12-17-2010

Forensic Reconstruction of Fragmented Variable Bitrate MP3 files

Abhilash Sajja
University of New Orleans

Forensic Reconstruction of Fragmented Variable Bitrate MP3 files

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
In partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

by

Abhilash Sajja

December, 2010

Acknowledgements

First I would like to thank my parents for their support and encouragement. Special thanks to my advisor Dr.Vassil Roussev for his guidance and feedback without which this thesis would not have been possible. I would also like to thank Dr.Golden Richard for his support and guidance throughout my Masters degree. I would like to thank all the people who have helped me in one or the other way during these two years. Finally I am grateful to UNO for providing me with all the necessary equipment for carrying out my research.

Table of Contents

List of Figures

List of Tables

Abstract

File carving is a technique used to recover data from a digital device without the help of file system metadata. The current file carvers use techniques such as using a list of header and footer values and key word searching to retrieve the information specific to a file type. These techniques tend to fail when the files to be recovered are fragmented. Recovering the fragmented files is one of the primary challenges faced by file carving.

In this research we focus on Variable Bit Rate (VBR) MP3 files. MP3 is one of the most widely used file formats for storing audio data. We develop a technique which uses the MP3 file structure information to improve the performance of file carvers in reconstructing fragmented MP3 data. This technique uses a large number of MP3 files and performs statistical analysis on the bitrates of these files.

Keywords: Data, Compression

Chapter 1. Introduction

Forensic science involves the gathering and analysis of evidence to establish facts that can be presented in a legal proceeding. Similarly *digital forensics* is a practice which involves the use of analytical and investigative techniques to identify, collect, examine and preserve the information that has been stored on digital media. Typically, this digital media resides in a personal computer but could also include other types of storage devices such as memory cards and portable media players. The information stored on these devices can include anything from e-mail, to photographs, to confidential documents and hence is crucial from an investigator point of view. Recovering this information can be difficult when the data has been deleted or damaged. Thus it requires the use of specialized tools to recover and reconstruct the data. Before we discuss how data is recovered we need to understand how an operating system stores and manages the data for the user.

1.1 File Systems

A *file system* can be defined as the way in which a disk logically stores, retrieves and deletes the data. A computer or any other digital device requires a storage mechanism that can store the information regarding the files location and supply it when required. The purpose of a file system is to manage the data that is being stored on the hard drive and to keep track of its location on the disk. All this information is referred to as metadata. The most commonly used file systems are FAT, NTFS, ext* family (such as ext2, ext3, ext4), UFS and HFS Plus. These file systems can be called as native file systems which are supported by different operating systems. Microsoft Windows supports FAT and NTFS. Linux uses the ext* family of file systems and Mac OS X uses HFS Plus.

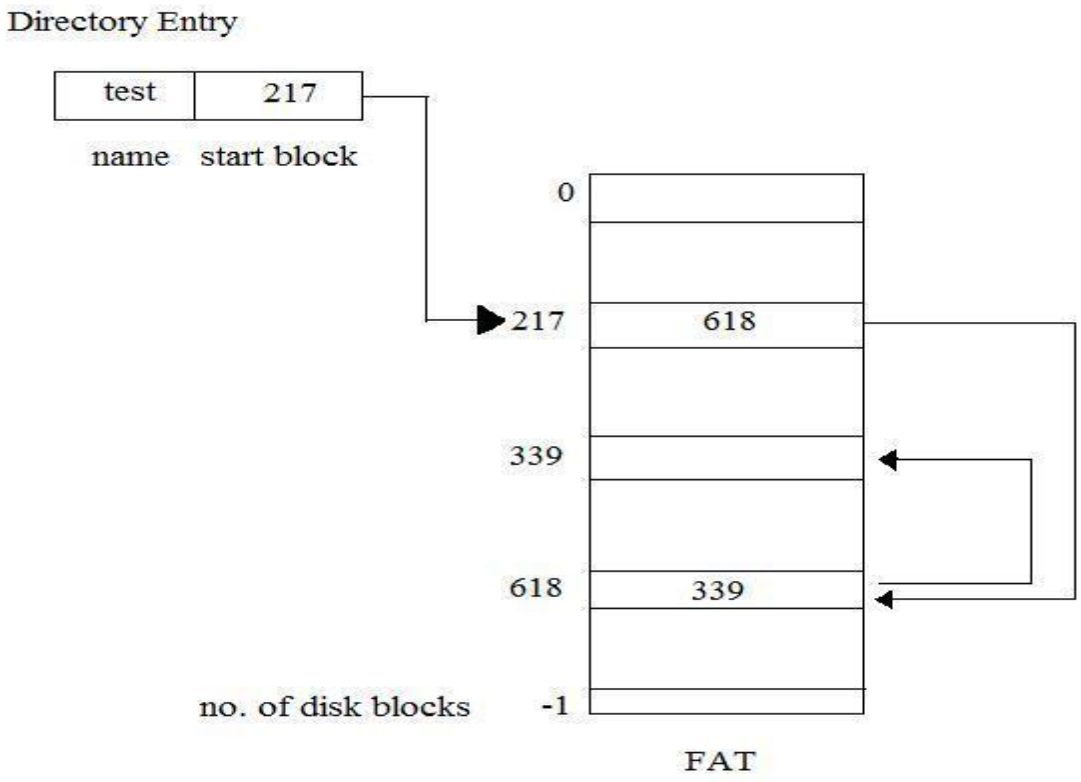

Figure 1 : Example of file system: FAT

Figure 1 illustrates the structure of the FAT file system. A file named test and the block number at which it is stored is recorded in the form of an index in the table. In order to retrieve the contents of the test file the file system looks at the entry in the directory and then goes to the block number specified in the entry. From there it locates the other blocks at which the file is located. When a hard drive is formatted the metadata regarding the files such as file name and its creation time stored in the file table is erased and the corresponding blocks are marked as available for future storage but the locations on the disk at which the file content is stored are not overwritten. This makes the recovery of deleted data possible. In order to retrieve the data the file system metadata has to be reconstructed using special techniques. Now we look at the file system reconstruction techniques and the problems faced by them.

