Data Fingerprinting with Similarity Digests

Vassil Roussev

Chapter 15

DATA FINGERPRINTING WITH SIMILARITY DIGESTS

Vassil Roussev

Abstract State-of-the-art techniques for data fingerprinting have been based on randomized feature selection pioneered by Rabin in 1981. This paper proposes a new, statistical approach for selecting fingerprinting features. The approach relies on entropy estimates and a sizeable empirical study to pick out the features that are most likely to be unique to a data object and, therefore, least likely to trigger false positives. The paper also describes the implementation of a tool (`sdhash`) and the results of an evaluation study. The results demonstrate that the approach works consistently across different types of data, and its compact footprint allows for the digests of targets in excess of 1 TB to be queried in memory.

Keywords: Data fingerprinting, similarity digests, fuzzy hashing

1. Introduction

One of the most common tasks early in the investigative process is to identify known content of interest and to exclude known content that is not of interest. This is accomplished by hashing data objects (typically files) and comparing them to a database of known hashes such as NSRL [10]. The limitations of "known file filtering" become apparent when one attempts to find an embedded object (e.g., JPEG image) inside a document or an archive – file-level hashes are useless and block-level hashes barely make a difference. A similar situation arises when analyzing network traces. Ideally, one would like to quickly identify the presence of objects of interest without paying the overhead of reconstructing network connections and extracting entire files.

Another situation where current approaches fall short is the identification of "similar" objects such as different versions of a document,

library or executable. The latter is especially important when dealing with online updates that are common in modern software packages – it is impractical to expect reference databases to contain every single variation of a distribution file.

This paper attempts to address the above scenarios and to develop a practical solution that investigators can employ in the field. The method is based on the idea of identifying statistically-improbable features and using them to generate "similarity digests." Unlike cryptographic digests, which support only yes/no query results, similarity digests allow queries to be answered approximately (in the 0 to 100 range), thereby providing a measure of correlation. The method is specifically designed to deal with the problem of false positives. In prior work [15], we have shown that the root cause of false positives is that the underlying data does not contain enough unique features to be reliably identified. Therefore, the method detects and flags situations in which the query does not contain enough characteristic features for a reliable comparison and notifies the investigator. This paper describes a new tool, `sdhash`, that implements the method and presents an evaluation study that demonstrates its effectiveness.

2. Related Work

In the domain of security and authentication, a fingerprint is often synonymous with the message digest produced by a cryptographic hash function. Identical digests (or signatures) for two different objects are considered conclusive proof that the data objects themselves are identical. Digital forensic investigators make wide use of cryptographic hashes such as SHA-1 to ensure the integrity of forensic targets and to identify known content. The Achilles heal of cryptographic hashes is that they (ideally) depend on every bit of the input, which makes them inherently fragile and unsuited for similarity detection.

2.1 Rabin Fingerprinting

The idea of generating a more flexible and robust fingerprint for binary data was proposed by Rabin in 1981 [13]. Since then, considerable research has focused on developing ever more sophisticated fingerprinting techniques, but Rabin's basic idea has carried over with relatively small variations. We limit our discussion to the essential ideas. Interested readers are referred to [16] for a detailed survey of hashing and fingerprinting techniques.

Rabin's scheme is based on random polynomials and its original purpose was "to produce a very simple real-time string matching algorithm

and a procedure for securing files against unauthorized changes" [13]. A Rabin fingerprint can be viewed as a checksum with low, quantifiable collision probabilities that can be used to efficiently detect identical objects. In the 1990s, there was a renewed interest in Rabin's work in the context of finding similar objects, with an emphasis on text. Manber [8] used it in the `sif` tool for Unix to quantify similarities among text files; Brin and colleagues [2] used it in a copy-detection scheme [2]; Broder, *et al.* [3] applied it to find syntactic similarities in web pages.

The basic idea, which is referred to as anchoring, chunking or shingling, is to use a sliding Rabin fingerprint over a fixed-size window that splits data into pieces. A hash value h is computed for every window of size w. The value is divided by a constant c and the remainder is compared with another constant m. If the two values are equal (i.e., $m \equiv h \bmod c$), then the data in the window is declared as the beginning of a chunk (anchor) and the sliding window is moved one position. This process is continued until the end of the data is reached. For convenience, the value of c is typically a power of two ($c = 2^k$) and m is a fixed number between zero and $c - 1$. Once the baseline anchoring is determined, it can be used in a number of ways to select characteristic features. For example, the chunks in between anchors can be chosen as features. Alternatively, the l bytes starting at the anchor positions may be chosen as features, or multiple nested features may be employed.

