequency domain analysis is reported.

3.3.1 Drop That Bass

One sound attribute that is noticed in pieces containing significant low frequencies on a regular basis (low base). The sample that has been chosen is from the Method Man - Wu Tang Clan. This is a Rap piece with many instances of ultra low frequency. What is needed here is to be able to pick a wave form that presents the low frequency range and compare that with

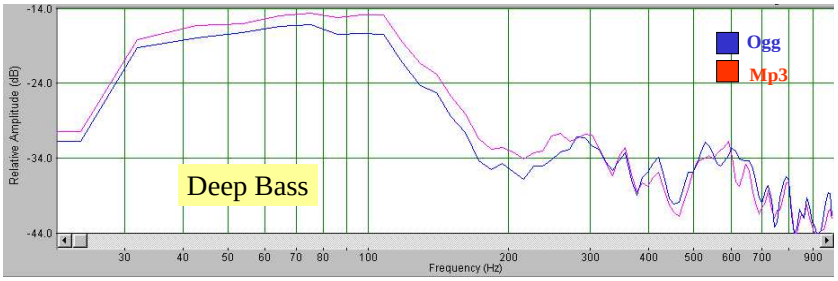

Figure 8: Method Man (the Wu Tang Clan) 50-trace average. This is the average over the last 50 traces in the sample. The focus of the wave form is on the ultra-low frequencies. Mp3 does deliver approximately +2dB gain over Ogg in this range.

the original and the Mp3waveform. Once again, this is difficult to do since is almost impossible to be able to isolate manually traces. Instead, we have edited the Wave sample into a 3-second sample the end of which is a long deep-base sound. As before, this sample was encoded to Mp3 and Ogg and then decoded back to Wave. Then Spectra Lab Pro was used to average the last 50 traces of the sample. The resulting average frequency can be seen in Figure 8. The ultra low base known as "base-drop" that can be produced audio speakers is within the range 20Hz - 70Hz[7].

3.3.2 General Sound Quality

It is true that the be able to detect the minute changes in encodings, very fine and well-toned instruments have to be used. The samples that we have listened to were compared using a headphone set-that is isolates a good deal of the outside noise. Besides the ultra low frequency mentioned above, Ogg Vorbis does have a "cleaner" sound. What does this mean? For the most part, while listening to vocal artist, samples from LeAnn Rimes [1], Ogg seems to deliver a better sound through a more noticeable stereo separation. That is, with Ogg encoded pieces it is easy to note that a sample is indeed in stereo. Mp3 seems to produce better stereo at higher bitrates.

In addition, the Signal to Noise Ratio (SNR) was computed for different samples. The SNR is the ratio of the signal peak level to the total noise level. These levels are measured by their power, and hence, the SNR is expressed in decibels (dB). SpectraLab (the software used)computes the SNR by searching the entire spectrum to find the peak frequency and then

[7]Anything lower than 30Hz can only be reproduced by professional speaker systems.

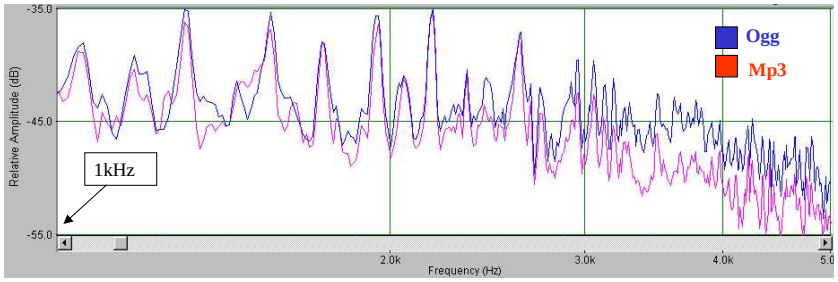

Figure 9: This figure shows the Mid-Low frequency range. Notice how Ogg is more dominant through the the 5KHz mark.

calculates the total noise power in the remaining spectrum. That is, a high SNR indicates less distortion (since more signal than noise) while low SNR indicates greater distortion.

1. Wav: SNR (left) = 4.88 dB, (right) 5.65 dB
2. Ogg: SNR (left) = 6.92 dB, (right) 7.51 dB
3. Mp3: SNR (left) = 9.69 dB, (right) 10.20 dB

These results seem to favour Mp3 as a sound with a "less distortion," but this measure is objective and to judge the sound by listening is indeed subjective. It is probably because of the greater stereo separation in Ogg that it does sound better! Figures 9, 10, 11, and 12 illustrate in more detail the various attributes of Ogg. All these figures have been obtained from the same sample (LeAnn Rimes) considering only the average of the final 100 traces in the sample. What is interesting is that Ogg not only reproduce frequencies in the range 15KHz - 16KHz better than Mp3 but it also preserves the original dynamic range of the original wave file.