1.2 File System Reconstruction Techniques

File system reconstruction techniques attempt to recover data from a digital device using the information stored in the file system structures such as file tables. These techniques work with the assumption that the file system information is not corrupt or damaged. When a file is deleted file systems do not touch the physical location of the file but the information regarding the location of the file in the file table is removed and the corresponding locations are marked as available for new data to be written. From the file system point of view these locations are unallocated. In some cases even after deletion the file system information may be partially present making the reconstruction possible. By looking at this information we can identify the locations on the disk that are marked as unallocated.

Relying on the file system information for data recovery has a lot of problems. The main problem is that this information cannot be trusted as it can be corrupt. Using corrupt or altered information to recover files would result in random data. Another problem is that this technique does not consider the information from partially deleted files which are important from a forensic point of view. This makes it necessary to have techniques that do not rely on file system information to reconstruct the data.

1.3 File Carving

The term *file carving* is used to indicate the act of recovering a file from unstructured data based on the format of the content. It is a forensic technique that recovers data based on the file's internal structure and content without the help of file system information. It makes use of the knowledge of common file structures, and information contained in files. A wide variety of techniques are used by file carving for data recovery. One of the most commonly used

techniques is using a list of header and footer values of common file structures against a continuous bit stream of data. This technique is called header-footer carving. Many file formats have distinct header and footer values which are assigned when the file format is designed. A header represents the start of a file marker and footer represents the end of a file marker. Once the file carver finds a possible match for the header it looks for the footer value and then carves out all the data in between the header and footer.

Extension	Header	Footer
art	\x4a\x47\x04\x0e	\xcf\xc7\xcb
art	\x4a\x47\x03\x0e	\xd0\xcb\x00\x00
bmp	BM??\x00\x00\x00	
gif	\x47\x49\x46\x38\x39\x61	\x00\x00\x3b
gif	\x47\x49\x46\x38\x37\x61	\x00\x3b
jpg	\xff\xd8\xff\xe0\x00\x10	\xff\xd9
jpg	\xff\xd8\xff\xe1	\xff\xd9
jpg	\xff\xd8	\xff\xd9
png	\x50\x4e\x47?	\xff\xfc\xfd\xfe

Figure 2 : Sample list used by header footer carving [1]

Figure 2 shows a list of header and footer values for specific file formats. Other techniques include key word searching which looks for patterns that help in identifying the file type. Both these techniques work on the assumption that the files contents are stored contiguously i.e. the file is stored in continuous blocks. A number of file carving tools are available which use the above techniques. Some of the most noted file carvers are Foremost [1] which was developed by U.S. Air force and Scalpel [2] developed by Golden Richard III. Scalpel is a complete rewrite of

[1] DFLabs PTK http://ptk.dflabs.com/sw_app_ptk.php

Foremost. It is focused on using less memory and has better performance when compared to Foremost but uses the same overall approach.

Thus file carving tools provide a good solution for data recovery but have one serious limitation. They can only recover files that are stored contiguously i.e. files that are not fragmented. This makes fragmentation one of the primary challenges in file carving.

1.4 Fragmentation

A file is said to be fragmented when it is not stored on consecutive blocks of a disk. A file system normally tends to store a given file sequentially in order to achieve optimum performance but the operations performed on these files such as deleting and reclamation cause fragmentation. Fig 3 shows an example of a fragmented file.

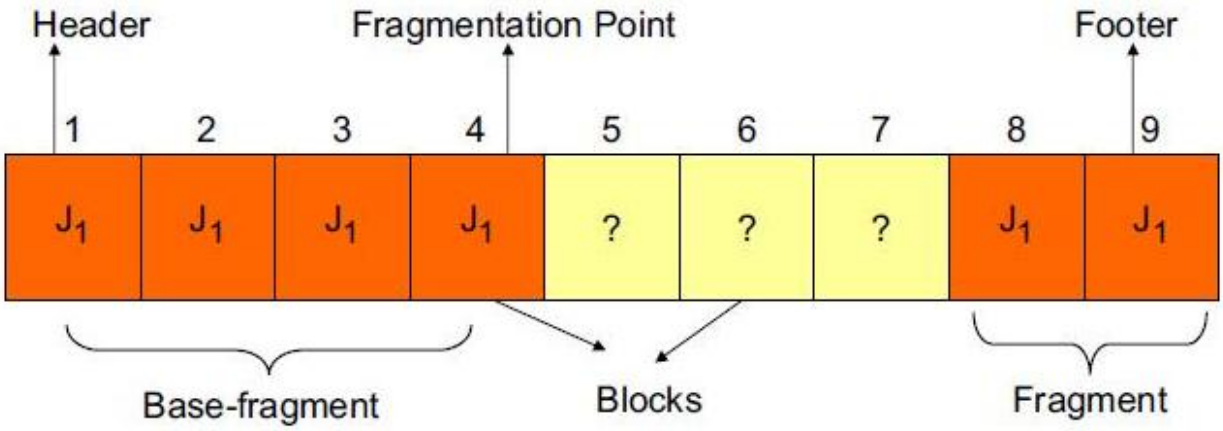

Figure 3 : JPEG file that has been fragmented into two parts [2]

[2] Anandabrata Pal and Nasir Memon , *The evolution of file carving* http://digital-assembly.com/technology/research/pubs/ieee-spm-2009.pdf

Fig 3 shows 9 consecutive blocks on a disk. The JPEG file spans 7 blocks but is not stored contiguously. In other words the file is fragmented. It is fragmented into two parts indicated by base-fragment and fragment. All the blocks that contain the JPEG file are indicated by J_1. The base-fragment starts at block1 and ends at block 4. The second fragment starts at block 8 and ends at block 9. The three blocks in the middle blocks 5, 6 and 7 can either be unallocated or another file can occupy these empty blocks.