Note that, while shingling schemes pick a randomized sample of features, they are deterministic, i.e., given the same inputs, produce the same features. Also, they are locally sensitive in that the determination of an anchor point depends only on the previous w bytes of input, where w could be as small as a few bytes. This property can be used to solve the fragility problem in traditional file- and block-based hashing.

Consider two versions of the same document. One document can be viewed as being derived from the other by inserting and deleting characters. For example, an HTML page can be converted to plain text by removing all the HTML tags. Clearly, this would modify a number of features, but the chunks of unformatted text would remain intact and produce some of the original features, permitting the two versions of the document to be automatically correlated. For the actual feature comparison, the hash values of the selected features are stored and used as a space-efficient representation of a "fingerprint."

2.2 Fuzzy Hashing

Kornblum [7] was among the first researchers to propose the use of a generic fuzzy hashing scheme for forensic purposes. His `ssdeep` tool

generates string hashes of up to 80 bytes that are the concatenations of 6-bit piecewise hashes. The result is treated as a string and is compared with other hashes on the basis of edit distance – a measure of how many different character insert/delete operations are necessary to transform one string into the other. While `ssdeep` has gained some popularity, the fixed-size hash it produces quickly loses granularity and works for relatively small files of similar sizes.

Roussev, *et al.* [18] proposed a similarity scheme that uses partial knowledge of the internal object structure and Bloom filters. Subsequently, they developed a Rabin-style multi-resolution scheme [19] that attempts to balance performance and accuracy by maintaining hash values at several resolutions. This approach provides similarity comparisons that are flexible and meaningful, but it requires a basic understanding of the syntactic structure of the objects, which affects its generality.

Outside the realm of digital forensics, Pucha, *et al.* [12] proposed an interesting scheme for identifying similar files in a peer-to-peer network. Their scheme focuses on large-scale similarity (e.g., the same movie in different languages) and strives to select the minimum number of features necessary for identification.

2.3 Evaluation of Fingerprinting Approaches

Rabin's randomized model of fingerprinting works well on average, but suffers from problems related to coverage and false positive rates. Both these problems can be traced to the fact that the underlying data can have significant variations in information content. As a result, feature size/distribution can vary widely, which makes the fingerprint coverage highly skewed. Similarly, low-entropy features produce abnormally high false positive rates that render the fingerprint an unreliable basis for comparison.

Research in the area of payload attribution has produced more sophisticated versions of Rabin fingerprinting that seek to increase coverage (see, e.g., [5, 11, 21, 22]). These techniques manage the feature selection process so that big gaps or clusters are avoided. However, they do not consider false positives due to weak (non-identifying) features. It is important to recognize that coverage and false positives are fundamentally connected; selecting weak features to improve coverage directly increases the risk of false positive results.

3. Non-Rabin Fingerprinting

The general idea behind any similarity scheme is to select multiple characteristic (invariant) features from the data object and compare

them with features selected from other objects. The collection of features can be viewed as a digital fingerprint or signature. A feature can be defined at multiple levels of abstraction, where the higher levels require more specialized processing. For the purposes of our work, we define a feature very simply as a bit sequence. In other words, we view binary data as a syntactic entity and make no attempt to parse or interpret it. This approach has obvious limitations, but is motivated by the need to develop generic, high-throughput methods that can rapidly filter large amounts of data. The expectation is that the approach would be complemented in the later stages by higher-order analysis of the filtered subset.

Our work has three main contributions. First, it presents a new feature selection scheme that selects statistically-improbable features as opposed to the randomized approach of Rabin schemes; this provides more reliable identification of characteristic features and offers even more coverage. Second, it incorporates a new approach that allows for the generic screening of inherently weak (non-identifying) features based on entropy measures; as our evaluation shows, this leads to a significant reduction in false positives. Third, it defines a new, scalable measure of similarity based on the statistical properties of Bloom filters; the measure supports the efficient comparison of objects of arbitrary sizes.

