2013-06

Optimal sector sampling for drive triage

Taguchi, James K.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

OPTIMAL SECTOR SAMPLING FOR DRIVE TRIAGE

by

James K. Taguchi

June 2013

Thesis Advisor:	Joel D. Young
Thesis Co-Advisor:	Simson L. Garfinkel

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
06-21-2013	Master's Thesis	2102-06-01—2104-10-31

4. TITLE AND SUBTITLE

Optimal Sector Sampling for Drive Triage

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

James K. Taguchi

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number N/A.

14. ABSTRACT

With digital storage becoming cheaper, bigger, and more prevalent, finding evidence from the hard drives collected for a case is too difficult and time consuming. Simply reading an entire drive takes hours and it takes even longer to analyze the drive for deleted files and data fragments. Investigations frequently involve multiple drives, and this traditional method of reading entire drives for analysis simply cannot keep up in modern cases. Furthermore, investigators often search drives only for known files, which we call target data, that could help identify a drive holding evidence such as child pornography or malware. Triage is needed to sift through drives to quickly identify drives containing target data. One way is by randomly sampling drive data to find known files or to give a confidence that less than some small amount is present. We determine the optimal sampling strategy bypassing the file system to find even deleted files and fragments in minimum time with maximum confidence. With 15 minutes of sampling we can give a 90% confidence that less than 10MiB of target data is present on a 500GB hard disk drive. By using statistical sampling in combination with sector hashing, our software forms an efficient triage tool for digital forensics.

15. SUBJECT TERMS

Sector Hashing, Forensic Triage, Digital Forensics, Random Sampling, Disc Drives, File Hashing, Target File Detection, File System, Digital Forensic Investigators

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	63	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

OPTIMAL SECTOR SAMPLING FOR DRIVE TRIAGE

James K. Taguchi
Civilian, Department of the Navy
B.S., Computer Engineering, Santa Clara University, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2013

Author: James K. Taguchi

Approved by: Joel D. Young
Thesis Advisor

Simson L. Garfinkel
Thesis Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

With digital storage becoming cheaper, bigger, and more prevalent, finding evidence from the hard drives collected for a case is too difficult and time consuming. Simply reading an entire drive takes hours and it takes even longer to analyze the drive for deleted files and data fragments. Investigations frequently involve multiple drives, and this traditional method of reading entire drives for analysis simply cannot keep up in modern cases. Furthermore, investigators often search drives only for known files, which we call target data, that could help identify a drive holding evidence such as child pornography or malware. Triage is needed to sift through drives to quickly identify drives containing target data. One way is by randomly sampling drive data to find known files or to give a confidence that less than some small amount is present. We determine the optimal sampling strategy bypassing the file system to find even deleted files and fragments in minimum time with maximum confidence. With 15 minutes of sampling we can give a 90% confidence that less than 10MiB of target data is present on a 500GB hard disk drive. By using statistical sampling in combination with sector hashing, our software forms an efficient triage tool for digital forensics.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

I would like to thank my thesis advisor, Joel Young, who has worked with me for a year to complete this thesis. Without you, I never would have been able to complete my Master's Degree. You were always there when I was stuck and needed help, and you would always sit down with me for as long as it took to resolve my issues. Your presence as a mentor and guide was invaluable through this adventure. Thank you for everything you have done for me.

I would like to thank my co-advisor, Simson Garfinkel, who gave me the background necessary to complete this thesis and provided advice when I did not know how to proceed. You may not have thought that your words had any significance, but without them, I would not have been able to figure out how to put my work together in this thesis.

I would like to thank Cynthia Irvine for the educational opportunity at Naval Postgraduate School through the Scholarship for Service program.

I would like to thank my loving mother, Sachiko, for always caring about me and being there for me. I would also like to thank my late father, Tomohiko, who has always been supportive of my work and who I know is proud of me.

I would like to thank my girlfriend, Memori, who has been with me for years, every step of the way, and has always provided me with the small things I need in life when I need it most, whether it's love, humor, or a best friend.

Partial support for this work was provided by the National Science Foundation's CyberCorps®: Scholarship for Service (SFS) program under Award No. 0912048. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Modern technology has enabled criminals and terrorist organizations around the world to reach new levels of sophistication, and many crimes in the physical world have some ties with the digital world. Even for non-cyber crimes, we live in a society where the typical person uses computers, phones, and other digital devices in daily activities. Furthermore, digital evidence has an increasingly important role in court.

For example, in 2011, a massive riot in Vancouver unleashed a police investigation that brought in thousands of tips from the public including images, videos, and links to social media identities [1]. Sergeant Dale Weidman, an investigator on the case, commented that "the sheer volume and speed of the information is overwhelming." In both cyber and non-cyber crimes, investigators and digital forensic examiners have the difficult task of finding key data that can be used as evidence in criminal cases. There is no shortage in demand for digital investigations, yet the time it takes to conduct full examinations is steadily increasing. This is largely due to the huge advances in storage capacity over the past few decades. This combination of demand and technology has caused a backlog of digital forensic examinations which has been frequently documented by major organizations such as Dell [2] and the FBI's Regional Computer Forensics Laboratories [3].

A primary reason for this backlog is the sheer volume of data that must be consumed and analyzed. The FBI Regional Computer Forensics Laboratories reported that in 2011, they conducted 7,629 examinations and processed 4,263 terabytes of data—nearly a dozen terabytes of data per day [3]. The Defense Computer Forensics Laboratory conducted 1,406 examinations and processed 835 terabytes of data in 2012 [4]. Hard drive technology has improved significantly since first introduced in 1956, both in terms of performance and capacity. The growth in capacity has severely outpaced the gains in performance where the physical limits of spinning disks have reached the point of diminishing returns making the cost of better performance not worth the performance gained. Although much faster solid state drives are now available for consumers, their cost is significantly higher than traditional hard disk drives whose cost per storage unit is unrivaled guaranteeing their presence on the market for years to come. Hard disk drives take a long time to process as it takes two to three hours to simply read all of the contents of a consumer grade terabyte hard drive. In addition, popular forensic tools such as EnCase

take additional processing time before the examiner can begin analysis of the drive. This poses a challenge to forensic examiners who must determine how to spend their time. Consequently, triage is necessary in digital forensics where the limiting factor is not a lack of ability to examine hard drives but time. With limited time, the examiner must pick and choose which hard drives in a case to look at first. There is currently no easy or automatic way to find hard drives that are of interest without investing a considerable amount of time looking at contents.

These difficulties on top of the demand for the timely identification of forensic evidence at the scene of the crime or within a short time period make the current approach to digital forensics too slow. Richard and Roussev noted in 2006 that cases are becoming increasingly complex due to "terabytes of storage becoming more common and cases routinely involving more than a single computer" and that "significant levels of innovation" will be required for the next generation of digital forensic tools [5]. What we need is a new tool or technique to perform cyber forensic triage to quickly identifying high-value targets to counter the proliferation of digital devices [6]. A common source of evidence, the hard drive, may be the first thing that an investigator will go through. To perform triage, the investigator may take a quick look to see if any interesting files are on the drive. The conventional hard drive forensic tool reads the entire content of a hard drive and attempts to locate data that would be useful to the investigator. These tools will often, though not always, use the file system to browse the directory listing and identify files. This allows all non-deleted files to be found and obtain additional metadata about these files such as time stamps. Then, it will go through the unallocated areas to try to find files or even fragments of files. Finally, it would index the data so that the investigator can search by file type, name, or even content. This can take a long time, especially if there are many drives to sort through.

It is not necessary for the investigator to search by hand for first tier triage. An investigator may only be looking for the presence of certain data that will indicate that the drive may contain high-value data. Cryptographic hashes are frequently used in computer forensics to build a digital signature that is unique to that file and can be used to compare two pieces of data and quickly determine if they are identical. The bottleneck is not the speed at which hashes are being compared, but the speed that data can be read from a drive. Young, Foster, Garfinkel, and Fairbanks presented an approach with the speed of hashing, but instead of using file hashes, they use sector hashes [7]. By using this technique, file systems, metadata, and other extraneous details can be discarded and the presence of known files can be detected merely by going through the hard drive from front to end, hashing each sector individually, and looking the hash

up in the database to determine if a known file is present. Using sector hashing, reading the whole drive is often not necessary. Garfinkel and Nelson used statistical sampling on a drive with known files and calculated that only a small fraction of the drive needs to be sampled to reach a relatively high probability of finding at least one sector containing target data [8]. In addition to being able to use sampling techniques, sector hashes allows file fragments and files that have been modified to be detected. Simon Key developed an EnCase script that uses this approach to rebuild incomplete files on drives from a good hash map [9].

By combining sector hashing and statistical sampling, investigators can perform drive triage in mere minutes instead of in hours. We present an optimal strategy for sector sampling triage, or more specifically, the ideal number and locations of sectors to minimize the sampling time while giving the maximum confidence if target data are not found. Building on the sampling pattern, we develop a confidence model and verify through experimentation that our model works and that sector sampling is in fact faster than a traditional full drive analysis. Finally, we attempted to describe how sector sampling should be performed, how much of a benefit it provides, and how an actual implementation would work. Over the course of the project, we also attempted to develop a quick way to estimate the time needed to achieve a certain probability of finding known files.

We began by testing and analyzing hard drive random sampling performance using small sample sizes and sector ranges to find sampling characteristics. Next, a probability model was developed that would take into account the unique ways that sectors can be read and contain file data. Then, the results of the sampling performance analysis was used to predict how long it would take to sample enough sectors to reach a target probability. Lastly, the probability and prediction models were tested on real and simulated drives to verify their accuracy and determine how long it takes to sample a drive. We developed a tool called `drivesampler` that incorporates the knowledge gained to efficiently find known data. In the remaining chapters, we discuss the concept and importance of the *transaction size* and how it influences every detail in efficient drive sampling. Chapter 2 provides a background on the limited prior work on sector sampling and hashing. Chapter 3 defines terms and discusses in detail how a drive is sampled efficiently and how the probability of finding files is calculated. Additionally, we explain how `drivesampler` was designed and implemented. Chapter 4 contains an overview of experiments to test performance and probability followed by the results of the tests. Finally, Chapter 5 concludes this work with an overview of what we discovered, potential weaknesses in our approach, and future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Related Work

To our knowledge, sector sampling has not been widely adopted or studied. We were unable to find many studies on hard drive performance and none regarding sampling performance. Random sampling in general has been used for a long time in many fields and forms the basis of the robust Monte Carlo method. In digital forensics however, applications of random sampling has mostly been used for sampling files to reduce the number of files that needed to be examined [10], [11].

In this thesis, we ignore the file system and use block based forensics—the analysis of file fragments. File blocks are being examined in a variety of ways to support this type of forensics. The National Software Reference Library (NSRL) which holds a large collection of known software recently hashed the library on 4096 byte block boundaries to explore the usefulness of this approach [12]. The ability to properly conduct block based forensics will be the key to the success of sector sampling. The following are some prior work we found in this area.

Chaudhuri, Das, and Srivastava [13] present a technique to use block based random sampling for identifying the contents of a large database. Their goal was to find statistics about a database, such as building a histogram of its data. The most straightforward way of doing this is by taking a uniform random sample of the data, however, this can be very inefficient because a single piece of data often does not fill up an entire sector and more data than needed are being read from the disk. This means that uniform sampling can be very expensive as most of the retrieved sector is being discarded and additional data are needed to obtain the required amount of samples. The natural solution is to not discard data and use the entire block that is read. The problem is that this is no longer uniform sampling so a bias may occur if data are not stored randomly, e.g., in sorted order. Chaudhuri, Das, and Srivastava explain how to do statistical sampling using entire blocks without having to discard data. Sampling the disk efficiently is important in the context of forensics, but bias is a non-issue if we are simply looking for interesting data and not comparing contents with each other.