If a file is fragmented into two or more pieces the file carver can no longer recover the original file by simply carving out the data between the header and footer because this would result in an incorrect reconstruction of the file. Thus it is very important to either know the correct order in which the fragments are to be joined or the fragments should be present in the correct sequence on consecutive blocks. The latter is rarely a case due to file fragmentation. Hence from a forensic point of view it is important to know the right order in which the fragments are to be joined. This information is with the file system but is no longer available when a disk is formatted. Even if the information can be recovered it can be unreliable as it can be corrupt or damaged. This leaves the forensic investigator with a raw binary stream of data arranged in a number of blocks from which he has to reconstruct the files with no file system information to depend on. Thus fragmentation is a major problem for file carving.

According to a study done by Simson L. Garfinkel [3], fragmentation on a typical disk is less than ten percent however; the fragmentation level of forensically important file types is relatively high. In this study he reveals that the fragmentation rates for JPEG's was 16%, Microsoft Word documents had 17% fragmentation, Audio video interleave(AVI) a file format

used for movies had a 22% fragmentation rate and personal information store(PST) files used by Microsoft Outlook had 58% fragmentation rate. A file is said to be *bifragmented* if it is divide into two parts. His study also reveals that bifragmentation is the most common type of fragmentation, however, files that have been fragmented in to three or more parts are also common. These statistics come from identifying over 350 disks containing FAT, NTFS and UFS file systems. Hence it is very clear that fragmentation is a common phenomenon and hence the recovery of fragmented files is a critical problem in forensics. Now we shall try to understand the common scenarios in which fragmentation occurs.

One of the most common scenarios which lead to fragmentation is appending or editing files. Let us consider two files ‘a’ and ‘b’. If the files are stored contiguously on the disk and later file ‘a’ is appended then there would be no room for the new data to be written as file ‘b’ occupies the immediate block after file ‘a’. This makes the file system find another location for the rest of the data. Ideally the file system should move the entire file ‘a’ to a new location which would avoid fragmentation. But in order to save time and avoid write cycles the file system skips this and stores the new data in a location away from the original file. The other reason for fragmentation is low disk space. When a disk that is already fragmented runs out of space there might be a group of small blocks of free space. If the file to be stored is larger than all the free blocks then in some cases file system can force fragmentation of the file into smaller parts so that it can accommodate the new file. This in turn causes fragmentation. The other cause is wear leveling algorithms. These algorithms are used by solid state drives (SSD) to improve their lifetime. They are used by the controller in the storage device to remap logical block addresses to different physical block addresses. In case the controller is damaged the data extracted will be

inherently fragmented which makes it harder to find the correct sequence of blocks to reconstruct the file.

In order to solve this problem the file carvers can make use of some information about how file system fragments data although this cannot be very useful. Hence fragmentation is not completely avoidable which makes it necessary to develop advanced file carving methods which make use of in depth knowledge of the internal file structure to reconstruct the file from the available raw binary data. This motivates us to find a solution for helping the file carvers in overcoming the problem of fragmentation.

1.5 Related Work

The problem of recovering fragmented files with file structure based carvers became more evident and resulted in development of new techniques. DFRWS [4] is a conference in which a number of challenges are designed for forensic researchers to develop new techniques and algorithms that can improve the tools used for forensics. DFRWS 2006 [5] and DFRWS 2007 [6] focused on file carving. The goal of the challenge was to design and develop file carving algorithms that identify more file formats and reduce the number of false positives.

Simson Garfinkel who was one of the participants developed a technique called *semantic carving* [7] which according to him was an improvement over traditional carving. This technique has two steps. First a trial sequence of blocks is generated and then individually tested using a validator. The sequences of blocks that validate are stored as a complete file and the others are discarded. He also mentions that this technique is computationally expensive. Another technique was developed by Simson Garfinkel known as *bifragment gap carving (BGC)* [8] which works by exhaustively searching all combinations of blocks between an identified header and footer

while excluding different number of blocks until a successful validation is possible. This technique was developed for the recovery of bifragmented files. It is said to perform satisfactorily when the two fragments are close to each other; however, it has many limitations for the more general case.

Many of the techniques developed either have limitations or are computationally exhaustive. All the above mentioned techniques are “generic” which rely on a verifier which validates a sequence after exhaustively trying out all the possible permutations. This makes them computationally expensive and hence there is a need for developing specialized techniques which are less complicated and that can work on specific file formats.

One such technique called *smart carving* [9] was developed by Anandabrata Pal and Nasir Memon. It focuses on the recovery of fragmented JPEG images. It makes use of information regarding how file system fragments data and the internal structure of a JPEG image. This technique is not limited to recovering files containing two fragments. It involves three phases preprocessing, collation and reassembly. The first phase involves decryption and decompression of the contents in the disk if required and the second phase involves the classification of data based on the files contents. The third and the final phase involves finding the fragmentation point of each unrecovered file and then finding the next fragment’s starting point. This process is repeated until a file is built or determined to be unrecoverable. This technique requires a lot of processing and involves complicated algorithms but tries to solve the problem of fragmentation in the case of JPEG.

We have seen that a good amount of work has been done for improving the file carving techniques for JPEG images but no work on a file format such as MP3. Hence we decided to work on MP3.

1.6 Our Contribution

In this research we have chosen one of the most commonly used file formats for storage of audio data, MPEG -1 Audio Layer 3 more commonly known as MP3 and have successfully found a solution that can be used to improve the reconstruction of fragmented variable bit rate (VBR) MP3 files. A challenge presented in DFRWS 2007 [10] led to the development of *mp3scalpel* a file type specific carver which works on constant bitrate (CBR) MP3 files. Our work can be looked at as a development of the mp3scalpel which will allow it to work on another mode of MP3 knows as variable bitrate or VBR. This mode is more commonly used when compared to CBR since it presents higher quality audio and does not show a significant increase in size. This is achieved by using different bitrates depending on the audio data rather than choosing the same bitrate value.

In this thesis we study the internal structure of a VBR MP3 file and describe a technique which uses the bitrates as a parameter that can be used to identify which fragments belong to a particular file and thus reconstruct the fragmented MP3 files. In this technique we use a large data set of MP3 files of different genres and calculate the bitrates using the file structure information. We then calculate the conditional probabilities from the bit rate values. With the help of this information we try to reconstruct the MP3 files from their individual fragments. The MP3 file format will be discussed in detail in Chapter 2 and our technique will be discussed in detail in Chapter 3.