3.1 Selecting Statistically-Improbable Features

The statistically-improbable feature selection process is somewhat similar to Amazon's use of statistically-improbable phrases to characterize publications. The goal is to pick object features that are least likely to occur in other data objects by chance. The challenge is that this approach has to work for binary data (not just text) and, therefore, it is not possible to parse or interpret the data.

In this work, we consider features of size $B = 64$ bytes, which we have found to be a suitable granularity for identifying objects in disk blocks and network packets. However, there are no conceptual or implementation differences in using a different feature size. Note that a fundamental trade-off exists: the smaller the features, the higher granularity, the larger the digests and the more processing that is involved.

In all cases, the feature selection process involves the following steps:

- **Initialization:** The entropy score H_{norm}, precedence rank R_{prec} and popularity score R_{pop} are initialized to zero.
- $\boldsymbol{H_{norm}}$ **Calculation:** The Shannon entropy is first computed for every feature (B-byte sequence): $H = -\sum_{i=0}^{255} P(X_i) \log P(X_i)$, where $P(X_i)$ is the empirical probability of encountering ASCII

R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				1														
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				2														
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				3														
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4														
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1													
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1						
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		1				
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		2				
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		3				
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		4				
R_{prec}	882	866	852	834	834	852	866	866	875	882	859	849	872	842	849	877	889	880
R_{pop}				4	1							1		5				

Figure 1. Example R_{pop} calculation.

code i. Then, the entropy score is computed as $H_{norm} = \lfloor 1000 \times H / \log_2 B \rfloor$.

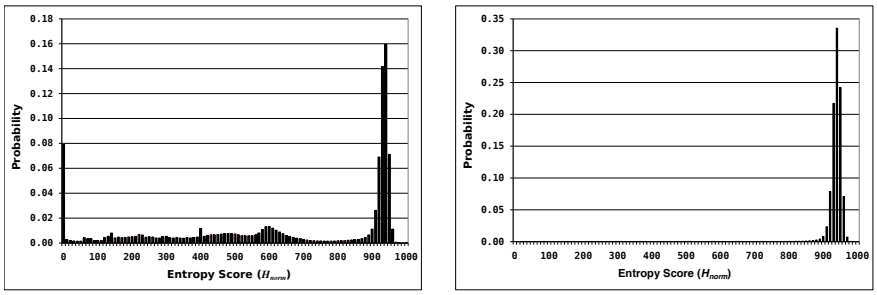
- ***R_{prec}* Calculation:** The precedence rank R_{prec} value is obtained by mapping the entropy score H_{norm} based on empirical observations.

- ***R_{pop}* Calculation:** For every sliding window of W consecutive features, the leftmost feature with the lowest precedence rank R_{prec} is identified. The popularity score R_{pop} of the identified feature is incremented by one.

- **Feature Selection:** Features with popularity rank $R_{pop} >= t$, where t is a threshold parameter, are selected.

Figure 1 illustrates the R_{pop} calculation and feature selection steps. A snippet of 18 R_{prec} numbers from an actual computation is used; a window $W = 8$ is used for the R_{pop} calculation. Assuming a threshold $t = 4$ and feature size $B = 64$, two features are selected to represent an 82-byte piece of data.

The principal observation is that the vast majority of the popularity scores are zero or one; this is a very typical result. For an intuitive



(a) `doc` set. (b) `gz` set.

Figure 2. Example entropy score distributions.

explanation, consider the entropy score histogram of the `gz` compressed data set in Figure 2(b). As expected, the overall entropy is close to the maximum, but the entropy of individual features is, in fact, normally distributed. This means that, although the entropy measures of the neighboring features are highly correlated, every once in a while, an entropy score that is less frequent is encountered (by chance). By extension, the feature itself is likely to occur less frequently. Based on our statistics, this manifests itself as a local minimum in the precedence rank, which ultimately results in a higher popularity score.

The same logic applies to other types of data; a more detailed account of our empirical observations is presented in [15]. In general, the feature selection procedure described above works on any type of data for which a reasonable (not necessarily perfect) approximation of the feature entropy distribution is available.

3.2 Filtering Weak Features

Much of the impetus to reduce the number of weak fingerprint features comes from observations of the feature entropy distribution in `doc` data (Figure 2(a)). As can be seen, full 8% of the data has zero entropy due to large blocks of repeated characters (mostly zeroes). Such a feature yields a raw false positive rate approaching 100%, meaning that the probability that the feature will not be unique to a specific data object is almost 100%.