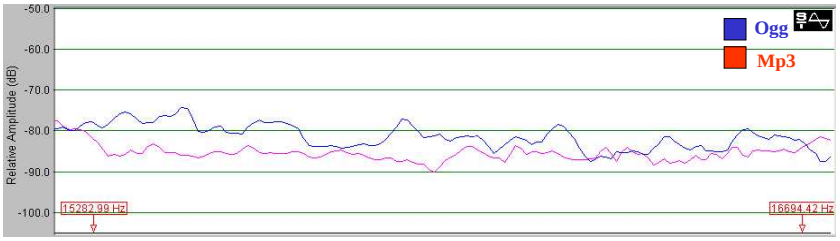

Figure 10: A closer look at what happens at high frequency ranges. Ogg has a better handel on the frequency ranges above the 15kHz mark when compared to Mp3.

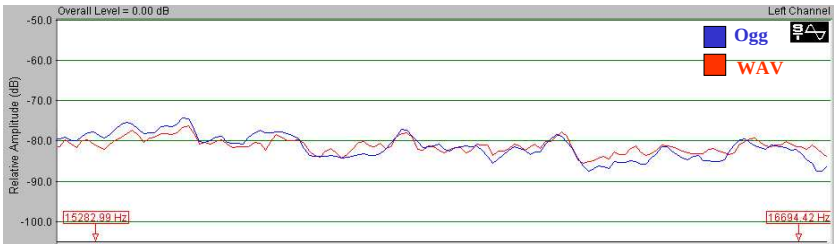

Figure 11: The important thing to notice here is that Ogg remains "brilliant" at high frequencies. It is almost identical to the original wave file wave form.

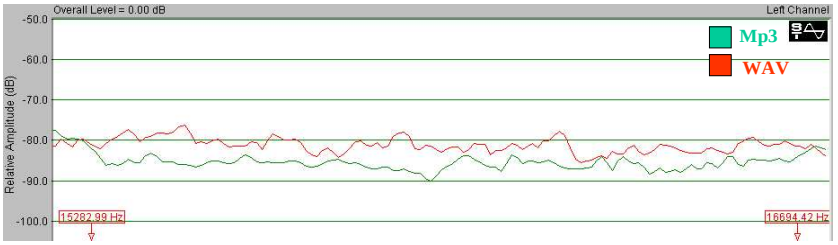

Figure 12: This figure is used to further illustrate the "strait cut" exhibited by the LAME encoder at high frequencies.

4 Autocorrelation and Self-Similarity

As one of the objectives of this project is to analyse the Ogg bit stream, it is important to be able to extract some of the encoded information. The bitrates as they are encoded in each frame is used to examine the relation between that and time.

We have modified the mp3stat program so that we are able to extract the bitrates for every frame encoded in the Ogg file. This data was output to a temporary file that contains a sequence of the bitrates as the file is decoded.

Looking at the source code, the bitrates that are generated correspond to one per frame. So in our subsequent development we will treat the bitrate values as a vector or series:

$$\begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix}$$

The value n represents the number of frames found in the Ogg file. This value will vary depending on the audio sample used .In our attempts to examine the self similarity of the data that have been obtained the following experiments were done in order.

4.1 Time Series Plots

The first intuitive plot that may be used to visualize the extracted bitrates is to plot the values chronologically. That is, a graph that plots the frames (time) verses their corresponding bitrate. The plot is seen in Figure 13.

It is apparent that the bitrates distribution seem to be narrowed down to two regions, upper and lower. It is important to note again here that the encoders were not restricted by any external parameters to try and group the output to fit an average bit bitrate. This point is re-enforced by the graph seen in figure 14.

4.2 Examining Temporal Correlations

From the above it is observed that the bitrates are gathered into two regions but without any time correlation between the points. What needs further examination is the way these points are generated. In particular, does the encoder generates a sequence of points in one region and then the following sequence in the other, or it merely oscillates between the two?