Chapter 2. MP3 File Format

2.1 Introduction

MPEG-1 Audio Layer 3 also referred to as MP3, is a digital audio encoding technique that is used to represent audio data. It is a compressed file format that is widely used for encoding audio data because of its ability to reduce the file size significantly when compared to other audio formats. For example the digital information on a standard audio CD would require about 10 MB per minute of music. If the same data is ripped from a CD to a computer and encoded using MP3 format the same minute of music would require about 1 MB which is one tenth of the original size. The algorithm aims to preserve the sound quality while minimizing the space required for storing the signal. The MP3 codec achieves this by using lossy compression. This technique removes any sounds in the original signal with frequencies that are outside the audible scale and thus reduces the amount of data required to represent that signal. This algorithm chooses which parts of the file are to be removed based upon a method known as Perceptual coding [11]. Reducing the file size has a number of benefits. It allows the users to store more audio information in less space and also allows them to share these files easily in less time. This makes MP3 one of the most widely used file formats for audio storage.

2.2 The MP3 File Format

An MP3 file is composed of a sequence of data blocks called frames. Unlike other file formats MP3 format does not have a file header. Every frame has its own individual header which is constituted by the first four bytes or 32 bits. The frames in an MP3 file are self contained and hence independent of each other. The first twelve bits in the header are all set to one. These twelve bits are together called as ‘frame sync’. Followed by the frame sync is the information

about version, bit rate, sample rate and padding. Based on these values the length of the frame can be calculated and the appearance of the next frame header can be predicted. This data is followed by the audio data and then the next frame and so on. The following figure shows the MP3 file structure.

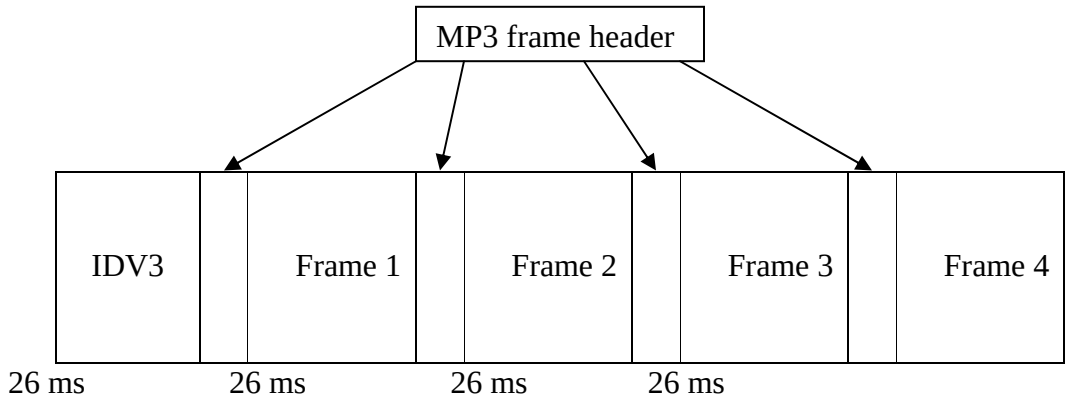

Figure 4 : MP3 File Structure

Figure 4 illustrates the structure of an MP3 file. It starts with the IDV3 tag which holds the metadata information of the file such as title, artist, album, track number and other information about the file's contents. This field allows the information regarding the file to be stored inside in the file itself. The standard tag formats are ID3v1 and ID3v2 and APEv2. ID3v1 tags can appear either in the beginning of an MP3 file or at the end. This is different form ID3v2 which always appears at the beginning of the bit stream. The reason for this is that the MP3 player has to be able to display the information throughout the duration of the track not at the end which has no use. This is followed by a sequence of frames which are independent of each other. The size of each frame is constant at 26 ms and the number of samples per frame is also constant at 1152. Every frame consists of a frame header and the encoded audio data. The following figure illustrates the structure of an MP3 frame.

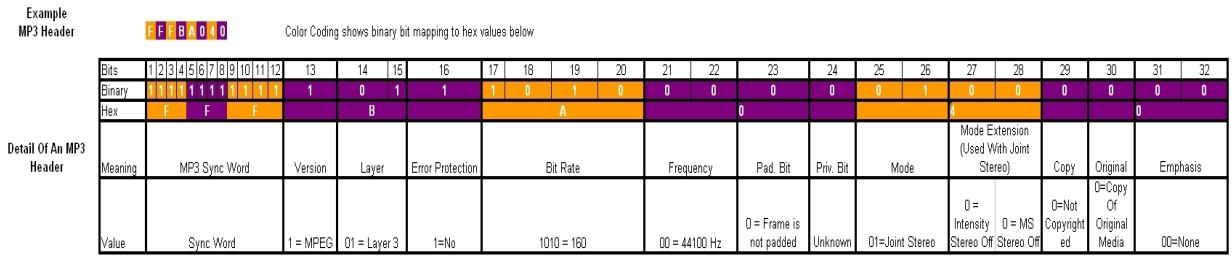

Figure 5 : MP3 Frame Format [3]

Fig 5 shows the format of an MP3 frame in detail. It consists of various fields which hold the information regarding the MP3 file. Now we shall discuss each of the fields in detail.

- *Frame Sync*: This field consists of 12 bits. All the bits are set to 1 and this field is used to identify the beginning of a valid MP3 frame. One of the reasons for this is that MP3 was designed in such a way that it would be suitable for broadcasting. This helps the receivers to lock onto the signal at any point in the stream.
- *Version*: This field appears directly after the frame sync. It consists of two bits which are used to indicate which version is being used to encode the audio data. The different values of the bits indicate different versions. Table 1 shows the different values represented by this field.

Table 1 - Representation for Version

Bits	Version
00	MPEG Version 2.5
01	Reserved
10	MPEG Version 2
11	MPEG Version 1

[3] MP3 file structure : http://en.wikipedia.org/wiki/MP3

Based upon the above values we can know which MPEG version is being used to encode the data. Since we are dealing with MP3 we would be looking for this field to represent 11.