This problem is by no means constrained to `doc` data or to zero-entropy features. Text data exhibits similar properties with raw false positive rates staying above 10% for entropy scores up to 180 [15]. At the same time, the weak features account for less than 2% of the total number of features. Eliminating weak features from consideration can

reduce false positive rates with minimal effect on coverage and likely no impact on real-world applications.

In developing our `sdhash` tool, we used a 600 MB sample set of mixed, randomly-obtained files to derive the entropy score distribution and the entropy-to-precedence mapping table. During the filtering process, all features with entropy scores of 100 or below, and those exceeding 990 were unconditionally dropped from consideration. The latter decision is based on the observation that features with near-maximum entropy tend to be tables whose content is common across many files. For example, Huffman and quantization tables in JPEG headers can have very high entropy but are poor characteristic features.

3.3 Generating Similarity Digests

After the object features have been selected and filtered, the next step is to build the fingerprint representation. For this purpose, we employ a sequence of Bloom filters as in our earlier multi-resolution similarity hashing scheme [19]. A Bloom filter [1, 4] is a bit vector used for space-efficient set representation. The price paid for the space savings is the possibility that membership queries may return false positive results. To insert an element, it is hashed with k different hash functions; the results are treated as addresses inside the filter and the corresponding bits are set to one. Membership queries are handled in a similar manner; however, instead of setting the bits, they are tested to see if they are set – if all k bits are set to one, then the answer is "yes," otherwise it is "no." It is possible that the combination of bits checked during a query is set by chance, which results in a false positive. The probability of false positives can be quantified and controlled by limiting the number of elements inserted into the filter.

In our implementation, selected features are hashed using SHA-1 and the result is split into five sub-hashes, which are used as independent hash functions to insert the feature into the filter. The implementation uses 256-byte filters with a maximum of 128 elements per filter. After a filter reaches capacity, a new filter is created and the process is repeated until the entire object is represented.

One subtle detail is that before a feature is inserted, the filter is queried for the feature; if the feature is already present, the count for the number of elements inserted is not increased. This mechanism prevents the same feature from being counted multiple times, which reduces the false positive rate by forcing the inclusion of non-duplicate features; the accuracy of the similarity estimate is also increased.

3.4 Comparing Digests

The basic building block for digest comparison is the comparison of two Bloom filters. In general, the overlap between two compatible filters is a measure of the overlap between the sets they represent – the number of bits in common grows linearly with the amount of set overlap.

Consider two Bloom filters f_1 and f_2 of size m bits containing n_1 and n_2 elements ($n_1 \leq n_2$), respectively, and n_{12} elements in common. Let k be the number of hash functions used and e_1, e_2 and e_{12} be the number of bits set to one in f_1, f_2 and $f_1 \cap f_2$, respectively. Using classical Bloom filter analysis [4], the estimate of the number of expected common bits set to one is:

$$E_{12} = m\left(1 - p^{ks_1} - p^{ks_2} + p^{k(s_1+s_2-s_{12})}\right),\ p = 1 - 1/m$$

Furthermore, the estimates of the maximum and minimum number of possible overlapping bits due to chance are:

$$E_{max} = \min(n_1, n_2);\quad E_{min} = m\left(1 - p^{ks_1} - p^{ks_2} + p^{k(s_1+s_2)}\right)$$

Next, we define a cutoff point C below which all bit matches are assumed to be due to chance:

$$C = \alpha(E_{max} - E_{min}) + E_{min}$$

Thus, the similarity filter score SF_{score} of the two filters is defined as:

$$SF_{score}(f_1, f_2) = \begin{cases} -1 & \text{if } n_1 < N_{min} \\ 0 & \text{if } e_{12} \leq C \\ \left\lfloor 100\frac{e_{12}-C}{E_{max}-C} \right\rfloor & \text{otherwise} \end{cases}$$

where N_{min} is the minimum number of elements required to compute a meaningful score. Our implementation uses an experimentally-derived value of $N_{min} = 6$.