Garfinkel and Nelson [8] showed that sector sampling is a fast way to identify the contents of a storage device. In the simplest case, sampling can determine if a drive has been properly wiped. By randomly sampling sectors we can determine the probability of finding a sector that is not

empty. This is a case of the "urn problem" in probability theory [14] and forms the basis of establishing confidence in sector sampling. The urn problem describes a simple scenario of an urn filled with two types (black and red) of balls. If a 1 TB disk is represented as an urn with two billion balls (sectors) of two colors (empty and non-empty sectors), and n balls are drawn without replacement, the probability of finding x of the K non-empty balls within N total balls can be represented with the hypergeometric distribution.

$$P(X = x) = h(x; n, K, N) = \frac{\binom{K}{x}\binom{N-K}{n-x}}{\binom{N}{n}} \tag{2.1}$$

More generally, this is easier to compute if we seek the probability that X = 0, or the probability of finding only empty sectors.

$$P(X = 0) = \prod_{i=1}^{n} \frac{((N-(i-1))-K)}{(N-(i-1))} \tag{2.2}$$

Using this general equation, we can find the probability of not finding data if we assume that 20,000 sectors (roughly 10MB) contain data on a two billion sector drive when 10,000 sectors are sampled, as shown in Table 2.1. We know that there are data on the drive, so we are interested in the probability of finding the data, i.e., how reliable it is. The probability of finding data is simply the probability of not finding data subtracted from one.

$$p = 1 - \prod_{i=1}^{n} \frac{((N-(i-1))-K)}{(N-(i-1))} \tag{2.3}$$

By sampling 500,000 sectors, or 0.025% of the drive, the probability of not finding data is 0.00673. In other words, there is approximately 99% chance of finding data and knowing that the drive has not been wiped properly. Table 2.2 shows the probability of not finding data when taking 10,000 samples with different amounts of non-empty sectors [8].

The obvious benefit to sampling a drive is that it can be performed in a fraction of the time that it would take to read the entire drive. Sampling 10,000 sectors in sorted order from a 1TB hard drive using eSATA took an average of 88.4 seconds. For the sample empty/non-empty problem above, a simple estimation shows that we can reject the hypothesis that the drive has less than 10MB of non-empty sectors within minutes and achieve 99% confidence in 40 minutes

Sampled sectors	Probability of not finding data
1	0.99999
100	0.99900
1000	0.99005
10,000	0.90484
100,000	0.36787
200,000	0.13532
300,000	0.04978
400,000	0.01831
500,000	0.00673

Table 2.1: Probability of not finding any of 10MB of data on a 1TB hard drive for a given number of randomly sampled sectors. Smaller probabilities indicate higher accuracy. From [8].

Non-null data		Probability of not finding data
Sectors	Bytes	with 10,000 sampled sectors
20,000	10 MB	0.90484
100,000	50 MB	0.60652
200,000	100 MB	0.36786
300,000	150 MB	0.22310
400,000	200 MB	0.13531
500,000	250 MB	0.08206
600,000	300 MB	0.04976
700,000	350 MB	0.03018
1,000,000	500 MB	0.00673

Table 2.2: Probability of not finding various amounts of data when sampling 10,000 disk sectors randomly. Smaller probabilities indicate higher accuracy. From [8].

compared with 200 minutes to read the entire drive. Two factors are discussed in their findings that can improve sampling time. They found that reading sectors in sorted order performed significantly better than random order. They also found that reading samples of 4KiB sectors was slightly faster than 512-Byte sectors [8].

The sampling technique is well suited when combined with research done to identify distinct blocks, or small unique chunks of data. Garfinkel, Nelson, White, and Roussev [15] analyzed methods to perform block based forensics by looking for distinct blocks. They present one approach using cryptographic hashes of individual blocks to identify files. They attempt to identify unique fragments of files that only occur in a single distinct file, which they call distinct blocks. Their work forms the basis of how these distinct blocks can be used with sector sampling to uniquely identify files on a hard drive. They found that user generated data such as documents, images, and videos have enough entropy to be unique in most instances. One of the problems they identified with block hashing rather than file hashing is that it takes a significantly larger amount of space to store the hashes. A SHA1 database of all blocks on a terabyte drive would require 40GB of storage which would not fit in the memory space of average consumer computers.

Young et al. [7] continued the work on file identification using block hashes by looking at ways to build a block hash database. Sampled drive blocks can be hashed and checked against the database to look for known files. As explained above, it is infeasible to store a hash database in memory so it must be stored on disk. The goal of the hash database was to "be fast enough to support searches of hashes that are created by reading a consumer hard drive at maximum I/O

transfer rate." The "maximum I/O transfer rate" was not determined by experimental data but a goal of 150,000 hash lookups per second was set based on the assumption that it takes approximately 200 minutes to read a terabyte drive. Read speed is less than optimal with sampling, so this rate is sufficient. Their article evaluated several back end databases for querying large amounts of hashes based on how many records were in the database.

This thesis is based on several major facts learned from these prior efforts. First, we know that sector sampling will save time. If we assume that there are even 10MB of data that we are looking for, then we do not need to read the entire contents of a 1 TB disk to achieve a high probability of reading at least one sector of the target data. Second, we know that individual sectors are likely to be distinct for user generated data, which are often searched for in forensics. Therefore only one known sector from most files is required to detect the presence of a file. This sets the stage for a quick method of forensic analysis by checking if a drive has user generated data that we are interested in.

CHAPTER 3:
Methodology

Given a drive, our objective is to give information about the contents of a drive to the analyst so that they can decide whether the drive is worth further investigation. Sampled blocks on a drive can be analyzed in different ways, but our approach is to look for *target data*. Target data are the content of any files that the user is looking for which are stored in a database in the form of block hashes. These are not hashes of entire files, but the hashes of each 4KiB block contained in the file. The basic strategy is to sample amounts of data across a hard drive, hash the data blocks, and check if the hashes are present in the database. If a hash is found in the database, then the drive is very likely to contain the files that the user is looking for. We cannot say this with 100% certainty because of a possible hash collision from different files or files that contain the same block. If no hashes are found after sampling, then a *confidence level* that target data are not present can be computed using the probability that the sampling missed the sectors containing the target data. We discuss how the confidence is calculated in Section 3.2.

The question then is how many sectors at a time should be read from a drive to maximize the confidence of finding target data in the shortest amount of time? In order to determine the answer to that question, we first examine how a drive is sampled, and then determine how to apply the "urn problem" to calculate the probability of finding target data. This chapter discusses our approach for applying sector sampling on a physical drive including potential problems, followed by the experimental setup, and lastly an overview of our software `drivesampler`.

3.1 Definitions

These are terms that have a specific meaning in the context of this thesis.

size

Size refers to an amount of digital information represented in bytes. Adding a prefix, such as kilo and mega, will scale the unit. The prefix can be an IEC binary modifier (powers of two) or a SI decimal modifier (powers of 10). The SI prefix is only used in the context of hard drive capacity. All other references to a size uses the IEC binary prefixes kibi (Ki) for 2^{10} and mebi (Mi) for 2^{20}.

sector

The smallest physical or logical unit of data addressed by a hard disk drive. 512 bytes

has historically been the most common sector size. Modern drives use a sector size of 4096 bytes to reduce the overhead space required to store metadata for each sector [16]. For compatibility with older operating systems, drives may use 512 byte emulation where data will be read from the physical disk using 4096 byte sectors but the data are passed through the logical interface using 512 byte sectors [17]. In this thesis, we assume that the physical and logical sector size is always the same, that a sector always holds 512 bytes, and that files start on sector alignments.

block

A block commonly refers to a group of data units. In this thesis, a block refers to a contiguous set of sectors based on their logical ordering on the drive. Sector ordering is determined by the drive controller, so they may or may not be physically next to each other.

transaction

A transaction is the transfer of one block of data from the hard drive to memory.

transaction size

The transaction size is the number of contiguous sectors read at a time in bytes.

transaction block

The transaction block refers to the data block in memory after a read operation (transaction) has taken place. We are able to successfully detect target data if a transaction block that is read contains one or more target blocks.

target data

Target data are any known files that the user is interested in finding. We attempt to locate target data by searching for target blocks.

target data size

The target data size is the amount of target data present on a storage device. This value must often be assumed when calculating sampling statistics because the true amount of target data is not known unless the entire contents of the device is checked.

target block or hash block

Target data are organized into target blocks by taking equally sized pieces of a file. The size of the pieces is called the target block size. If the file size is not divisible by the target block size, the remainder is currently thrown away. These chunks each become a target block and its computed hash is stored in a database. For this reason, target blocks can also be referred to ask hash blocks.

target block size or hash block size

The target block size is the size of the target block that is hashed. The target block size must be a multiple of the sector size. In this thesis, a target block size of 4KiB is assumed.

lookup

A lookup is the act of taking a target block retrieved from a device, computing the hash of the block, and querying a database to check for the presence of the hash.

confidence

The confidence refers to the probability that there are no more than a certain amount of target data, chosen by the user, on the target drive. The true probability cannot be known unless the content of the entire drive is already known, which would defeat the purpose of sampling. Instead of finding the true probability, we assume that some amount of target data are present on a drive in a way that is least likely to be found. Under these assumptions, the program can say that we are $x\%$ confident that there are less than the assumed amount of target data on the drive. See Section 3.2 below for a discussion on confidence.

3.2 Goals: Confidence and Time

Statistical sampling is a robust technique that can estimate the characteristics of an entire population by choosing only a small subset of the population. This has the advantage of lowering the cost to measure the population. The disadvantage is that this is only an estimate and the actual characteristics cannot be known unless the entire population is measured. An estimate always has some sampling error, and this is reported as the margin of error in surveys. The margin of error is dependent on the sample size, because larger sample sizes generally means higher estimation precision. Picking an optimal sample size can be a problem as the most accurate results is desired with the smallest sample size. There are various ways of determining how large the sample population for a survey should be, usually based on how much room for error is tolerated. In sector sampling, the population is the entire data content of the drive and we attempt to minimize the cost (time) to find interesting content, called target data, on the drive by sampling. The question is how big should the sample size be?

As stated in the beginning of the chapter, our objective is not to find all the target data, but to determine if any are present or not. After sectors are sampled from a drive and checked, there are two outcomes: target blocks were found or not found. If target blocks were found, then the user only found further proof of what they likely already suspected. The more interesting

and difficult case is when target blocks are not found. This could mean that target data are not present, or it could simply mean that the sampling missed the target blocks. There is no way to be certain that target data are not present unless the entire drive is read, but it is certain that some target data are present if at least one target block is found by sampling. In other words, it is much more difficult to prove that target data are not present which would be the more useful information for triage.

Instead of proving that target data are not present, we assume that there are some amount of target data present and calculate the probability that the sampling misses the target data. The probability can be calculated using the drive capacity, the amount of target data, and the sample size as discussed in detail in Section 3.3.1. We define *confidence* to be this probability subtracted from one because we are $x\%$ confident that there is less than the assumed amount of target data present on the drive. We use 10MiB as the assumed amount of target data as this would hold only two or three high quality pictures or a short video—likely not enough to be of significant interest if overlooked. With a confidence and size of target data, the sample size can be computed. For example, we can ask "how many samples are needed to be 90% confident that there are less than 10MiB of target data on this 1TB drive?" The sample size is directly related to the time spent sampling, so the question can be asked in other ways, such as "what % confidence can be achieved given 15 minutes of sampling?" The answers can vary even on the same drive depending on the sampling strategy and target data layout, both of which are discussed in detail in the following sections.

It is up to the user to decide on how confident to be. Obviously, the higher the confidence, the more samples are required and takes more time. Desired confidence and time spent must be weighed, but more time can always be spent sampling to raise the confidence. The entire drive can be examined to be 100% confident, but how much data is needed for a high confidence? Figure 3.1 shows the percentage of data required from a 1TB drive with 10MiB of target data to achieve a confidence when sampling 4KiB blocks at a time using our sampling strategy. It is noteworthy that only 1.4% of the drive is needed for 99% confidence.