- *Layer*: This field is used to indicate the layer that is being used in the encoding process. It has two bits to indicate the layer. Depending upon the layer and the version fields the set of bit rate values used to encode the data change. Since we are dealing with MP3 files we would be looking for this field to represent 01.

Table 2 : Representation for Layer

Bits	Layer
00	Reserved
01	Layer 3
10	Layer 2
11	Layer 1

- *Protection*: This field specifies whether the data is protected or not. It is represented by one bit. If the bit is set to 1 then the data is not protected and if it is set to 0 the data is protected which means that a 16 bit CRC will follow the header. CRC is used to check the integrity of the file. If the CRC does not match then the frame is considered to be corrupt. This field is not normally used. Hence we can expect the bitrate field to follow the layer field.
- *Bitrate*: This field specifies the bit rate at which the frame is encoded. Bitrate is defined as the number of bits processed per unit of time. MPEG-1 Audio Layer 3 standard specifies a number of bit rates at which the audio data can be processed. They are 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256 and 320 Kbit/s. This field

consists of four bits that are used to indicate the bitrate value. The following table shows the bit representations and their corresponding values.

Table 3 : Representation for Bitrates

Bits	Bitrate (Kbps)
0000	Free
0001	32
0010	40
0011	48
0100	56
0101	64
0110	80
0111	96
1000	112
1001	128
1010	160
1011	192
1100	224
1101	256
1110	320
1111	Bad

In the above table 0000 represents “Free” format which means that there is no restriction. 1111 represents “Bad” which indicates that the value is not allowed.

There are three modes in which an MP3 file can be encoded. They are:

- *Constant Bit rate (CBR)*: With CBR encoding, the same number of bits are added to each frame of the audio data regardless of the input signal. This method is used if the file size is the primary concern.
- *Average Bit rate (ABR)*: This method allows the user to choose an average bitrate value and the encoder then tries to reach a target value by allowing the bitrate to vary from frame to frame.

- *Variable Bit rate (VBR)*: This method allows the encoder to maintain high quality throughout the MP3 file. The file size is not predictable in this method as it depends on the decisions made by the encoder. This method is recommended because it uses the bitrate values depending on the audio data. Simple parts of songs that include silence do not need the same amount of bits used for representing a complex signal.

This research is focused only on VBR MP3. There are a number of encoders available which support this method of encoding. One such encoder is Lame encoder which also gives the user an option of choosing a quality for VBR encoding. The encoder then allows the bitrate to vary based on the content for more consistent quality. It would use a higher bitrate value for signal that requires more data to represent and lower bitrate value for a signal that requires less data.

- *Sampling rate frequency*: This field consists of two bits that are used to indicate the sampling frequency. The most widely used sampling rate is 44.1 KHz or 44100 Hz. Sampling rate is the rate at which every second of audio data is sampled. The table below shows the different sampling rates and their representations.

Table 4 : Representation for Sampling Rate

Bits	Sampling Frequency
00	44100
01	48000
10	32000
11	Reserved

- *Padding*: This field consists of one bit. Padding is used to make sure that each frame satisfies the bitrate requirements exactly. It is used in case a frame is not filled up with the right amount of data. If the bit is set to 1 it indicates that the frame is padded with one extra slot.
- *Private*: This bit is reserved for specific needs of an application i.e. if an application has to trigger an event.
- *Channel Mode*: This field consists of two bits. It is used to indicate the type of channel being supported by the frame i.e is the status of the frame mono or stereo. The different possible channels are Stereo, Joint stereo, Single channel, Dual channel.

Table 5 : Representation for Channel Mode

Bits	Channel Mode
00	Stereo
01	Joint Stereo
10	Dual Channel
11	Single Channel

- *Mode Extension*: This field is used only in the case of Joint stereo. It is used to join information that are of no use for stereo effect, thus reducing the required resources. It consists of two bits that indicate which type of joint stereo is used.

Table 6 : Representation for Mode Extension

Bits	Intensity stereo	MS stereo
00	Off	Off
01	On	Off
10	Off	On
11	On	On

- *Copyright*: This field consists of one bit. It is used to indicate if the file is copyrighted or not. If this bit is set, it is officially illegal to make a copy of the file.
- *Original*: This field consists of one bit. It is used to indicate whether a file is original or a copy of an original file. If the bit is set then the file is original.
- *Emphasis*: This field consists of two bits. It is used as a flag bit which is used when the file is created during the original recording. This field is no longer used and hence is not very important. The following table shows the representation.

Table 7 : Representation for Emphasis

Bits	Emphasis
00	None
01	50/15 ms
10	Reserved
11	CCIT J.17

When an MP3 file has to be decoded the decoder has to check for all this information and then decode the audio data and play the file. This is just for one single frame in the entire file. A single MP3 file can contain thousands of frames.

The length of a frame can be calculated from the bitrate, sampling rate and padding fields. The formula for computing the frame length is as follows:

Frame Length (bytes) = 144 * Bit rate (kbps) / Sample rate + Padding

By calculating the frame length we can reliably estimate the location at which the next frame would start.

This can help us in locating the next MP3 chunk (block containing MP3 data) and thus help us decide if the mp3 file is fragmented. This can be done in the following way:

1. Find the sync bits.
2. Then check for a valid MP3 header. This can be done by checking for all the above fields to be present.
3. Now we can calculate the length of the current frame with the above formula.
4. Since we have the frame length we know where to expect the next frame header.
5. Now we can go to that location and look for the next frame header. If we do find a valid MP3 frame then we have found a valid MP3 frame boundary. Then we repeat the process again until we reach the end of the file.

Since we now know the MP3 file format in detail we can look at the technique we have developed to tackle the problem caused by fragmentation in reconstructing VBR MP3 files.