Given two digests SD_1 and SD_2 consisting of Bloom filters $f_1^1, \ldots, f_s^1$ and $f_1^2, \ldots, f_t^2$, respectively ($s \leq t$), the similarity digest score is formally defined as:

$$SD_{score}(SD_1, SD_2) = \frac{1}{s}\sum_{i=1}^{s} \max_{1 \leq j \leq t} SF_{score}(f_i^1, f_j^2)$$

Informally, the first filter from the shorter digest (SD_1) is compared with every filter in the second digest (SD_2) and the maximum similarity score is selected. This procedure is repeated for the remaining filters in SD_1 and the results are averaged to produce a single composite score.

The rationale behind this calculation that a constituent Bloom filter represents all the features in a continuous chunk of the original data. Thus, by comparing two filters, chunks of the source data are compared implicitly. (In fact, it is possible to store (at a modest cost) the exact range that each of the filters covers in order to facilitate follow-up work.) Thus, the size of the filters becomes a critical design decision – larger filters speed up comparisons while smaller filters provide more specificity.

The interpretation of the scores is discussed in Section 5. At this point, however, we note that the parameters, including $\alpha = 0.3$, have been calibrated experimentally so that the comparison of the fingerprints of unrelated random data consistently yields a score of zero.

4. Implementation

We have implemented the fingerprinting method in the `sdhash` tool, which is available at [17]. The usage format is `sdhash [options] {<source_file(s)> | <digest_files>}`. Users may pick one digest generation/storage option and one digest comparison option as follows:

- **Option `-g`:** This option treats the files in the list as original (source) data. For every file name `pathname/file.ext`, a corresponding `pathname/file.ext.sdbf` file containing the SD fingerprint is generated. No fingerprint comparisons are performed.
- **Option `-c`:** This option treats the files in the list as digest data and comparisons are performed.
- **Option `-f`:** This option is the combination of options `-g` and `-c`. The digest files are generated and compared.
- **Option `-m`:** This default option is the same as the `-f` option except that no digest files are created as a side effect.
- **Option `-p`:** This option causes the header(s) of fingerprint file(s) to be printed. The following example illustrates its use.

```
> ./sdhash -p 100M.doc.sdbf
100M.doc.sdbf:  bf_count: 10858, bf_size: 256,
hash_count: 5, mask:  7ff, max: 128, count: 77
```

In the example, the fingerprint consists of a sequence of 10,858 256-byte Bloom filters. Five (32-bit) sub-hashes are generated from the base SHA-1 hash and based on the bit mask, the 11 least-significant

bits are used to address bits within each filter. Each filter encodes 128 features, except for the last filter, which has 77 features. The total number of features is $10{,}857 \times 128 + 77 = 1{,}389{,}773$ features.

- **Option `-2`:** This default option specifies that for n source files or digests, a total of $n-1$ pairwise comparisons should be performed: $< \#1, \#2 >, < \#1, \#3 >, \ldots, < \#1, \#n >$.
- **Option `-n`:** This option specifies that for n source files or digests, all unique pairs must be compared: $< \#1, \#2 >, < \#1, \#3 >, \ldots, < \#2, \#3 >, < \#2, \#n >, \ldots, < \#2, \#n >, \ldots, < \#n - 1, \#n >$.

The `sdhash` output consists of three columns. The first two columns list the files that are compared; the third column gives the corresponding SD scores.

5. Cross-Set Fragment Detection Experiments

This section discusses a "needle in a haystack" fragment detection scenario. Given a relatively small snippet of data such as a disk block or network packet ("needle"), the goal is to determine whether or not parts of it are present in the large set of data ("haystack").

Based on the scenario, the `sdhash` parameters were tuned to work for fragments in the 1024-byte to 4096-byte range. We also studied the boundary case of a 512-byte fragment to understand the behavior outside the design zone. As the results show, the method degrades gracefully; however, the accuracy of the results inherently drops because it becomes more difficult to find 64-byte characteristic features. For reasons of space, the scenario involving the detection of similar objects is not presented in this paper; it will be the subject of a future article.

5.1 Experimental Setup

This section presents the data, parameters and metrics used in the experimental evaluation.