3.3 Transaction Size and Sampling Strategy

Like a political poll where the sample population and location are selected carefully, some thought must go into how a hard drive is sampled. Unlike people, files are present on drives in various ways and could be spread across multiple sectors. A block is defined as contiguous sectors, and contiguous sectors may be required to successfully identify a file. It is therefore

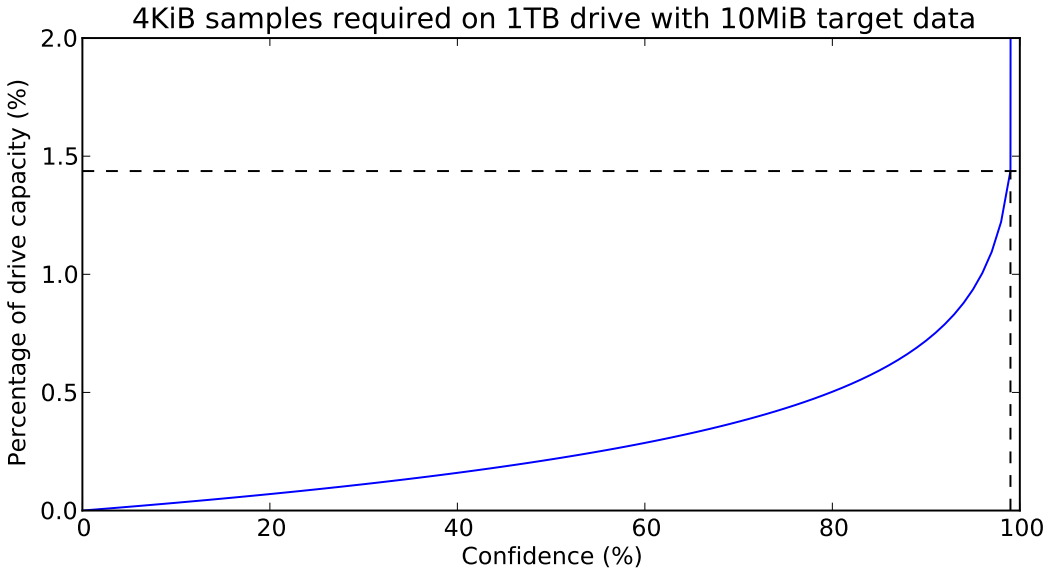

Figure 3.1: Percentage of data required from a 1TB drive with 10MiB of target data sampling 4KiB at a time. The dotted lines show that 1.4% of the drive is required for 99% confidence. Note that the entire drive is required for 100% confidence. The data required can change depending on the transaction size (see Section 3.3).

necessary to perform sampling in blocks, called transaction blocks, rather than individual sectors. The transaction size, or the size of the block that is being read, is something that we look at extensively because it has a huge impact on how fast sampling can be. The following discusses how transactions and target data are related.

For the purpose of our experiments, we use a target block size of 4096 bytes (4KiB). This means that we assume that target data are grouped into contiguous sectors of 4KiB. Any data less than 4KiB in size are considered too small to be identified. Young et al. [7] discuss the benefits and potential pitfalls of using 4KiB over smaller block sizes. Using a smaller block size means that target data do not have to be in as large contiguous sectors, but there is also less entropy to identify distinct files. However, using a larger block size also significantly reduces the number of records that must be indexed and searched when attempting to identify a given block. This does not mean that target blocks will be at 4KiB offsets. It is always assumed that the drive store data at sector offsets in sector sizes (512B). Figure 3.2 shows the layout of a drive with sectors and blocks.

Next, we consider the transaction size, or the amount of data read per sample. At a minimum, this should at least be the target block size to be able to identify target data. A smaller transaction

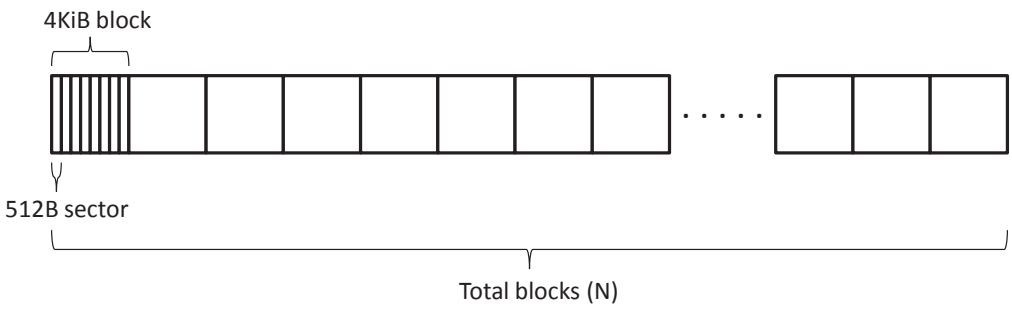

Figure 3.2: All data on a drive are stored in 512B sectors. It is assumed that target data are located in blocks of eight contiguous sectors, creating 4KiB blocks.

size will allow for more samples and increase the odds of locating target data. However, using a larger transaction size reads data faster because drives are optimized to read contiguous sectors. Choosing the most efficient transaction size is a difficult problem and can greatly affect the probability of locating target data. The simplest case is when the transaction size and target block size are the same. Let us assume for now that target blocks will always be block aligned. For this scenario, the drive can be divided into block size segments as seen in Figure 3.3.

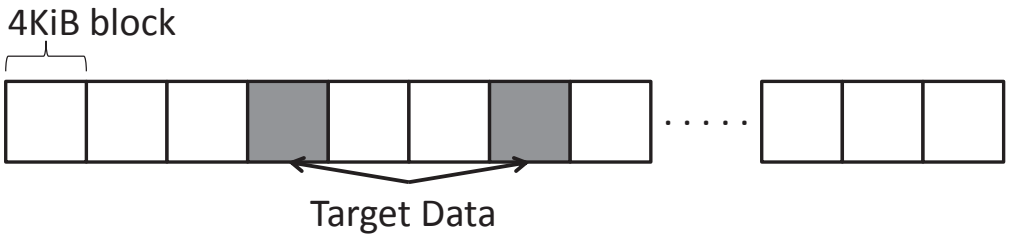

Figure 3.3: Target data must be on the drive in blocks. In our implementation, target blocks are 4KiB. This is a sample drive with two target data blocks.

This can be represented by the urn problem model where each ball represents a block of data and the balls combined represents all of the data on the drive. Each ball can either be a "red" target block or "black" non-target block. For every block that is read, a ball is removed from the urn. We can then use Equation 2.3 where M is the number of target blocks, N is the total number of blocks, and n is the number of blocks read to get the confidence of finding at least one target block.

The most important detail is that when the transaction size and block size are equal, one transaction yields exactly one block lookup. Let us now remove the assumption that target blocks

are block aligned and can now start at any sector boundary. If we continue to sample at block boundaries, only target blocks that are block aligned can be found. As seen in Figure 3.4 non-block aligned target blocks can be missed as each transaction will only see part of the data which is not enough to identify it as a target block.

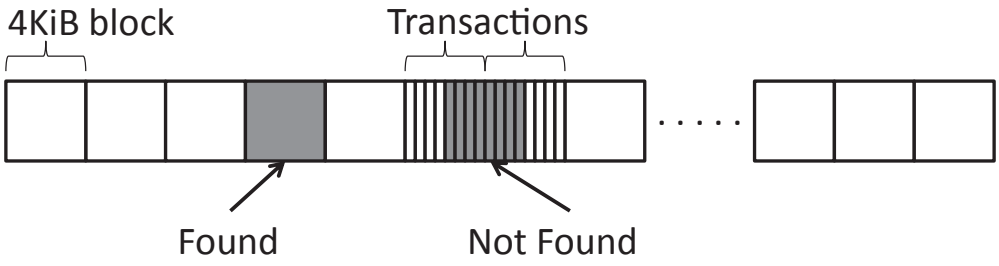

Figure 3.4: When sampling at block boundaries, target blocks are missed if they do not start on a block multiple and cross a block boundary.

The first way to get around this problem would be to sample at sector offsets. By sampling blocks at any sector offset, we capture every possible position that a block can be in. Unfortunately, this quickly creates a few problems. The initial problem is that if the previous algorithm is used for scheduling, sectors may be read multiple times, each time as a part of a different block which would be inefficient. If we use a smarter algorithm that avoids duplicate reads, this is no longer the simple urn problem and complicates the confidence calculation. We want to keep the scenario as an urn problem so that the confidence is fast and easy to compute.

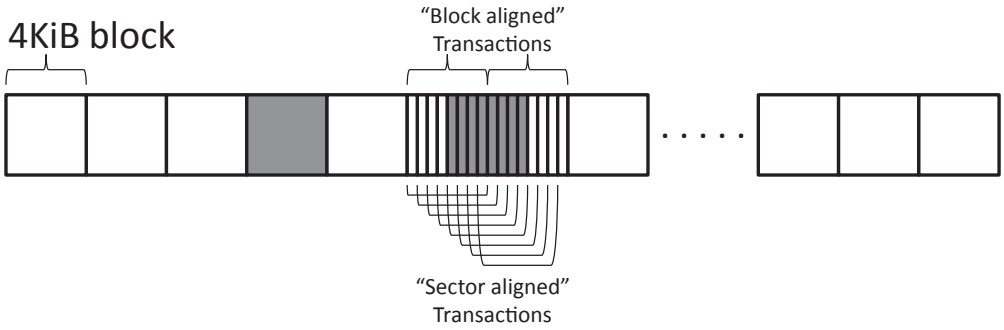

Figure 3.5: With sector aligned sampling target blocks at any offset can be found, but this causes redundant reads and complicates confidence calculations.

A better solution would be to use a transaction size that is larger than the target block size (see Figure 3.6). A larger transaction size increases the chances of finding target blocks as by reading more data at once, the start and end points do not have to be perfectly aligned. Additionally, it is

possible to look up all sector aligned blocks in the range of data read. For example, doubling the transaction size to 8KiB results in 9 block lookups and a 16KiB transaction size gives 25 sector aligned blocks. However, a larger transaction size means there are fewer samples obtained in the same time period as the smaller transaction size.

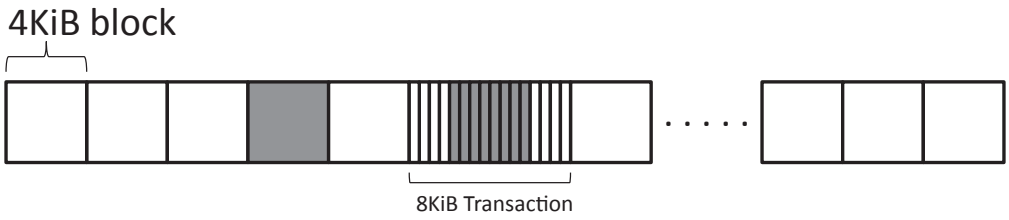

Figure 3.6: An 8KiB transaction size would be able to read 2 sequential blocks and check for target blocks at any sector offset for a total of 9 block database lookups. This is a huge improvement over 4KiB sampling.

With larger transaction sizes, non-block aligned blocks can be found as long as the transaction fully covers the block. A problem still exists because if reads only take place at transaction offsets, there is always the chance that the block spans two transactions as illustrated in Figure 3.7.

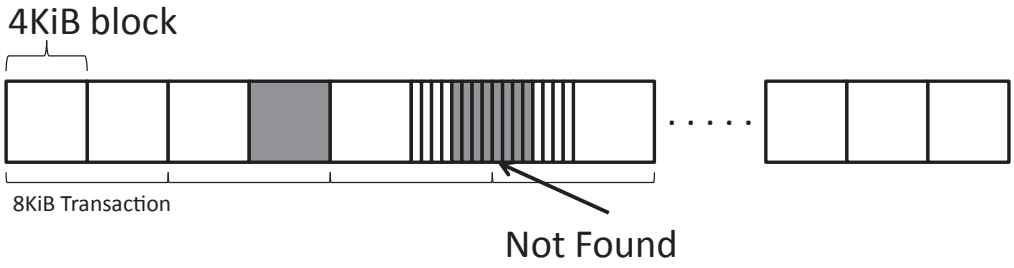

Figure 3.7: Using a transaction aligned sampling method will result in the original scenario where some target blocks are not found.