Chapter 3. VBR MP3 Recovery Technique

3.1 Rationale

The purpose of this research is to develop a technique for improving the performance of file carvers dealing with fragmented variable bit rate MP3 files. In the previous Chapter we have looked at the structure of an MP3 file in detail. By looking at the various fields in an MP3 frame we observe that all the fields except the bitrate hold almost the same information throughout the file and also we see that in VBR files bitrate information varies depending on the audio content in the frame. This indicates that the VBR MP3 frames are correleated and bitrates can be used to establish the way in which they are related. In an MP3 file that has been encoded in VBR (variable bit rate) mode the bit rate value changes from frame to frame. Hence we focus on this parameter in the MP3 format that could help us in choosing the right fragment from a number of fragments to reconstruct the original file. This project is divided into two phases. In the first phase we analyze the bitrate fields of a number of MP3 files and perform statistical analysis on them. After we obtain the results we move on to the second phase. The second phase involves testing the results obtained in the first phase.

In order to test our idea we need a number of MP3 files and hence an encoder which can be used to generate these MP3 files from uncompressed audio. For this purpose we have chosen the LAME MP3 encoder [12]. It is considered to be the best MP3 encoder at mid–high bit rates and for VBR mode. This encoder will be used to generate three different qualities of VBR MP3 files from the source files.

The three different qualities that the encoder provides are as follows:

- *High or v0*: This option would generate bigger files but maintains very good quality i.e higher bitrate values can be expected.
- *Medium or v4*: As the name suggests the files would be of medium quality and medium size.
- *Low or v9*: This level of quality ensures less file size but quality would be low and hence lower bitrate values can be expected to be used.

In our project we use 276 different source files from three different genres. Each of these files can be used to generate three MP3 files of different qualities. Hence from the 276 files we have 276*3 = 828 MP3 files. The way this is done is that all the source files are ripped from their original disks and they are encoded using the LAME MP3 encoder into three different qualities namely v0, v4 and v9. Where v0 represents high quality, v4 represents medium quality and v9 the lowest quality. This gives us a data set of 828 MP3 files.

3.2 Statistical analysis of VBR MP3 files

Since we are interested in the bitrates of these files we use the file structure details to locate and read the bitrate field in each frame and calculate the bitrate values. After obtaining the bitrate values for all the files we try to find a pattern in which the bitrates are distributed. The following figure illustrates the results of the analysis. We separate the results based on the genres and the qualities of the files.

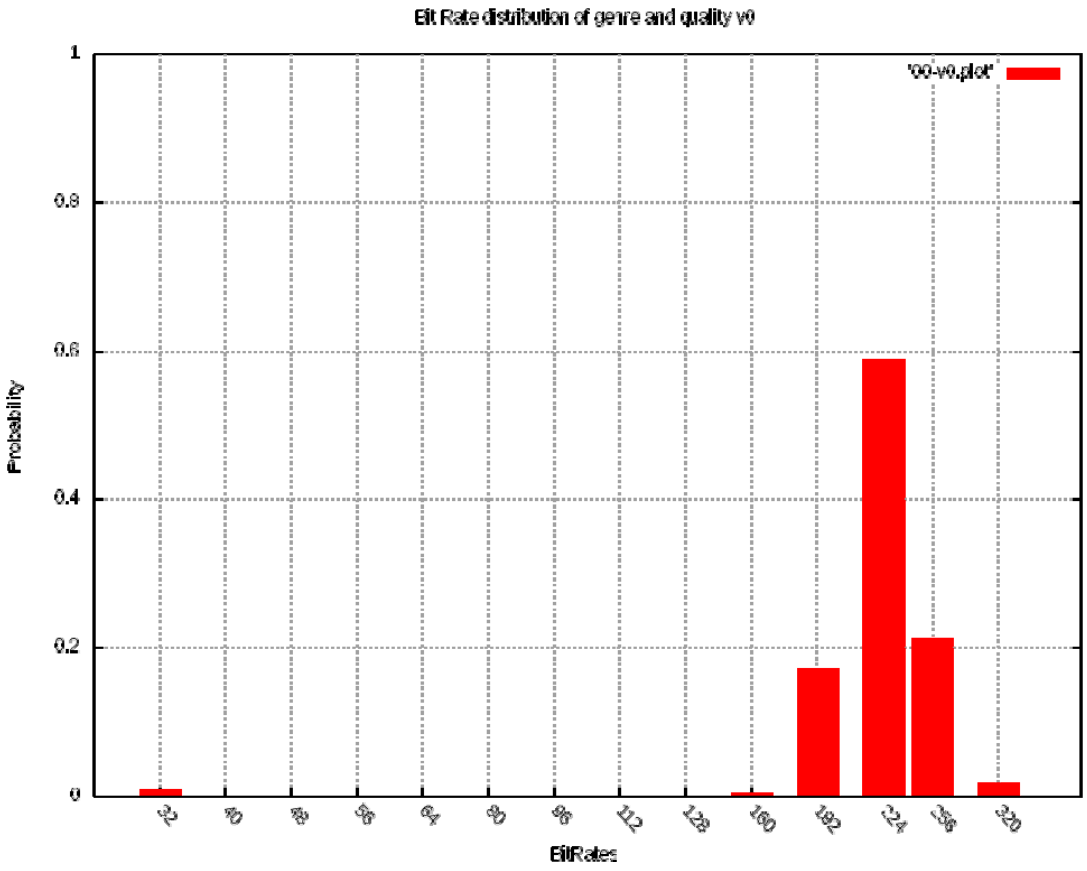

Figure 6 : Bitrate distribution of classical genre and high quality

In the above figure we have the empirical distribution of the bitrates of a high quality MP3 file. The x –axis has the possible bitrates and the y-axis has the probability scale. We see that bitrate 224 has the highest probability of occurrence. The other bitrates have lower probability and some of them are very close to zero which is not noticeable in the figure. We perform similar analysis for all the genres and qualities.

Since we now have the bitrates information we can perform statistical analysis on them to find the probability with which the bitrates of different frames follow one another. For this we use the concept of conditional probability. We choose a bitrate value among the possible 14 values and calculate the probability of occurrence of a particular bitrate after each of the possible

14 bitrates. Table 8 shows a sample of the file that contains the bitrates and their corresponding probabilities.