Data Six sample 100 MB document sets from the NPS Corpus [6] were used in the experiments: (i) Microsoft Word documents (`doc`); (ii) HTML documents (`html`); (iii) JPEG images (`jpg`); (iv) Adobe PDF documents (`pdf`); (v) Plain text documents (`txt`); and (vi) Microsoft Excel spreadsheets (`xls`). In addition, a seventh 100 MB pseudorandom data set (`rnd`) was obtained from `/dev/urandom`. The `rnd` set represents a calibration benchmark because its content is unique and no features selected from it appear in the other sets. It is infeasible to provide similar guarantees for the remaining sets. The results obtained for the `rnd` set

essentially correspond to a best-case scenario for what can be achieved in practice. Therefore, it is used as a benchmark to evaluate how close the other results come to the best case.

Parameters Based on the primary scenario and our preliminary evaluation of SD hashes, the following parameters were chosen for the fingerprints:

- **Feature Selection:** The feature size $B = 64$ bytes, i.e., all possible sliding 64-byte sequences in an object were considered. The window size $W = 64$ bytes, i.e., one representative feature was selected from every 64 consecutive features based on the entropy measures defined earlier. The threshold $t = 16$, i.e., only the representative features selected in at least 16 consecutive windows were considered.

- **Bloom Filters:** Upon selection, each feature was hashed using SHA-1 and the resulting 160 bits of the hash were split into five sub-hashes of 32 bits, each of them treated as an independent hash function. The actual similarity digest is a sequence of 256-byte Bloom filters with 128 elements (features) per filter. Thus, the expected false positive rate of the Bloom filter is 0.0014 for an individual membership query.

- **Fragment Size:** Four different fragment sizes of 512, 1024, 2048 and 4096 bytes were used to evaluate the behavior of the similarity measure in the range of the fragment sizes of interest.

Evaluation Metrics We considered three basic measurements: detection rates, non-classification rates and misclassification rates. The first step was to generate sample fragment sets for every combination of fragment size and data set. These were obtained by picking random file-offset combinations and extracting a fragment of the appropriate size. This gave rise to 28 ($= 4 \times 7$) fragment sets, each with 10,000 samples. SD fingerprints were generated for each reference set and sample, which were then compared depending on the scenario:

- **Detection Rate:** This metric assesses the likelihood that the SD hash method correctly attributes a sample set to its source. In other words, the fragment fingerprints are compared to the source set fingerprint for every sample set. Note that the classification results depend on the choice of minimum score threshold value – the higher the value, the lower the detection rate.

- **Non-Classification Rate:** This metric assesses the likelihood that the SD hash method rejects a fragment set as not containing enough characteristic features for reliable classification. This is the equivalent an "I don't know" answer to a similarity query. Note that this rate is solely a function of the fragment content and does not depend on the comparison target; thus, the sample can be identified as being weak before comparisons are performed. Intuitively, the non-classification rate can be expected to be very close to zero for high-entropy (random, compressed or encrypted) data. Furthermore, an elevated non-classification rate is an implicit indicator that the comparison results are less reliable due to small fragment size and/or low sample entropy.

- **Misclassification Rate:** Recall that an SD hash comparison produces a score in the 0 to 100 range (-1 for non-classification). Given a random sample that is present in the reference file, it is not guaranteed that the returned result will be 100. In particular, it is quite likely (depending on alignment) that the features selected from the sample will spread into more than one Bloom filter in the fingerprint of the source.

  In practice, this implies that the selected threshold should balance the probabilities of false negatives and false positives. To cumulatively capture these errors, the false negative and false positive rates are summed to produce the misclassification rate. Evidently, the misclassification rate is a function of the threshold value chosen to separate false negatives and false positives. The closer the threshold is to zero, the higher the probability for false positives; conversely, the closer the threshold is to 100, the higher the probability of false negatives.

  Ideally, there is a range of SD scores for which the misclassification rate is zero, so a safe threshold could be picked in the range. In some cases, such perfect ranges do, in fact, exist. Most of the time, however, the goal is to minimize the misclassification rate. To obtain the misclassification rate estimates, the scores of 10,000 true samples from a source set were compared with those of 10,000 samples taken from each of the other sets. Then, the misclassification rate was identified for every possible value of the threshold (1-99); the best choice showed up as a global minimum in the plot.