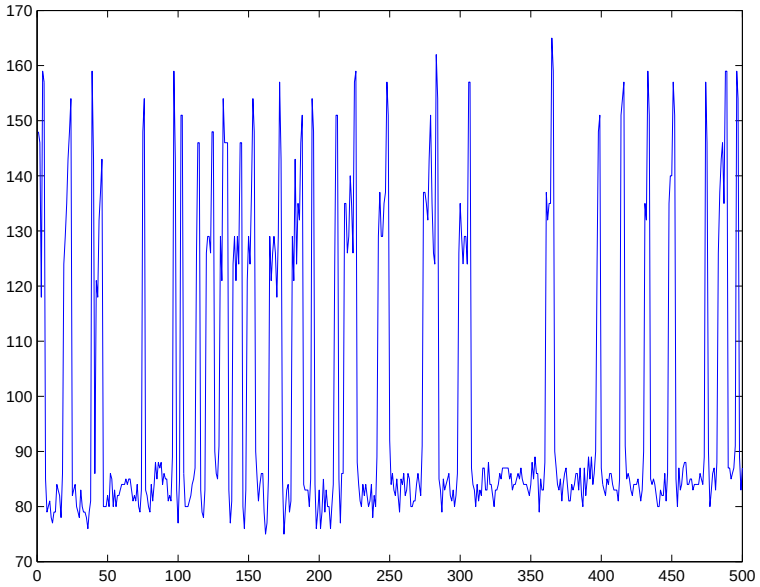

Figure 13: frame number versus bitrate for the file macy.ogg

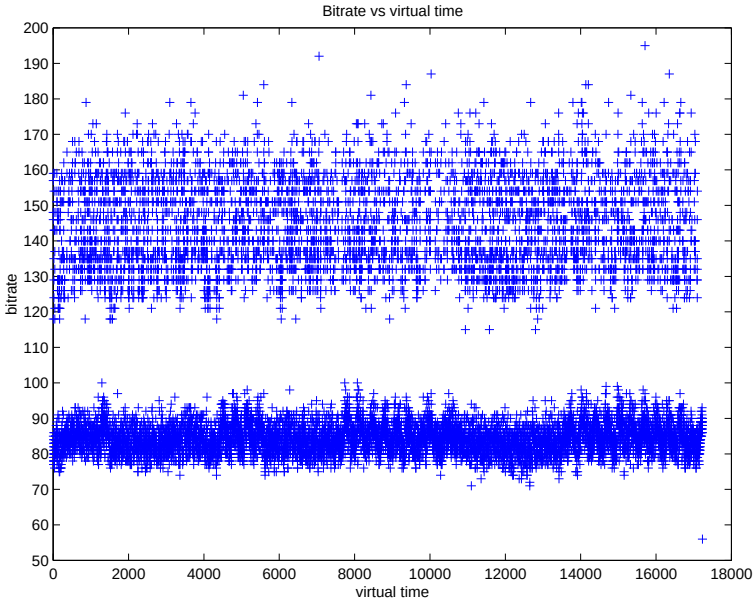

Figure 14: Frame number versus bitrate for the file macyvbr.ogg plotted as discrete points

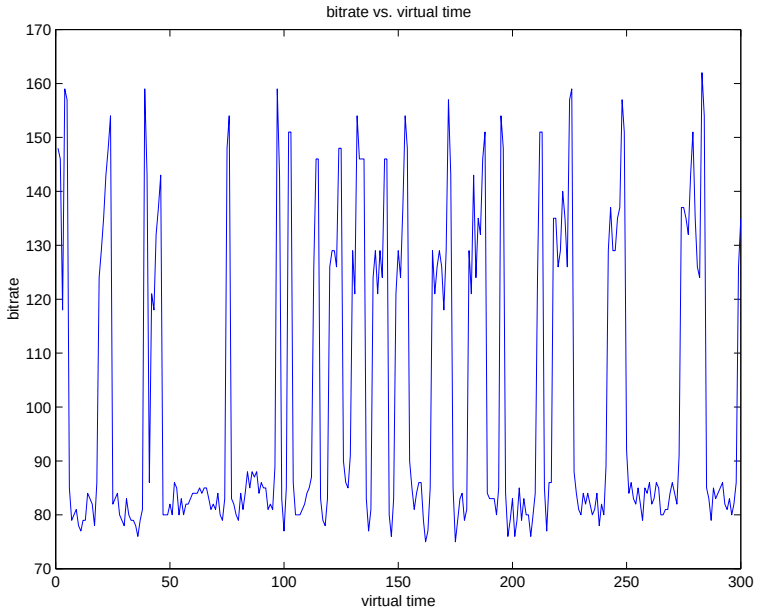

Figure 15: First 300 frames of the graph frame number versus bitrate for the file macyvbr.ogg

For that purpose, the plot in Figure 15 was generated. The data here is a subset of the data seen in Figure 13. Such a smaller subset makes it possible to visualize how these points are generated in time. As is evident from the graph the encoder seems to be generating frames that are in one of the regions seen in Figure 14 and then there is a jump to the next region were the encoder generates the next frames and then moves one to the first region.