Table 8 : Bitrates having conditional probability of more than two percent

B_1	B_2	P_C
032	032	0.98425
040	032	0.42452
048	040	0.20588
048	048	0.20588
056	080	0.20537
064	056	0.21111
064	080	0.33333
080	080	0.31746
096	080	0.31932
096	096	0.31092
112	112	0.42168
112	128	0.20481
128	128	0.37755
128	160	0.34693
160	192	0.54911
160	224	0.24263
192	192	0.21125
192	224	0.68955
224	192	0.20062
224	224	0.57940
224	256	0.20801
256	224	0.56739
256	256	0.34326
320	192	0.32998
320	224	0.48990

The table above shows a sample of the results we have obtained from the statistical analysis done on high quality classical genre MP3 files. The table shows all the bitrates that have conditional probabilities more than 2 %. The values in the columns labeled 'B_1' are the bitrates that appear first and the columns labeled 'B_2' are the bitrates that appear in the next frame. The column labeled P_C has the conditional probabilities. The same process is applied for all the genres and all

the three qualities available. Now we move on to the next phase of the project where these results will be tested.

3.3 Evaluation

In order to test the results we have obtained we need to create a scenario which has a number of fragmented MP3 files which have to be recovered. In order to do this we take an MP3 file and logically split it into chunks of size 4096 Bytes or 4KB (will be referred to as 4k from now on). We have chosen this number because it is the most commonly used allocation size by a file system. If we have an MP3 file of size 4MB and we divide it into 4k sized blocks we would have 1024 blocks. Each of these blocks can be imagined to be an MP3 fragment. In order to do this we logically split a file into 4k sized blocks. After splitting the file into the blocks we obtain the following parameters for each of the blocks.

File name: This is the name of the file. It would be used to know to which file the block originally belongs.

Version number: This is used to identify the quality of the file to which the block belongs whether it is v0 (high), v4 (medium) or v9 (low).

Block number: This indicates the number of the block in the MP3 file. It is used torecognize the block.

Leading offset: This is the size of the first complete frame or a part of the first frame that has been split at the block boundary.

Trailing offset: This is the size of the last frame or a part of the last frame that has been split at the block boundary.

Expected offset: This is the size of the remaining part of the frame that has been split at the block boundary which is required to complete the frame.

Expected bitrate: This is the bitrate that has to be present in the block that matches the expected offset. If the expected bitrate and the expected offset do not match then the block is not the right match.

This procedure is repeated for all the MP3 files and the data is stored onto a text file. All these files would then be joined into one single file and that file would be uploaded into a database. The figure below shows a sample of the file that contains the above information for a single MP3 file.

Table 9 : Sample of the block data for an MP3 file

Block Number	Leading Offset	Expected Bitrate	Expected Offset	Next Bitrate
0000	0000	0032	0065	0032
0001	0065	0032	0025	0032
0002	0025	0192	0388	0224
0003	0388	0192	0468	0224
0004	0468	0224	0653	0192
0005	0653	0224	0107	0224
0006	0107	0224	0397	0224

The above table shows a sample of the data that contains the information for all the blocks that belong to a file named 00-001-v0.MP3 Here 00 is used to identify the genre and 001 is the number of the file. The columns show sample of data such as block number, leading offset, expected bitrate, expected offset and next bitrate.

3.4 Implementation

Since we have a lot of data in our project we use a database tool that would allow us to easily manage and retrieve data. Since we also need a lot of data manipulation we write a script which performs an operation a number of times repeatedly. But this script has to be able to perform operations on a database. Hence we use Pysqlite which is a Python binding for the SQLite light weight database engine. Pysqlite allows us to write a script which can query the tables that are stored on the database and perform operations based on the data. Hence we first export the data that we have generated into the database engine which stores it in the form of tables. Then we write a script in Python which would perform queries on the tables stored in the database and give us the results.

First we create a table in the database which stores the entire block data that we have generated using the block generator. Let's call this table as "Main Table". Then we create 9 more tables which store the probability data that we generated in the first phase of the project. There are three qualities of MP3 files that we have generated and there are three genres that we are using. Hence we store them individually in 9 different tables.

Now we write a script in python which connects to the database and performs the required operations. It chooses a particular file and selects a random block number from the file. It then stores the requirements of the block like the expected bitrate and expected offset. It does this for all the files and stores them in a list. After doing this it scans for all the available blocks in the list which satisfy the requirements of the particular block. Once it finds a match for the offset requirement it stores the details of the match in a list. It then scans for the bitrate value of the block that matched the offset requirement, we call this available bitrate. If 5 matches are found for a particular offset requirement then the probability of being able to make the right

choice which satisfies both the offset and birate is 20%. Now if we read the bitrate values of the blocks that matched with the offset values and query the table which has the conditional probabilities for all the possibilities then we have a better chance of making the right decision. For example if we are looking for an offset requirement of 417 and we find 10 matches for it we know that one of the matches is the perfect match for the block but we do not have any means of knowing that a particular match is the perfect match.

Hence in order to improve our chance of making the right decision we take the help of our conditional probability results. By reading the bitrates of all the possible matches we look into the probability values and choose the one that has the highest probability. If we get the same value for 2 matches we have at least significantly improved our chances of finding the right match from 10 to 50 %.

In this project we have performed a test in which we use the probability results and choose one of the matches that has the highest probability. We then check and see if we have actually chosen the exact match i.e. if the block we have chosen is the block of the original file. Our results showed that we had significantly improved the chances of finding the original block when compared to the scenario in which there is no probability table to look at.

3.6 Results

We will now have a look at the results we obtained from the tests. The following tables show the results with and without using the conditional probability statistics. The following symbols are used in the results table.

- N_m: This is used to indicate the number of matches i.e. the number of blocks that satisfy the offset requirement.

- Hits : Number of times we found the match to be the perfect match(belongs to the original file)
- Misses: Number of times we failed to find the match to be the perfect match.
- Samples: Total number of hits and misses.
- P_b : Chance of finding the right match in % using random choice i.e. before using the conditional probability results.
- P_a : Chance of finding the right match in % after using the conditional probability results.
- $P_{i:}$ Percentage increase in the results.