5.2 Experimental Results

Space constraints prevent us from presenting the complete set of results (these will be published separately as a technical report). Fortu-

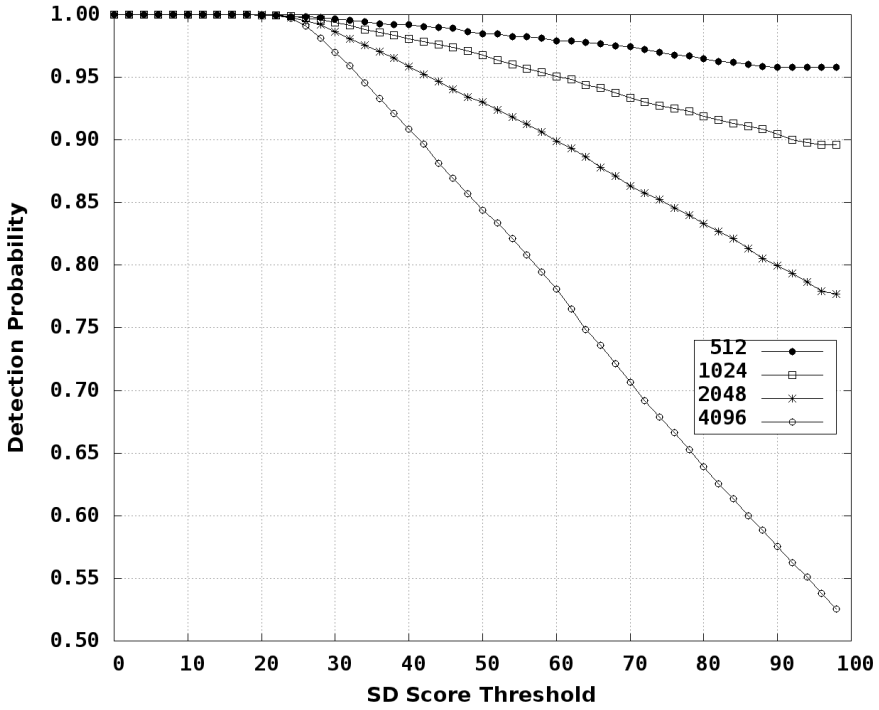


Figure 3. Detection rates for the `txt` reference set.

nately, the observed behavior was consistent for all the data sets. As a result, we only present the detailed results for the `txt` set.

Detection Rates Figure 3 presents the the `txt` detection performance (y-axis) as a function of the SD score threshold (x-axis) and the fragment size (512, 1024, 2048 and 4096 bytes). The first observation is that a threshold score of up to 22 yields near-perfect (0.999+) detection rates for all fragment sizes. Detection rates drop approximately linearly beyond this value, with rates for larger fragments dropping faster than those for smaller fragments. The latter result is expected because, as the fragment size grows, so does the probability that the fragment features selected will end up in multiple Bloom filters in the digest of the original file. This is best illustrated by the rightmost points of the curves, which represent the fraction of true positives that generate a score of 100.

Non-Classification Rates Table 1 shows the non-classification rates for various test sets and fragment sizes. The decision to refuse classification is based on the requirement that a fragment must contain a

Table 1. Non-classification rates for various test sets and fragment sizes.

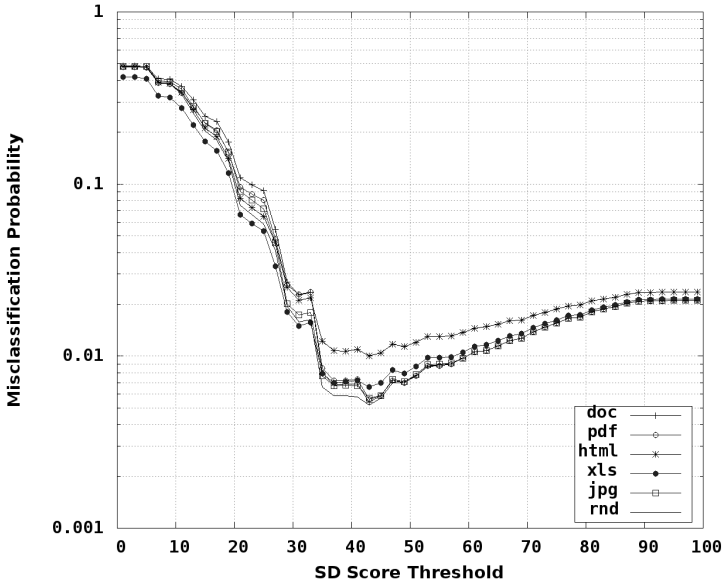
Test Set	Fragment Size 512	1024	2048	4096
doc	0.2383	0.1291	0.0764	0.0435
html	0.0995	0.0059	0.0025	0.0008
jpg	0.0281	0.0089	0.0045	0.0033
pdf	0.0533	0.0198	0.0163	0.0157
rnd	0.0166	0.0000	0.0000	0.0000
txt	0.0860	0.0192	0.0060	0.0031
xls	0.0706	0.0113	0.0058	0.0024