4.3 Lagged Scatterplot

The following analysis was performed using Matlab. Scripts were written that allowed us to compute both the lag-k scatterplots as well as the variance-time plots.

Our search for self-similarity started by looking at the lagged scatterplot [3] of our vector. A lagged scatterplot is a scatterplot of a time series against itself offset by one to several time units. That is, the lagged scatterplot for lag-k is a simple scatterplot of the last $n - k$ observations against the first $n - k$ observations , where n is the size of the time series. This plot is important because alignments of points in some direction indicate some

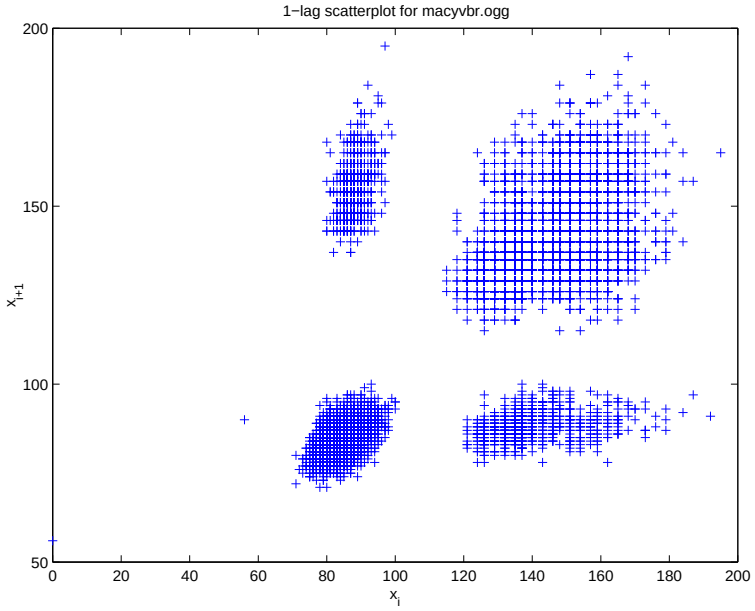

Figure 16: Lag 1 scatterplot for macyvbr.ogg

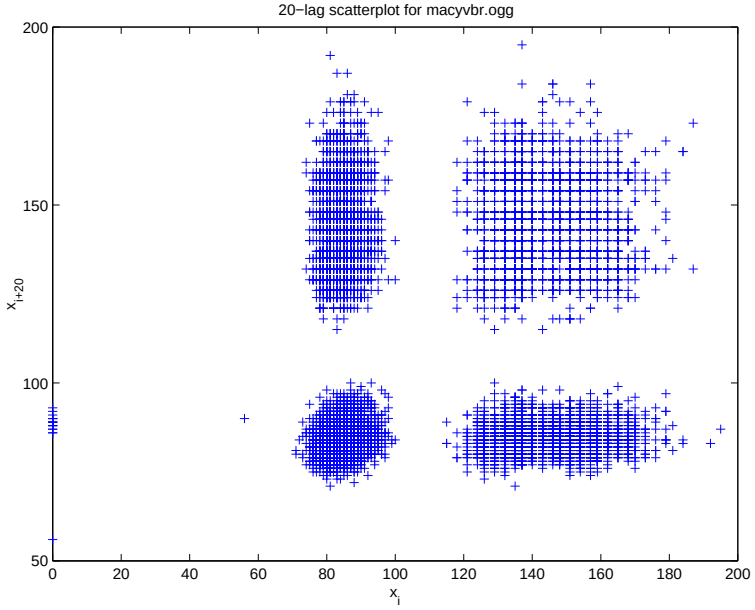

Figure 17: Lag 20 scatterplot for macyvbr.ogg

degree of correlation. Also if we look at the plots for several different lags k we will see that the general structure of the scatterplot remains the same suggesting that the underlying process might be self-similar.

A lag-1 scatterplot and lag-20 scatterplot are shown in figures 16 and 17 We can see how the bitrates are segregated into two very distinct groups along the line $y = x$. The clouds in the upper left and lower right region of the map represent instances during which there is a transition from low to high (upper left) and high to low (lower right) bitrates. It is clear that the two plots have an almost identical structure suggesting a self-similar process.

4.4 Variance-time plots

Our next step was to attempt to estimate the degree of self-similarity by evaluating the Hurst parameter. LeLand et. al [5] suggest three different approaches: (1) time-domain analysis based on R/S statistic, (2) variance-time plots and (3) periodogram-based analysis in the frequency domain. We used the second approach.