The following tables show the results obtained for v0 or high quality MP3 files, v4 or medium quality MP3 files and v9 or low quality MP3 files.

Table 10 : Results obtained for high quality MP3 files

N_m	Hits	Misses	Samples	P_b (%)	P_a (%)	P_i (%)
2	939	319	1258	50	74.64	49.20
3	104	082	0186	33	55.91	69.42
4	018	026	0044	25	40.9	63.00

Table 11 : Results obtained for medium quality MP3 files

N_m	Hits	Misses	Samples	P_b (%)	P_a (%)	P_i (%)
2	816	778	1594	50	51.2	2.40
3	133	263	0396	33	33.6	1.80
4	0195	057	0076	25	25	0
5	001	004	0005	20	20	0
6	001	005	0006	16	16.6	3.75

Table 12 : Results obtained for low quality MP3 files

N_m	Hits	Misses	Samples	P_b (%)	P_a (%)	P_i (%)
2	424	176	600	50	70.66	41.32
3	037	029	066	33	56.06	69.87
4	004	004	008	25	50.00	100.00

Table 13 : Results obtained for all qualities of MP3 files combined

N_m	Hits	Misses	Samples	P_b (%)	P_a (%)	P_i (%)
2	2774	864	3638	50	76.25	52.50
3	0900	573	1473	33	61.09	85.12
4	0181	155	0336	25	53.8	115.20
5	0040	050	0090	20	44.44	112.00
6	0008	010	0018	16	44.44	177.50

By looking at the results we can infer that the success rate has increased significantly after using the probability statistics. Thus our idea of using the bitrates of a variable bitrate MP3 as a parameter for improving the ability to reconstruct a fragmented MP3 file has worked and can thus be used for improving the performance of file carving for MP3 files.

Our technique also has many advantages when compared to the existing techniques and they are as follows:

- It is a simple technique which involves statistical analysis.
- It does not involve complex signal processing techniques which consume a lot of time and have severe impact on the performance of the file carver.
- Our technique is simple and involves looking up a table of probabilities and choosing which fragment to consider for reconstructing the file.
- Can be easily added to existing file carvers and more the number of samples used for analysis the better this technique becomes.

Chapter 4. Conclusions

File carving is a data recovery technique that relies on file structure information for recovering file specific data. This technique works when the data being recovered is stored sequentially i.e not fragmented. Hence fragmentation is the primary challenge faced by file carving. Statistics show that fragmentation is a common phenomenon and is even severe in the case of forensically important files. This makes it necessary to develop advanced techniques that can work on specific file formats by looking into the structure of the file in detail and developing methods which can improve the performance of file carving.

In our research we focus on Variable Bit Rate (VBR) MP3 which is one of the most widely used file formats for storing audio data. We develop a technique which makes use of in depth knowledge of MP3 file format for improving the performance of file carvers when dealing with fragmented VBR MP3 files. We target the correlation between MP3 frames and find a pattern in which the bitrates appear in the frames and make use of this knowledge to reconstruct fragmented MP3 files.

We have tested our technique on a number of MP3 files and the results we have obtained show that we can significantly improve the performance of file carving for VBR MP3 files. This can be done by using our results as a benchmark for choosing fragments to reconstruct VBR MP3 files.

Our technique has significantly improved the chances of finding the right fragment to reconstruct a file from a number of available fragments belonging to various files.

In order to have a large data set we used MP3's of all qualities and genres as one single dataset and have achieved an average percentage increase in the range of 52.50 – 177.50 %. The following are the results we have obtained for different qualities of VBR MP3 files.

1. For high quality files we have achieved a percentage increase in the range of 49.20 – 69.42 %.

2. For medium quality we have achieved an in line increase in the range 01.80 – 03.75 %. In this case we have observed that the bitrates do not change as much as they do for the other qualities. This is the reason why we have not been successful in achieving higher increases.

3. For low quality we have achieved a percentage increase in the range 41.32 – 100.00 %. In this quality we have observed that the encoder switches to bitrates that do not belong to MPEG version 1 and layer 3. This is because the encoder is focused on reducing the file size as much as possible and thus inclining to use the least possible bitrates.

Thus overall we have achieved significant increase in the probability of finding the correct fragment from a list of fragments thereby improving the performance of file carving in the case of fragmented VBR MP3 files.

References

[1] Foremost : Originally developed by the United States Air Force Office of Special Investigations and The Center for Information Systems Security Studies and Research. http://foremost.sourceforge.net/

[2] Scalpel: Developed by Golden Richard III. http://www.digitalforensicssolutions.com/Scalpel/

[3] Simson L. Garfinkel *Carving Contiguous and Fragmented Files with Fast Object Validation,* in Proceedings of the 2007 digital forensics research workshop, DFRWS, Pittsburgh, PA, August 2007 available at the following link : http://www.dfrws.org/2007/proceedings/p2-garfinkel.pdf

[4] DFRWS: http://www.dfrws.org/

[5] DFRWS 2006: http://www.dfrws.org/2006/index.shtml

[6] DFRWS 2007: http://www.dfrws.org/2007/index.shtml

[7] Simson L. Garfinkel *Carving contiguous and fragmented files with fast object validation* http://simson.net/clips/academic/2007.DFRWS.pdf

[8] Simson L. Garfinkel *Carving contiguous and fragmented files with fast object validation* http://simson.net/clips/academic/2007.DFRWS.pdf

[9] Anandabrata Pal, Nasir Memon *Detecting file fragmentation point using sequential hypothesis testing* : http://www.dfrws.org/2008/proceedings/p2-pal.pdf

[10] DFRWS 2007 challenge was based on data carving: http://www.dfrws.org/2007/challenge/index.shtml

[11] Jayant, Nikil; Johnston, James; Safranek, Robert (October 1993). *Signal Compression Based on Models of Human Perception*:

http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00241504

[12] Lame MP3 encoder: http://lame.sourceforge.net/

Vita

Abhilash Sajja was born in India in 1986.