This problem of having blind spots can be fixed by slightly overlapping the transactions with each other. Missed blocks are those that begin near the end of a transaction and the entire target block does not fit in the transaction. Instead of starting the next transaction block at the end of the first transaction block, we start at the first block that does not fit in the first transaction block. The size of the overlapping area is always the block size minus the sector size, as shown in Figure 3.8. The downside to this method is that it is slightly inefficient as some sectors are read twice. Increasing the transaction size will reduce the total number of transaction blocks which minimizes the amount of overlapping sectors.

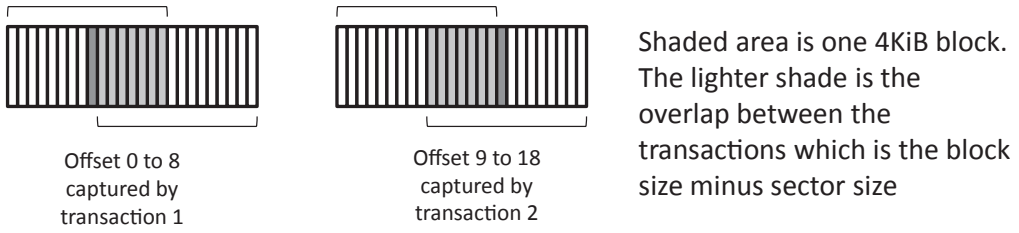

Figure 3.8: In order to capture all sector alignments, transactions must overlap by block size minus sector size.

3.3.1 Confidence Calculation

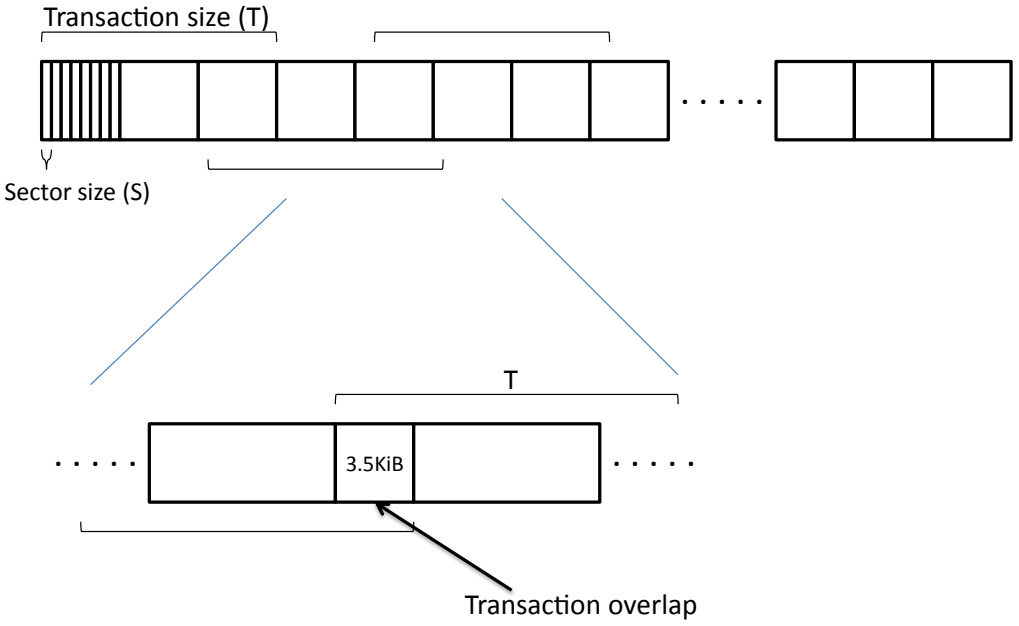

Figure 3.9: Data will be read from the drive in transaction size (T) blocks. With a target block size of 4KiB, the next transaction offset is 3.5KiB less than the end of the transaction window.

With our sampling strategy discussed in Section 3.3, we can now calculate our confidence. Figure 3.9 shows how transactions take place across a whole drive. This can be represented using the urn problem described above. Given N total balls (all possible transactions) of which K of them are red balls (transactions containing target blocks), we can find the probability of removing at least one red ball after n removals without replacement (unique transactions) using Equation 2.3. In other words, we find the probability of getting at least one transaction containing a target block.

N and K can be found before sampling begins. N is total amount of possible transactions which can be defined as

$$N = \left\lceil \frac{C}{(T-(B-S))} \right\rceil \tag{3.1}$$

where C is the total data capacity of the drive, T is the transaction size, B is the target block size, and S is the sector size.

K is the number of transactions that contain target data. This value can change depending on the layout of the target data on the drive. Different ways that target data can be on a drive are discussed in Section 3.3.2. Here, we assume the worst case layout to find the lower bound of the probability. In other words, we use the minimum number of transaction blocks that can fit the target data. Then

$$K = \left\lceil \frac{D}{T} \right\rceil \tag{3.2}$$

where D is the amount of target data and T is the transaction size.

Assuming that we have found an optimal transaction size, the only remaining variable is n which is the number of transactions which can only be determined at run-time and increases over time. Now, if we plug these values into Equation 2.3, the confidence is

$$p = 1 - \prod_{i=1}^{n} \frac{\left(\left(\left\lceil \frac{C}{(T-(B-S))} \right\rceil - (i-1)\right) - \left\lceil \frac{D}{T} \right\rceil\right)}{\left(\left\lceil \frac{C}{(T-(B-S))} \right\rceil - (i-1)\right)} \tag{3.3}$$

where C is the total data capacity of the drive, D is the amount of target data, T is the transaction size, B is the target block size, and S is the sector size. All variables except for n are constant at run-time once the optimal transaction size is determined. Using this equation, the number of samples required for a target confidence is computed and then sampled. Figure 3.10 shows how using different transaction sizes affect the number of samples required. The key here is that a small transaction size requires fewer samples than a larger transaction size. This is logical because doubling the transaction sizes roughly halves the number of total transaction blocks on the drive. Fewer total transactions means that more samples must be taken to achieve the same confidence. However, this does not necessarily mean that smaller transaction sizes are better because the speed gained by reading larger transactions may offset the disadvantage.

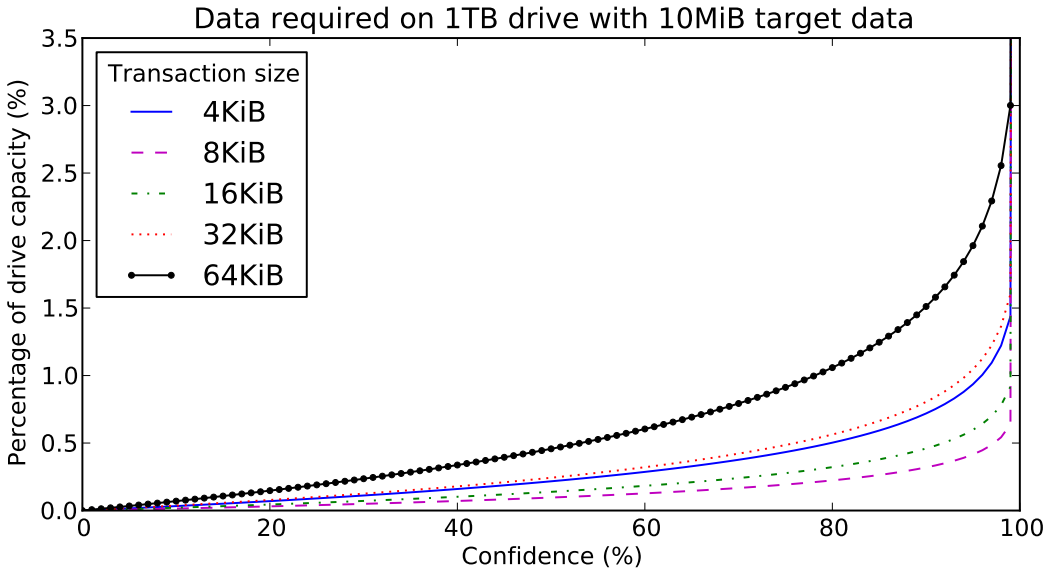

Figure 3.10: Percentage of data required from a 1TB drive with 10MiB of target data using different transaction sizes. The entire drive must be read for 100% confidence. Generally, smaller transaction sizes means that less data are required. 4KiB is a special case discussed in Section 3.3.3.

3.3.2 Target Data Layout

Target data so far have been assumed to always be in 4KiB blocks and that anything less than a block can not be found and hence does not count as a red ball, however, most files are not likely to be exactly 4KiB. Files can be much greater, and there is no guarantee that the data exist in sequential sectors. We call an arrangement of the target data on the drive a "target data layout." Unfortunately, the target data size or layout is not known unless the entire drive is read and analyzed so we must make an assumption about how much target data are on the drive and where the data exist on the drive.

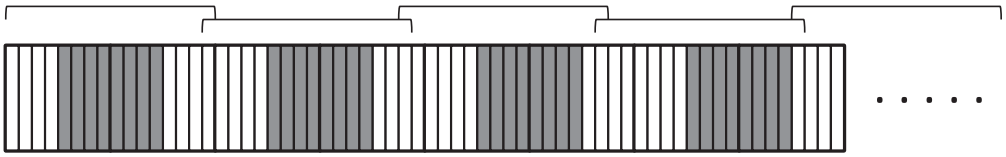

Figure 3.11: Best case layout example. The best case target data layout is when there is exactly one target block per transaction block. The target data are spread out across as many transaction blocks as possible.

For uniform random sampling, the probability of finding target data is highest when the target

data are spread out through as many transactions as possible in blocks of an identifiable size (4KiB in our case). As seen in Figure 3.11, in this scenario there is exactly one target block contained in each transaction block, no more and no less. This maximizes the transactions that contain a target block, or in other words this is the maximum number of red balls in the urn problem analogy. We call this the best case layout. However, most file systems tend to group blocks of the same file together to avoid fragmentation of files, so files larger than 4KiB are more likely to be contiguous. This increases the number of target blocks per transaction which means that there are fewer transactions containing target blocks, reducing the chances of finding target data.

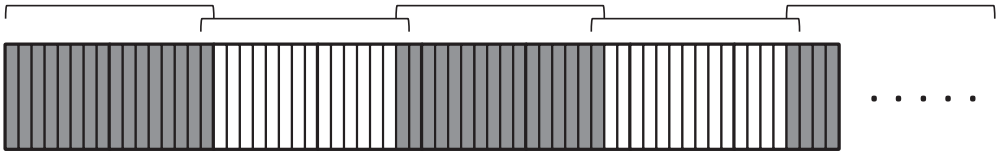

Figure 3.12: Worst case layout example. The worst case target data layout is when non-contiguous transaction blocks are filled as much as possible with target data. This creates the minimum number of target transaction blocks.

The worst case layout is when all target data are distributed across as few transactions as possible. This occurs when target data are in blocks equal to the transaction size and are not in contiguous transaction blocks. They cannot be contiguous because of the transaction overlaps. Contiguous transaction blocks share some data so less target data are required to fit two contiguous transaction blocks than non-contiguous transaction blocks. In the urn problem analogy, this layout minimizes the number of red balls.

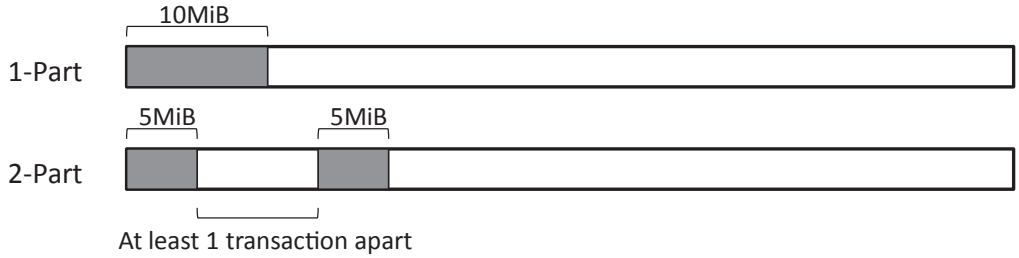

Figure 3.13: N-part layout example. Files are more likely to be in chunks, so the n-part layout is used for a more realistic scenario. The target data are split into n equally sized chunks and placed in non-contiguous transactions.