The variance-time plots technique is based on the observation that in a self-similar process, the variance for the aggregated process $X^{(m)}(m = 1, 2, 3, ...)$ [5] decreases linearly for large m in log-log plots against m. Values of the estimate of the slope β of the best-fit line between -1 and 0 suggest self-similarity and the Hurst parameter can be estimated by $H = 1 + \beta/2$. In [5] a typographical error must have slipped in since the authors suggest that $-1 < \beta < 0$ implies that the Hurst parameter $H = 1 - \beta/2$

Figure 18 show the variance-time plot for a piece by Macy Gray. The value of the Hurst parameter H is found to be equal to 0.5580. It is known (and can be mathematically demonstrated) that $H = 0.5$ corresponds to non-self-similar data, while $0.5 < H < 1.0$ corresponds to self-similar data. In their paper Leland et al. suggest that high values for H implies a high degree of "burstiness".

We pointed out earlier that the bitrate values fluctuated between two ranges. Knowing this, we decided to investigate the duration of virtual time that the bitrates spent in a given range before moving over to the other. We used Figure 14 to give us a cutoff value and then scanned the bitrate vector counting runs of values below the cutoff as well as above the cutoff value. The results are given in the histograms in figure 19 and 20. The intent here was to see if the self-similarity was a result of some type of on-off, or in our case low-high, heavy-tailed process. The results are inconclusive.

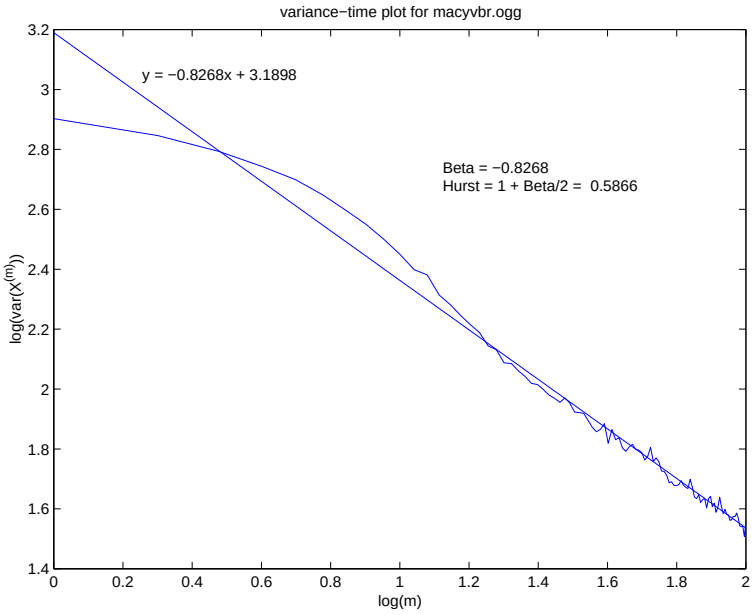

Figure 18: Variance-time plots with β and hurst parameter and least-squares line.

4.5 Further Results

As it may be on the mind of the reader at this point, would the above hold if a different type of musical sample was chosen? To reveal or dispel and music-type dependence, the same exact tests have been done on two other full, and completely different, music pieces.

The lag-1 lagged scatterplot for the two pieces are presented in figures 21 and 22. The first sample is the Strauss piece and the second is the Enya. The Strauss sample is a very complicated instrumental piece. The song from Enya, on the other hand, is mellow piece with a vocal track laid over a simple musical accompaniment. We have also included the variance-time plots for these two pieces in figure 23 and 24. The interesting thing as it is clearly illustrated in these pictures is that the basic properties hold for these samples.

5 Concluding Remarks

Once the spectral and waveform tools are installed and ready to use, a beginner to this area will be faced with many considerations and alternative

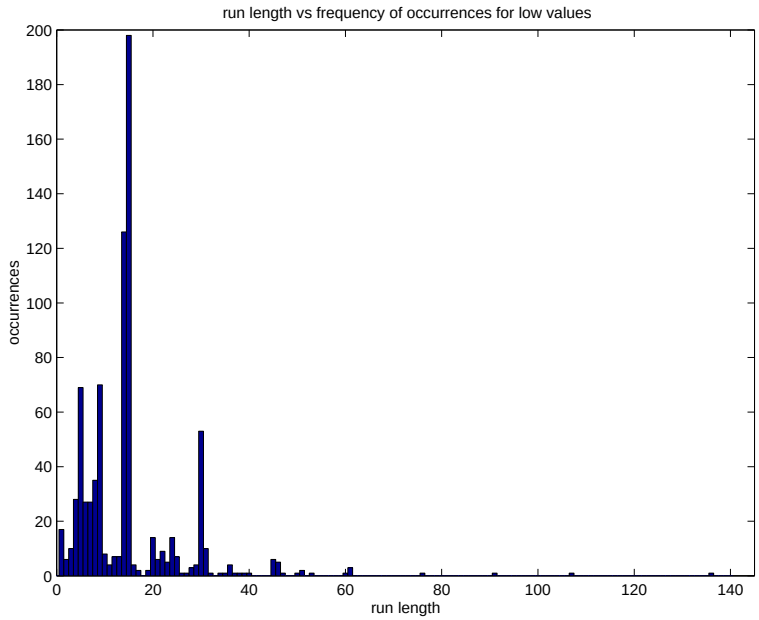

Figure 19: Run length versus frequency of occurrences for values below the cutoff.

as to how to conduct a comparative analysis of some CODEC. Issues such as parameter usage, sample length, and sample type have been easily dealt with. However, comparing waveforms is more difficult and the averaging approach that was taken in this studies was indeed our solution. The problem is that some reports ,such as the one by Andrei Aspidov [2], present a rather large number of comparative waveform figures of Ogg and Mp3 without any indication of *how* these frequency traces were obtained. As we have attempted obtaining our own results it became immediately clear that it is next to impossible to isolate a single trace so that identical traces from wav, Ogg, and Mp3 can be compared. Since how a CODEC deals with a given range of frequencies was the main concern, averaging several identical frames seemed reasonable for comparative purposes. From the testing that we have done Mp3 and Ogg seem to excel in different areas. Ogg Vorbis does provide a true variable bitrate streams that indeed preserves most of the details and definition when needed. This is especially true in the case of high frequencies where Ogg will encode traces above the 15kHz in some instances as determined by the psychoacoustic model used. For pieces were he ultra low base is needed, Mp3 delivers more power (dB) to the the lower

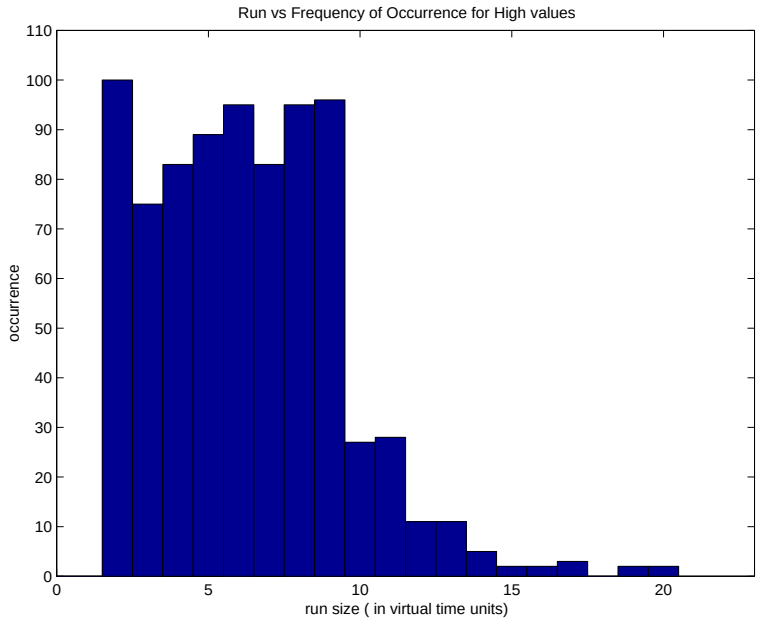

Figure 20: Run length versus frequency of occurrences for values above the cutoff.