In real-world environments, it is unlikely to see the best or worst case layout but we will always

assume the worst case when calculating the confidence. By using the worst case scenario, the confidence cannot be any lower. There is one special case where this is not true when the transaction size is equal to the target block size, and is discussed in Section 3.3.3. We also experimented with other layouts such as the *n*-part layout, where *n* is the number of equal parts that the target data are split into and placed in non-sequential transactions.

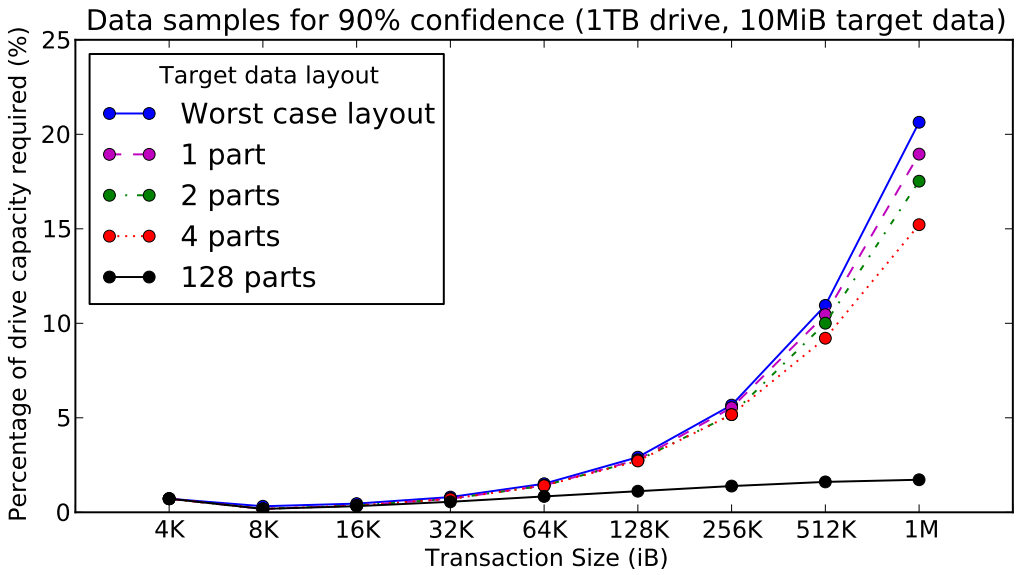

Figure 3.14: Percentage of data required from 1TB drive for 90% confidence with different layouts. Splitting the target data into more parts increases the number of target transactions and fewer samples are required. The more extreme 128-part layout clearly shows this effect.

Figure 3.14 shows the effects that the target layout can have on the data samples required. The *n*-part layouts produce slightly more target transaction blocks than the worst case because of the overlapping portions of contiguous transaction blocks sharing target data. While the shared data is small compared to the non-shared data, there is a noticeable difference for very large transaction sizes. Similarly, the 2-parts and 4-parts layouts produce a similar effect of slightly increasing the number of target transaction blocks at large transaction sizes by splitting the data into smaller pieces. The smaller transaction sizes are not as affected because when the target data are broken up into small pieces, large transactions are not fully filled with target data and generates extra target transaction blocks. A very high *n*-part layout is close to the best case scenario. A 2560-part layout would be equivalent to the best case scenario for 10MiB of target data because 10MiB can be split up into 2,560 4KiB pieces. This effect is clearly demonstrated in the 128-parts layout because each transaction block only contains a small piece of the target

data, increasing the number of target transactions. For example, 10MiB of target data fills 10 1MiB transaction blocks in the worst case layout which puts as much target data as possible into non-contiguous transaction blocks. In the 128-parts layout, the target data are split up into 128 pieces of 80KiB each, so if the transaction size is greater than 80KiB, not all of the transaction block is target data. This inflates the number of target transaction blocks to a minimum 128, one for each piece, even though the target data could fit in just 10. The more pieces that the target data are in, the more transaction blocks they occupy and therefore fewer samples are required. Unfortunately, we cannot take advantage of this because it is impossible to know the target layout on a drive without reading and analyzing the entire drive, so our software assumes the worst case at all times. In actuality, the probability is likely to be higher than the value calculated using the worst case assumption.

3.3.3 The 4KiB Problem

Although we defined the "worst case" target data layout, it is not truly the worst case when the transaction size is equal to the target block size—4KiB in our implementation. A non-worst case layout with small transaction sizes should generally have a much higher confidence than the worst case layout because there are more blocks containing target data. This is not true when the transaction size and target block size are equal. Due to our sampling strategy and target block hash database, not all transactions on the drive containing target data are actually target transactions. Consequently, more samples are needed for 4KiB transactions than other larger transactions for the same confidence. Recall that blocks are sampled to hash them and query a database to see if it is a known block. In our implementation, target blocks are stored in the database only at target block offsets rather than sector offsets to avoid hashing redundant data.

Assume that our target data are in a 1-part layout, all together in contiguous sectors starting at the beginning of a drive. The first transaction block, which covers the first 4KiB block starting at offset zero, is in our database because it is on a target block size boundary. However, due to overlapping transactions, the second transaction block begins at sector offset seven or 3.5KiB into the drive. The block will not be in the database because target blocks only begin at 4KiB offsets. Therefore, the second transaction block, even though it is full of target data, is *not* a target transaction. Similarly, the next six transactions will not be target blocks. Only every eighth block will be a target block when the beginning of the transaction aligns with a 4KiB boundary again. Figure 3.15 illustrates how transaction blocks on disk are unaligned with the hash blocks in the source file. This means that there are less target blocks when the transaction

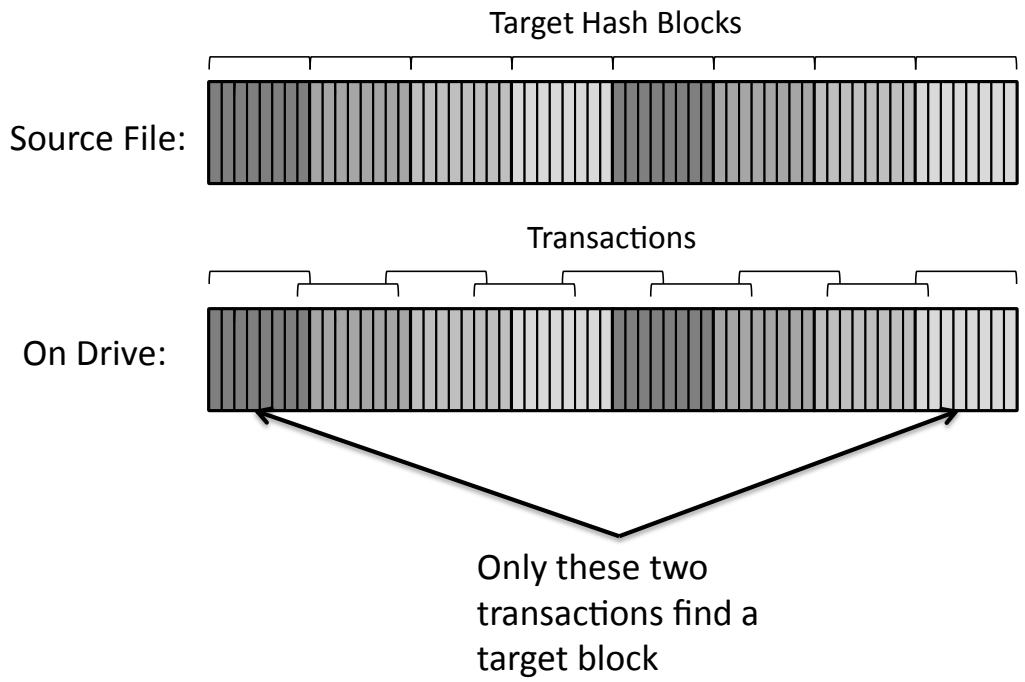

Figure 3.15: The 4KiB problem. Transactions are overlapping so when the transaction size is equal to the target block size, the transactions do not align with the target blocks. As shown here, all transactions contain target data, but only every eighth transaction block contains a target block.

size is equal to the target block size than when a larger transaction size is used. This effect is clearly seen in Figure 3.10 which shows that 4KiB transactions requires more data samples than 8KiB and 16KiB transactions for the same confidence. Furthermore, there can be even fewer target transaction blocks than the worst case layout in this situation.

There are two ways to fix this problem. First is to store every sector offset in the database so that a match will be found no matter where in the file the transaction begins. However, this will make the database eight times larger and the same sectors would be stored multiple times. The second and better way is to simply not use a 4KiB transaction size. The results in Chapter 4 show that sampling using a larger transaction size is much more efficient, and is not affected by this problem because a using a transaction size that is double the size of the target block size will contain enough block offsets that at least one will be on a 4KiB boundary. There is simply no advantages to sample using the target block size. The transaction size should be a minimum double the target block size for an increase in both performance and probability.

3.3.4 Scheduling

Scheduling is the process of deciding which blocks to sample, and is determined by a software component called the scheduler. Our scheduler implements the sampling strategy described in Section 3.3 as well as additional strategies that focus on the most efficient sampling of arbitrary drives. Garfinkel and Nelson [8] found that it is faster to read random blocks sequentially than to read them randomly for both magnetic disk drives and solid state drives. This property can be exploited by generating a list of block numbers ahead of time, sorting them, and then reading the blocks in order.

A topic that has come up frequently during reviews of scheduling is the decision to use fully uniform random sampling. With uniform sampling, any block of data are equally likely to be read. We could target specific areas of the drive depending on the file system present. There are several reasons why we stuck with uniform sampling. First, using any kind of metadata would mean that we need to develop rules for each file system type, creating much more work. Second, if the sampling is biased in any way, an adversary who wants to hide their data would know which areas of the drive were less likely to be sampled. A fully uniform scheduler is the only guaranteed method of not creating a safe haven for the adversary. Finally, the last reason is that it would complicate our probability equations. The urn problem is simple to understand and use, but if our samples are biased, then it is no longer a valid model. Targeting specific file systems could potentially increase our chances of finding target data, however, it could also decrease our chances or the probability model may be too complicated to compute the confidence for.

3.4 Experiments

After detailing how sampling would be done, we tested physical drives to measure the actual time taken to sample a drive. The main objective of the experiments was to determine the optimal transaction size on a drive which gave the maximum confidence in the minimal time, and find a way to quickly test a drive to find the optimal transaction size. As discussed previously, the confidence will change depending on the transaction size. Using a small transaction size increases the number of samples which improves the confidence. On the other hand, a large transaction size increases the rate that data are read from the drive. Either could potentially increase our chances of finding target data. Using our software, we measured the time it takes to sample different numbers of samples using different transaction sizes.

When conducting the same experiments repeatedly for multiple trials, we were careful not to

Drive	Brand/Model	Model Number	Capacity	RPM
Hard Disk 1	Seagate Barracuda	ST500DM002	500 GB	7200
Hard Disk 2	Seagate Barracuda	ST31000528AS	1TB	7200
Hard Disk 3	Western Digital Green	WD15EARS	1.5 TB	5400
Solid State 1	Samsung	MMD0E56G5MXP-0VB	256 GB	N/A
Solid State 2	Intel	SSDSA2CW600G3	600 GB	N/A

Table 3.1: List of drives used for experiments.

bias results. The danger was that data being read were cached, making read operations faster than they should be. The operating system cache in memory was cleared after every individual trial by running the following commands:

```
# sync; echo 3 > /proc/sys/vm/drop_caches
```

which writes any in-memory changes to disk and then removes cached content. The other cache that we considered was the disk cache. Modern hard drives contain a small circuit inside the drive called the disk controller. Data are read and written by communicating with the controller which handles the details of the operation. Controllers can be very intelligent, utilizing algorithms and cache space to predict sectors that might be read in the near future and cache them ahead of time. We first attempted to disable this feature, but found that because controllers are designed by each individual manufacturer, there was no universal command to disable the disk cache or any guarantee that the controller would actually obey the command. Instead, our software was programmed to not read from the same region of the disk twice if possible, so that the cache could not be utilized. We prepared five consumer grade hard drives for testing: three hard disk drives (500GB, 1TB, and 1.5TB) and two solid state drives (256GB and 600GB drive). The model numbers for all drives are in Table 3.1. All experiments were conducted on a HP 8570p laptop running a minimal installation of Fedora 18. The hard drives were attached to the laptop using an external hard drive dock connected with eSATA.