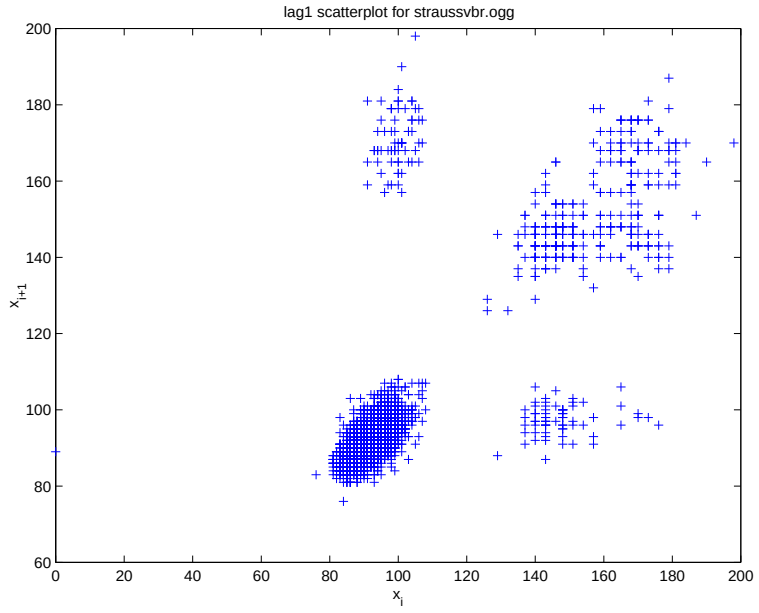

Figure 21: lag 1 scatterplot for straussvbr.ogg

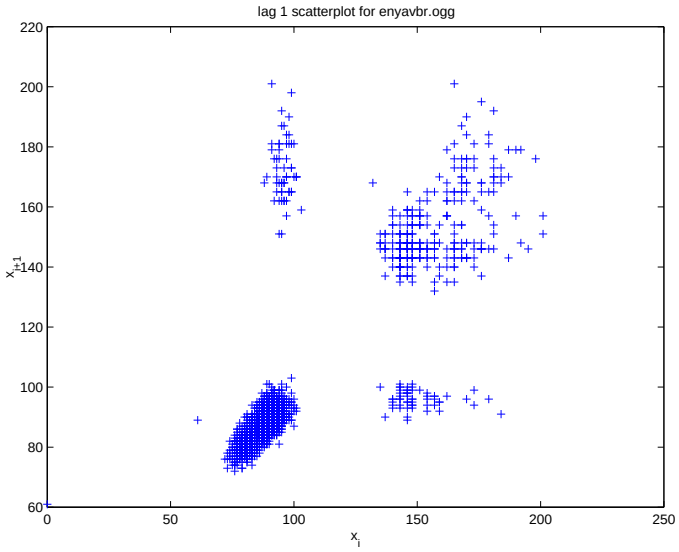

Figure 22: Lag 1 scatterplot for enyavbr.ogg

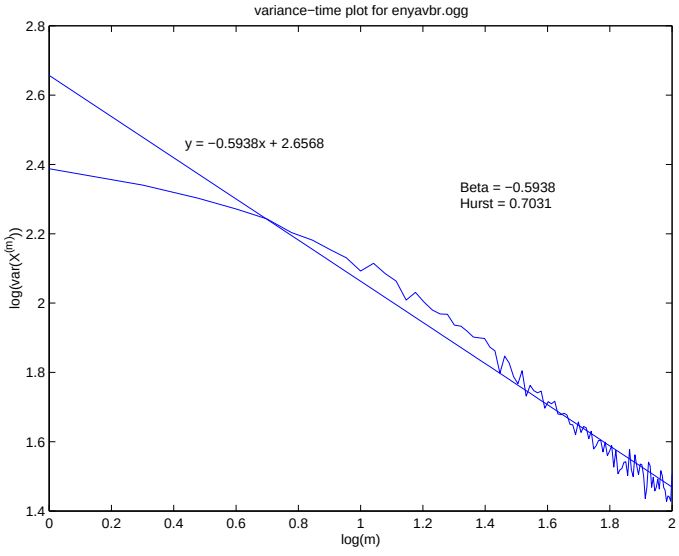

Figure 23: Variance-time plot for enyavbr.ogg

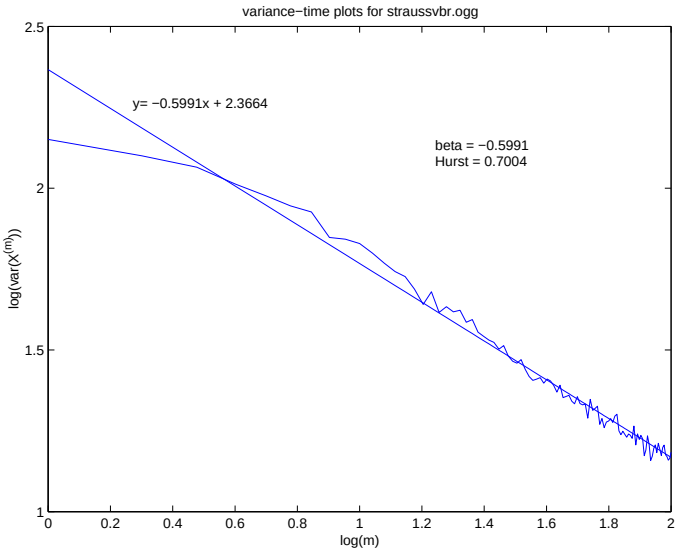

Figure 24: Variance-time plot for straussvbr.ogg

frequencies (30Hz - 100Hz).

As far as Ogg traffic is concerned, evidence seems to point to the fact that the bitrates output by a Vorbis decoder are self-similar with some reasonable high degree of burstiness. The k-lag scatterplots and the variance-time plots support this. Attempts at trying to explain this behaviour in terms of some underlying process were inconclusive. We would like to investigate this phenomenon further by trying the other two approaches suggested in [5]. Also the idea of simulating this bitrate generation and listening to the result sounds (excuse the pun!) like it might be interesting to try.

appendix

References

[1] Ayman Ammoura and Franco Carlacci. Web page for the oggvorbis and mp3 project. Internet Page, April 2002. `www.cs.ualberta.ca/~ayman/OggVorbis/`.

[2] Andrei Aspidov. Ogg vs. lame. *Digit Life*, page Internet Publication, 2001. `http://www.digit-life.com/articles/oggvslame/index.html`.

[3] George E.P. Box and Gwilym M. Jenkins. *Time Series Analysis*. Holden Day Publishing, 1976. ISBN:0-8162-1104-3.

[4] Andy Faulkner, Rich Smith, Brad Baylor, Jim Bailey, Paul Mack, Jim Lemaster, and Tom Hartel. Running a net radio station with open-source software. *Linux Journal*, 1(81), January 2001. Article available through ACM digital library.

[5] Will E. Leland, Murad S. Taqq, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In Deepinder P. Sidhu, editor, *ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.

[6] David E. McDysan. *ATM Theory and Application*. McGraw-Hill Series on Computer Communications, first edition edition, 1994.

[7] Jack Moffitt. Ogg vorbis–open, free audio–set your media free. *Linux Journal*, 1(81), January 2001. Article available through ACM digital library.

[8] Jack Moffitt. RTP payload format for vorbis encoded audio. Inetrnet-Draft, Febuary 2001. `http://www.xiph.org/ogg/vorbis/doc/`.

[9] Jack Moffitt. The xiphophorus home page. Internet HTML Page, 2001. `http://www.xiph.org/ogg/index.html`.

[10] Monty. Ogg logical bitstream framing. Web notes from the developers, Febuary 2001. `http://www.xiph.org/ogg/vorbis/doc/framing.html`.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rfc 1889: Rtp: A transport protocol for real-time applications. Network Working Group Memorandum, January 1996. `http://www.faqs.org/rfcs/rfc1889.html`.

[12] William Stallings. *Data and Computer Communications*. Macmillan, NJ, fourth edition edition, 1994. Page 587.

[13] Ogg Vorbis Team. Ogg vorbis comment field specification. Web notes from the developers, Febuary 2001. `http://www.xiph.org/ogg/vorbis/doc/v-comment.html`.

A Tools Used for Audio Generation and Analysis

As Ogg Vorbis is still a newly emerging CODEC, the documentation and the set of support tools are at a minimum. In our attempt to analysis the sound quality and the statistical properties of Ogg Vorbis traffic several tools were downloaded and experimented with.

On the back end we have used a statistical tool called **Mp3Stat** that uses the Ogg library. Mp3stat is an opensource tool that scans two media files, MP3 and/or Ogg, and generates a linear graph of bitrates per section of the file. The way that this tool present the statistics is focused on the visual aspect of things and, hence, tends to be on the course side so that the seen is not over whelmed with colours. Several examples of visual representations of the data files are provided on the associated web site (See [1]). To be able to take a closer look, on the frame level, of the bitrates that the decoder encounters we have modified the Mp3Stat code. Since Mp3Stat uses the lib-ogg tool *ogginfo*, we "tapped" into the logical flow of Mp3Stat so that every time a new bitrate is encountered, a text line containing that bitrate is written to an external file. The home page for Mp3Stat is:

`http://safemode.homeip.net/`

For the audio analysis of the Ogg Vorbis and Mp3 we have downloaded a demo version of **SpectraLAB** and **CoolEdit Pro**. Either one of these tools would have been sufficient to analyse the decoded sound signals, the problem was that in the demo version there are restrictions use. As CoolEdit does not allow use beyond the 30 minute mark and limits some functions, SpectraLAB does not allow anything to be written to disk. The combinations of both does all that one could need. The home page for CoolEdit and SpectraLAB respectively are:

`http://www.syntrillium.com/`
`http://www.cetaceanresearch.com/analysis.html`

To transform the PCM data, cdda, into wave file we have used **Creative Studio** while **Creative Mixer** was used to adjust the output levels and edit the appropriate 20-second samples obtained.