In addition to timing tests, we verified our probability model using simulations. This allowed us to quickly generate results as well as try out different target data layout images as described in Section 3.3.2 without having to re-write large amounts data to a disk for each test. The simulations had a dual purpose of checking that our probability models were correct and also for getting an idea of what the true confidence is when sampling under worst case layout assumptions. It is very unlikely for the worst case scenario to actually occur, so our confidence is actually higher in practice. As we will see in Chapter 4, the true confidence was much higher.

3.5 Application Overview

We built a fast and effective forensics tool called `drivesampler` for our experiments that can locate known content on a hard drive using sector sampling to search for known blocks. This tool does not use conventional forensic techniques such as analyzing the file system or file carving. We are able to return results quickly because the application is based on uniform random sampling which can immediately produce probabilistic results. The confidence level improves as more time is spent sampling more data.

Our tool is only effective in situations where the user is looking for known data.[1] One example is a law enforcement officer searching for evidence of child pornography from a large number of storage devices. This scenario works because it is unlikely for a user in possession of child pornography to only have files that have never been found before. As long as a database is kept with block hashes of child pornography files that have been previously found by law enforcement, sector sampling can quickly find traces of a file that might be on the drives.

We have also placed a large emphasis on speed to be effective in a situation where time is very limited. Sector sampling will never be a replacement for a thorough analysis of a hard drive, however, it can perform a preliminary to determine how much time should be invested on that drive. These time constraints may be caused by any number of reasons. For example, an examiner may have a hard deadline such as a case trial date and needs leads to find evidence quickly. Some situations may call for speedy processing, such as a customs and border agent who wants to quickly check hard drives for suspicious data.

The tool does not require a technical expert to use it effectively. The process was designed in a way that minimizes user intervention and can process a drive in the most efficient way automatically. The results are easy to comprehend by most users as they are told if the tool has found any known content on a drive and how confident it is with finding data. The user may then act accordingly based on that confidence level. A forensic examiner may be more interested in more detailed data such as exactly what kind of files were detected and how many samples were taken.

3.5.1 Requirements

When designing the software, the following were the minimal requirements.

[1] While outside the scope of this thesis, our tool has a mode to check a wiped drive which scans for non-null bytes. The confidence model is different than the one described here for this mode due to a different worst case layout.

Non-Functional:

- Compiles and runs on major Linux distributions with freely available tools and libraries (Windows support secondary)
- Must support a command line or graphical interface (command line prioritized)
- Will not write data to the target drive under any circumstances

Functional:

- Usable as a stand alone tool or with a back end database
- Provides real-time feedback to user about current progress of sampling and analysis
- User can give either a target confidence or time, and the program will find the optimal sampling strategy

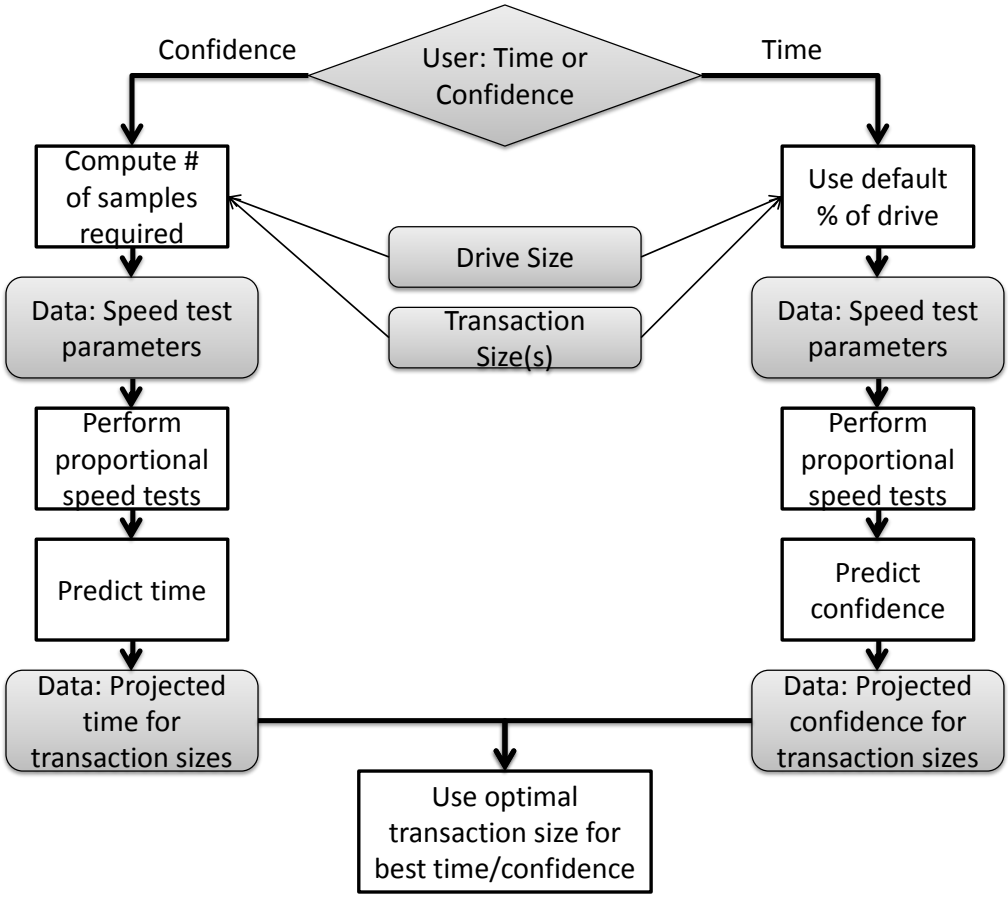

Figure 3.16: Flowchart showing how the optimal transaction size is chosen based on the parameter given by the user.

3.5.2 User-Program Interaction

1. The user has a hard drive to scan for any known files.
2. The user connects the hard drive to a computer running Linux using an eSATA port.
3. The drive is detected as a storage device by the operating system.
4. The user runs `drivesampler` with the device name and also sets either a target confidence or a time.
5. The user can pass additional options to override the default values, such as the assumed target data size and target confidence level.
6. The program samples the drive to estimate sampling speed.
7. If a time constraint is given, the program attempts to maximize the confidence that a given amount of target data are not present.
8. If a target confidence is given, the program attempts to minimize the time.
9. The user sees the speed at which the drive is being sampled and the estimated time/confidence.
10. The screen is continuously refreshed with any information found by the analyzer.

3.5.3 Technologies Used

C/C++ The primary programming language used for the development of `drivesampler` and all experiments. The widely supported GNU Compiler Collection (GCC) was used to compile the code. C++ was chosen for its high performance and portability.

Boost Free cross-platform libraries for C++ supporting a variety of common tasks. Boost was used primarily for threading support and the timer library for measuring read speed accurately. License: Boost software license.

Intel Thread Building Blocks (TBB) Free cross-platform library with components to support development of multithreaded software. TBB has a variety of threading-compatible containers which are not a part of Boost. Some of these containers were used to manage data being passed between different threads. License: GPLv2 with run-time exception.

3.5.4 Software Architecture

Sector sampling can be done in different ways and be used for various purposes, so our architecture was designed to be able to accommodate these needs. The central component is called the scanner which accepts rules and constraints, known as options, from the user interface. The scanner then load three independent components: a scheduler, reader, and an analyzer. These components are modular and have defined input and output interfaces so that they can be replaced. We identified these three components to be the key parts that need to be made modular

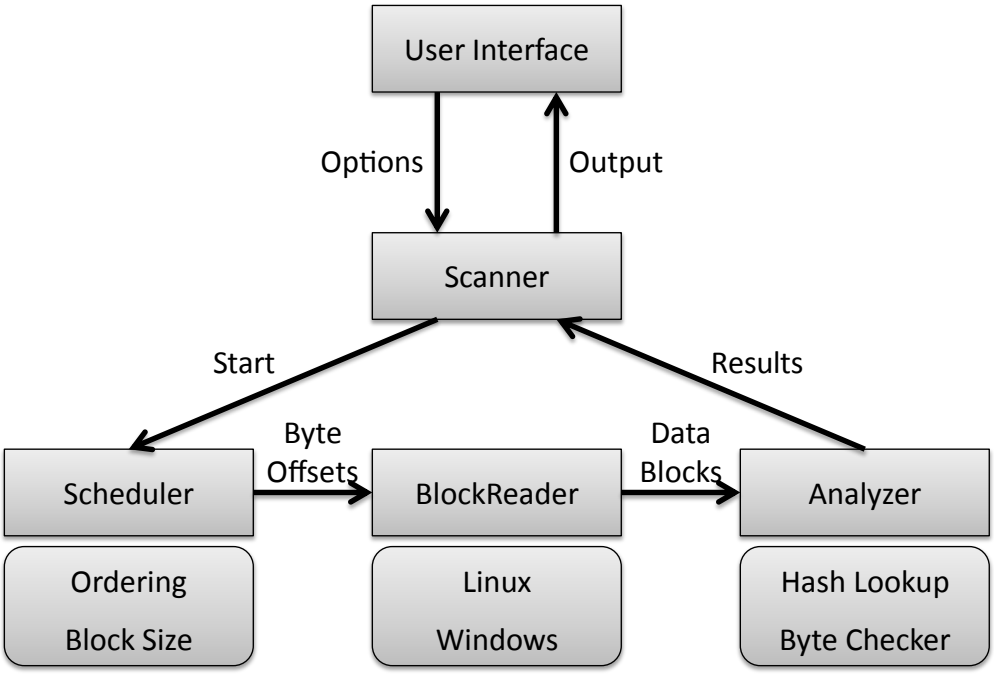

Figure 3.17: The software architecture. The scanner is the core of the program and controls three components: the scheduler, the reader, and the analyzer. Each component can be replaced or expanded as needed. The scanner receives input from the user interface and returns results.

to give flexibility for any sector sampling task.

The scheduler, as discussed in Section 3.3.4, is responsible for determining the blocks to read and their order based on the requirements such as the target confidence. The reader reads the data from a drive or file by following the order given by the scheduler. The data in memory are then passed to the analyzer which checks for interesting data such as known blocks. Finally, the analyzer's findings are sent back to the scanner which can output its results to the interface. Each component runs in its own thread as each has a potential to be a bottleneck in the process. By using multiple threads, one component will not delay the next from running until it has completed.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Results

We performed tests with the goal of finding the optimal transaction size for sampling and by examining the effects on speed and probability, a range of good transaction sizes appeared. Overall, results show that the three hard disk drives tested had similar characteristics, regardless of differences in capacity or speed. There were some minor differences but there were no significant outliers that stood out from other drives. There are massive quantities of hard drives in the world and we tested only a handful of them, so our tests were focused on testing drives against themselves and each other with repeated tests while staying mindful of findings that could prove useful for unknown drives.

4.1 Probability Verification

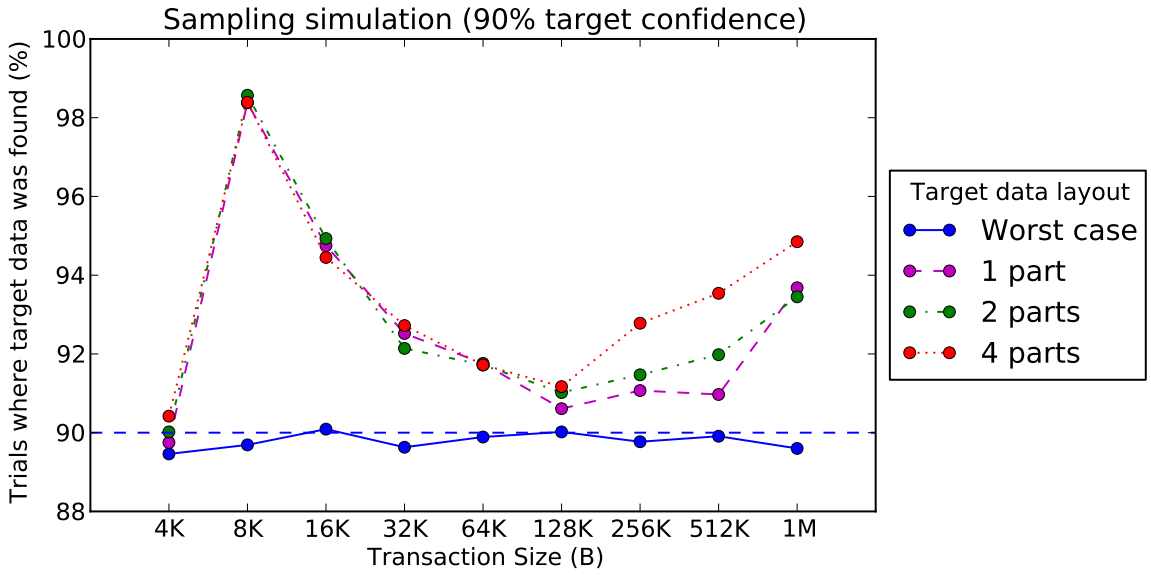

Figure 4.1: Percentage of 10,000 simulated sampling trials on a 1TB drive with 10MiB of target data that read a target transaction block using a target confidence of 90% (dotted line) assuming a worst case layout. The worst case simulation follows the target confidence as expected, and n-part layouts do better than worst case especially for larger transaction sizes. N-part layouts at 4KiB transactions behave similarly to the worst case layout due to the 4KiB problem (see Section 3.3.3).

We prepared a simulated environment of 1TB drives with 10MiB of target data randomly placed on the drive using several different layouts. The drives were then randomly sampled with no

knowledge about the layout with a target confidence of 50% and 90% assuming worst case layout and 10MiB target data. We performed the test 10,000 times for each drive and counter the number of trials where target data were found.

These results verified that our probability model was accurate and also confirmed our hypothesis that the true probability of finding data is higher than the worst case confidence. When the target data are divided into many pieces like the 128-part layout, the increase of target transaction blocks significantly improves the chances of locating one. Figure 4.1 shows the percentage of the trials where target data were found using a 90% target confidence. The worst case line on both graphs follow the target confidence closely, which is expected as the sample size calculated is based on worst case assumptions. When the target data are split into pieces, we find the targets more often because there are more target transaction blocks. In other words, the drive is being over-sampled, especially for layouts that have more parts. Similar results were seen for other target confidence, but lower target confidence generally showed an even greater number of times where the target was found.

4.2 Optimal Transaction Size

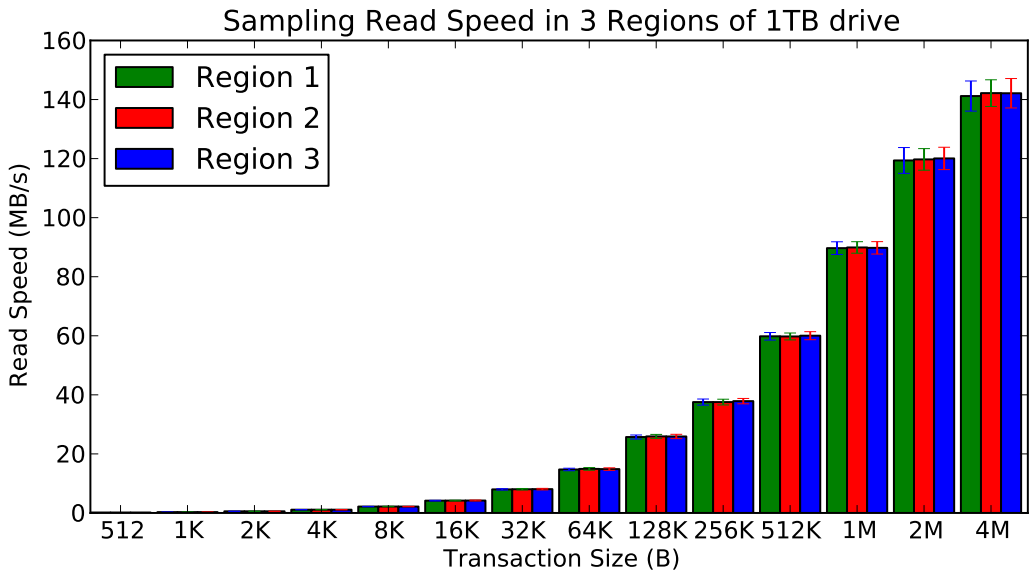

Figure 4.2: Sampling read speed in different "regions" of the 1TB drive. The green, red, and blue bars represent the first, middle, and last third of the drive respectively. Sampling speed is consistent across different regions of the drive. The results shown are averaged from 1000 trials.

We began by checking whether different areas of the drive, or "regions", behave differently.

The concern was that the physical location of data may affect sampling speed due to differences in seek time and rotational latency (the time needed for the disk sector to rotate around to the head). To do this, we divided a test drive into three equal regions and performed the same uniform sampling test of 1,000 blocks at different transaction sizes. There was no noticeable difference between the three regions on any drive. Figure 4.2 shows that the read speed across different regions on the 1TB drive was nearly the same. Based on these results, further tests were conducted across the whole drive instead of smaller regions.

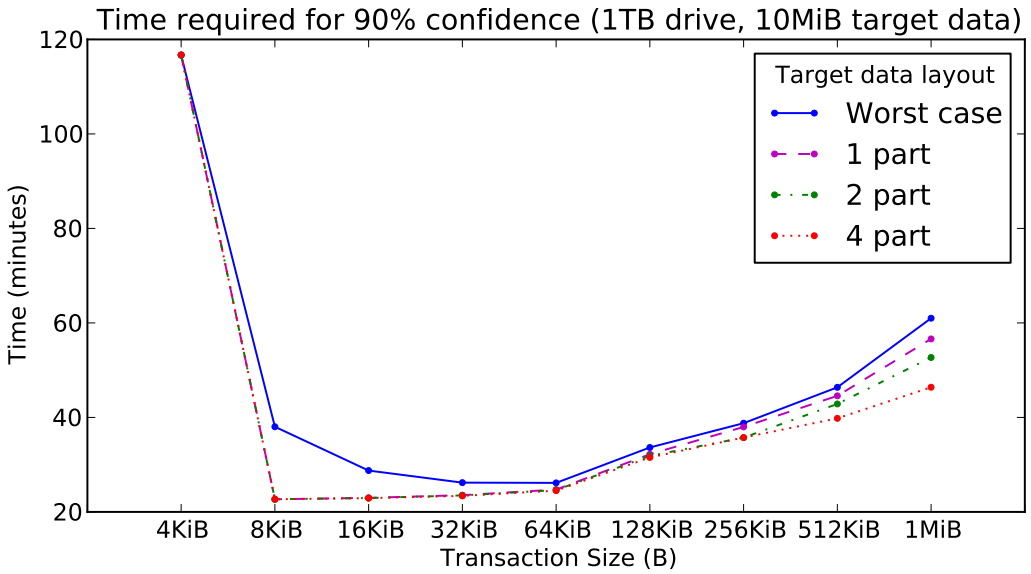

Figure 4.3: Time required for 90% confidence for 1TB drive with 10MiB target data. For the worst case layout, 64KiB was the optimal transaction size. It was able to achieve 90% confidence in 26 minutes. If the layout was known, 8KiB was the optimal transaction size.

With a better understanding of how well our probability model works and characteristics of hard drive sampling, we now test how long it takes to sample a hard drive to achieve a certain confidence level with our application. We first timed how long it would take to sample for 90% confidence if the layout of the drive was known. If the target data layout was known, then the smallest transaction size of 8KiB was the fastest. However, because the layout is not known, we found a 64KiB transaction size to be the fastest for all of our tests on the three test hard disk drives. 32KiB followed very closely as the second fastest transaction size, and 16KiB in third place. Figure 4.3 show the results of a 90% target confidence sampling on the 1TB drive which shows that 64KiB is the ideal transaction size for this drive. One could argue that a smaller transaction size is probably a more practical transaction size because the worst case

layout is very unlikely to occur. We decided that 64KiB is a good compromise because its speed was similar for both worst case and non-worst case layouts. This ensures that the optimal transaction size is used should the drive happen to have the worst case layout, and there is no need to guess the true confidence because it is roughly the same.

4.3 Speed Prediction and the Quick Test

By testing for certain confidence levels at different transaction sizes, we concluded that 64KiB was the ideal transaction size for our test drives, however, there is a major problem with this. We only found that 64KiB was the ideal transaction size after we had sampled the drive multiple times using all transaction sizes. All of our test drives yielded the same result, but there is no guarantee that all hard drives would behave the same way. If sector sampling is to be used for a forensics tool, there needs to be a way to find the optimal transaction size on any drive before sampling. This research began with an operational purpose in mind, so this problem was being considered from the beginning.

In addition to identifying drive sampling characteristics, a secondary objective was to develop a "quick test" that can determine the optimal transaction size within a short time frame. Our initial target time frame was 15 seconds, but we found that the quick test could be even less than 10 seconds. This quick test is crucial if the user will be allowed to give a time limit as we would need to know roughly how long it will take to sample for each transaction size. Based on all earlier results, we narrowed down the optimal transaction size range to be between 8KiB and 128KiB. We believe this is wide enough of a range without going into 4KiB or 256KiB which had the biggest drop in speed compared to its neighboring sizes. We sampled 1000 blocks, a small number compared to what is needed for a high confidence, to estimate the time for each transaction size. This failed for two reasons. First, the data were inconsistent for each drive because the seek time is different depending on the drive size. The difference between the estimated time and the true time varied greatly. Second, sampling 1000 blocks for each transaction size took over half a minute which was double our target time frame. We also experimented with 10 and 100 samples, but then the variance became too large because the seek time changed greatly, especially for larger drives.

After trying different methods, we found that sampling a small number of blocks from a proportional size area of the drive size was more accurate than sampling the whole drive. We knew from our earlier tests that sampling different regions of the drive did not affect the speed, so sampling only a small portion of the drive was sufficient. For example, if 100,000 transaction

	Transaction Size					
Samples	8K	16K	32K	64K	128K	Total
10	0.052	0.055	0.058	0.061	0.080	0.306
100	0.528	0.554	0.577	0.609	0.800	3.068
200	1.044	1.097	1.143	1.208	1.617	6.109
300	1.570	1.658	1.727	1.832	2.485	**9.273**
1000	5.374	5.638	5.880	6.229	8.469	31.59

Table 4.1: Average time required (in seconds) for one pass of a quick test based on 1000 trials. The increase in time is linear to the number of samples. Sampling 300 blocks or 100 blocks three times for the whole range of transaction sizes takes less than 10 seconds.

blocks were required for the target confidence, we could estimate the time by sampling 100 blocks from 0.1% of the drive $\left(\frac{100}{100{,}000} = 0.1\%\right)$. The average seek time between transactions would be the same, so the greatest cause of the variance was mitigated. This time could be multiplied by 1,000 to obtain a estimated projected time to sample 100,000.

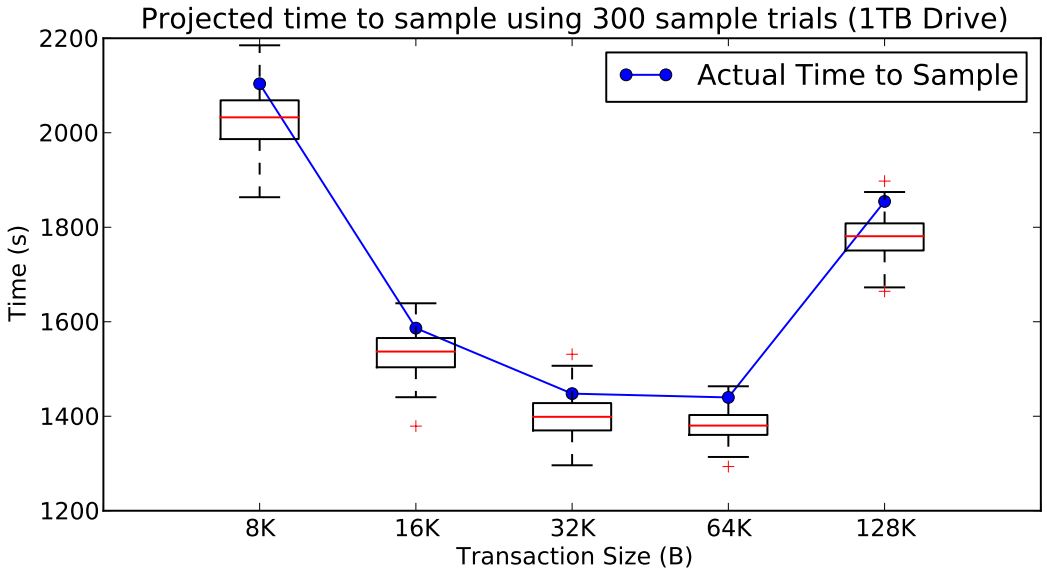

Figure 4.4: Time projected to sample for 90% confidence on 1TB drive using 300 samples. The estimated time underestimated the actual time (blue line) more than half of the time, but successfully guessed the optimal transaction size of 64KiB more than half of the time (64%). The remaining trials guessed 32KiB as the optimal transaction size.

The 100 block quick test successfully guessed the optimal transaction size of 64KiB 54% of the time. It made a slightly incorrect guess of 32KiB 44% of the time, and a significantly incorrect guess of 16KiB 2% of the time. As seen in Table 4.1, a quick test of 100 samples only took 3

	Sample Count					
Transaction Size	**100**	**200**	**300**	**100 x2**	**100 x3**	**100 x5**
8K	90.392	74.008	59.688	71.276	60.034	40.668
16K	71.110	53.647	44.216	48.720	39.943	31.545
32K	61.980	43.012	40.431	45.799	36.666	28.048
64K	57.348	39.639	30.453	40.415	36.494	28.898
128K	67.444	49.008	41.430	52.051	45.454	38.103

Table 4.2: Standard deviation of 1,000 trials for projected times using samples of 100, 200, 300, and averages of two, three, and five 100-sample trials at different transaction sizes. The greater the value, the more variance seen in the trials. There is no significant difference between using 300 and three 100-sample trials and takes about the same amount of time. Using three 100-sample trials may be better so that outliers can be detected and discarded.

seconds, so extra time can be used to improve the results. Increasing the sample size yielded a tighter range of projected times, but the point of diminishing returns came early at around 300 samples. This may be the natural variance that comes from sampling only a few hundred blocks. Figure 4.4 shows the projected time of the 300-sample quick test trials versus the actual time to sample. The 300 sample quick test guessed 64KiB 64% of the time and 32KiB 36% of the time. This improved on the 100 sample quick test because it guessed correctly 10% more and did not guess 16KiB even once. We also took the average of 3 100-sample results which guessed 64KiB 58% of the time and 32KiB 42% of the time. While this did not guess correctly as much as the 300 sample quick test, it was also successful in eliminating the 16KiB guess. Table 4.2 shows the standard deviation of 1,000 trials of sampling different sample sizes and by taking the average of multiple 100 sample tests. We did not find sufficient evidence to say if either sampling many blocks or taking the average of smaller tests is better, but taking the average may be more practical as it allows multiple trials to be performed until the tests passes a time threshold and also allows for significant outliers to be discarded.

CHAPTER 5:
Conclusion

We looked at sector sampling as a viable method of rapid hard drive forensics and showed how statistics can help to give useful information about a drive. Through analysis and testing, we accomplished the following:

- Designed a simple sampling strategy that can find target data in arbitrarily sized blocks.
- Defined a general equation to find the confidence for any storage device and verified that our probability model is correct through experiments.
- Found hard drive sampling characteristics.
- Developed a sector sampling program.

The power of random sampling can be understood by the simplicity of the probability model. The simplicity is a strength because the computation is easy and analysis can be performed on any drive independent of the drive content including file systems, partitions, and operating systems as long as the sector size is known. By keeping the probability model simple, the code is also simple and the scheduling algorithm is nothing more than a slightly modified uniform random number generator. By using this robust framework, our program is able to perform analysis on hard drives quickly and efficiently for any scenario or hardware environment.

5.1 Future Work

While we believe that we have used a representative sample of modern consumer grade hard drives and solid state drives in this thesis, there is room for further drive testing. Our testing results is based on only a handful of drives and it may turn out that some drives behave in completely different ways. Hard disk drives have evolved rapidly in a time span as small as a decade with advances in areas such as magnetic recording techniques and increased cache size, data sampling on newer drives may perform differently than the results found in this thesis. We also limited our test drives to the 3.5-inch form factor, which are more common in desktops and servers. The other common form factor found in most laptops, the 2.5-inch, often has a slower rotational speed which could affect sampling performance.

There is also room for trying different connection methods to the drives. We used the fastest option available to us with a drive dock connected with eSATA. This is likely the ideal setting,

but in the real world there may be cases where sub-optimal connections must be used such as USB to read from a mobile device where the storage drive cannot be easily removed. Even with eSATA, sampling may be optimized further by taking advantage of modern hard drive controller features such as Native Command Queuing (NCQ) which automatically reorders the blocks to be read to minimize the overall read time. Our program reads blocks strictly in the order determined by the scheduler, but letting the controller handle the scheduling may yield better results. A related method includes Asynchronous I/O (AIO) which may provide performance gains and is supported in recent versions of the Windows and Linux kernels.

The scheduler can be improved to be more robust using other techniques. Currently, the scheduler calculates the required sample size and samples uniformly from beginning to end. We assume that the user would not halt the program while it is running so that enough samples can be collected to achieve the desired confidence level. If target data are located near the end of the drive, then this strategy fails if the user interrupts the program before reaching the target. Even an estimated confidence cannot be given because the samples taken are biased. This can be mitigated by using a different sampling strategy. One possible strategy may be to sample only a fraction of the required amount and make multiple passes. The disadvantage of this approach is that the blocks to sample are further away, adding additional seek time to the sampling time which already has less than optimal read speed, however, we do not know how much slower making multiple passes of fewer samples would be. A more complex system may begin by dividing the drive into multiple "bands" where each band is an equally sized, non-overlapping segment of the drive. All bands together will make up the entire drive. First, a random band is selected, and only blocks in that band is randomly sampled. Only a fraction of the total required samples will be sampled in this band. Then, another band will be randomly selected and sampling is done within that band. This process of selecting a band and sampling within the band is repeated until the user stops the program. While this does not enable uniform random sampling, this makes the end of the drive just as likely to be sampled as the beginning of the drive while maintaining the speed advantage gained from sampling nearby blocks with extended seek times only to jump to the next band.

Finally, there is much more that could be done with random sampling than what we have discussed. Here we only use the idea of looking up the sampled blocks in a hash block database for known files, however, there is other research on block based forensics. There have been significant work on the file fragment classification problem where a file type must be identified given only a part of the file [15], [18], [19]. Giving the user feedback on what kind of files are on the

drive, even if no known files were found. Knowing that a drive is full of images or executables may prove useful in an investigation. Our software architecture was designed so that the block analyzer could easily be expanded in new ways. We have only scratched the surface of what our drive sampling framework is capable of. Faster and more efficient digital forensic techniques are a high priority, and as more work is done in this area, drive sampling will improve and users will benefit with more broad and accurate information.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] "Vancouver police shift blame for riot." [Online]. Available: http://www.cbc.ca/news/canada/british-columbia/story/2011/06/20/bc-vancouver-police-riot.html

[2] "Dell digital forensics solution blueprint." [Online]. Available: http://i.dell.com/sites/content/business/solutions/brochures/en/Documents/digital-forensics-blueprint.pdf

[3] "Regional computer forensics laboratory program annual report fiscal year 2011." [Online]. Available: http://www.rcfl.gov/downloads/documents/RCFL_Nat_Annual11.pdf

[4] "Defense computer forensics laboratory," accessed: 06/08/2013. [Online]. Available: http://www.dc3.mil/dcfl/

[5] G. G. Richard III and V. Roussev, "Next-generation digital forensics," *Commun. ACM*, vol. 49, no. 2, pp. 76–80, Feb. 2006. [Online]. Available: http://doi.acm.org/10.1145/1113034.1113074

[6] M. K. Rogers, J. Goldman, R. Mislan, T. Wedge, and S. Debrota, "Computer forensics field triage process model," *Journal of Digital Forensics, Security, and Law*, 2006.

[7] J. Young, K. Foster, S. Garfinkel, and K. Fairbanks, "Distinct sector hashes for target file detection," *Computer*, vol. 45, no. 12, pp. 28–35, Dec. 2012. [Online]. Available: http://dx.doi.org/10.1109/MC.2012.327

[8] S. Garfinkel and A. J. Nelson, "Fast disk analysis with random sampling." [Online]. Available: http://cenic2010.cenic.org/program/slides/2010-03-09_CENIC.pdf

[9] S. Key, "File block hash map analysis," presented at Computer Enterprise and Investigations Conference 2011 Lab. May 15-18. Orlando, Florida. [Online]. Available: www.ceicconference.com/agenda2011.htm

[10] R.-J. Mora and B. Kloet, "Digital forensic sampling," 2010. [Online]. Available: http://blogs.sans.org/computer-forensics/files/2010/03/statisticalforensictriage.pdf

[11] B. Jones, S. Pleno, and M. Wilkinson, "The use of random sampling in investigations involving child abuse material," *Digital Investigation*, vol. 9, Supplement, no. 0, pp. S99 – S107, 2012, the Proceedings of the Twelfth Annual DFRWS Conference, 12th Annual Digital Forensics Research Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287612000369

[12] "Hashing of file blocks: When exact matches are not useful." [Online]. Available: http://www.nsrl.nist.gov/Documents/aafs2008/dw-3-AAFS-2008-blocks.pdf

[13] S. Chaudhuri, G. Das, and U. Srivastava, "Effective use of block-level sampling in statistics estimation," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '04. New York, NY, USA: ACM, 2004, pp. 287–298. [Online]. Available: http://doi.acm.org/10.1145/1007568.1007602

[14] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*, 5th ed. Pacific Grove, CA, USA: Duxbury, 2000.

[15] S. Garfinkel, A. Nelson, D. White, and V. Roussev, "Using purpose-built functions and block hashes to enable small block and sub-file forensics," *Digital Investigation*, vol. 7, Supplement, no. 0, pp. S13 – S23, 2010, the Proceedings of the Tenth Annual DFRWS Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287610000307

[16] "The facts: 4k advanced format hard disks." [Online]. Available: http://www.bit-tech.net/hardware/storage/2010/04/01/the-facts-4k-advanced-format-hard-disks/1

[17] "Advanced format definitions, abbreviations, and conventions," accessed: 10/05/2012. [Online]. Available: http://www.idema.org/?page_id=2153

[18] V. Roussev and S. L. Garfinkel, "File fragment classification—the case for specialized approaches," 2009. [Online]. Available: http://simson.net/clips/academic/2009.SADFE.Fragments.pdf

[19] S. Fitzgerald, G. Mathews, C. Morris, and O. Zhulyn, "Using nlp techniques for file fragment classification," *Digital Investigation*, vol. 9, pp. S44–S49, 2012.

Referenced Authors

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia
2. Dudly Knox Library
   Naval Postgraduate School
   Monterey, California