minimum of six unique selected features for the comparison to proceed. The cutoff point was obtained empirically based on the observation that misclassification rates escalate rapidly below this threshold without enhancing detection. The non-classification rates for 512-byte fragments are significantly higher than those for larger fragments. Note that the non-classification rate is zero for 1024-, 2048- and 4096-byte `rnd` fragments.

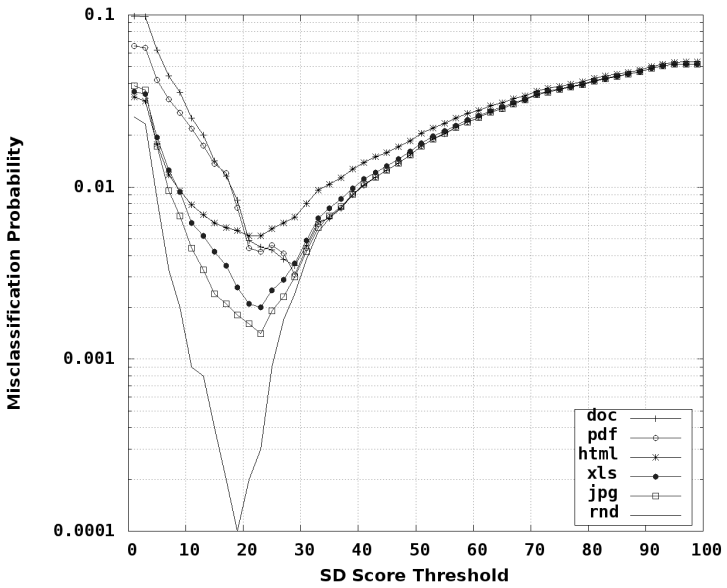
An important result is that the `doc` set exhibits high non-classification rates for all four fragment sets. This is not entirely surprising given our earlier survey of feature entropy distributions for different sets. Specifically, we showed that 8% of all potential features in the `doc` set have zero entropy (Figure 2(a)). The actual problem areas expand beyond this to encompass neighboring areas where entropy is too low.

It is important to emphasize that our design philosophy is to reject weakly-identifiable data and not classify them. This decision is justified because reliable estimates of error rates cannot be obtained without excluding weak data.

Misclassification Rates Figure 4 presents the misclassification results for all 512- and 1024-byte fragments with respect to the `txt` reference set as a function of the chosen SD score threshold values. Since the y-axis uses a logarithmic scale, the zero values are replaced with 0.0001 to enable visualization. Figure 4(a) is representative of the behavior observed across all experiments with 512-byte fragments. On the other hand, Figure 4(b) is representative of all the experiments involving 1024-, 2048- and 4096-byte fragments (larger fragments produce marginally better results). This is welcome news because we seek classification threshold values that work consistently across all sets. The observed consistency demonstrates that the method is stable and works well across the spectrum of data.



(a) 512-byte fragments.



(b) 1024-byte fragments.

Figure 4. Misclassification rates for the `txt` reference set.

Table 2. Misclassification rates for various test sets and fragment sizes.

Test Set	Fragment Size							
	512		1024		2048		4096	
	min	max	min	max	min	max	min	max
rnd	0.0050	0.0100	0.0006	0.0050	.0003	.0040	.0003	0.0040
doc	0.0050	0.0100	0.0002	0.0050	.0003	.0050	.0002	0.0040
html	0.0060	0.0100	0.0003	0.0050	.0005	.0055	.0000	0.0035
jpg	0.0050	0.0120	0.0005	0.0055	.0002	.0050	.0003	0.0040
pdf	0.0060	0.0120	0.0002	0.0050	.0002	.0050	.0002	0.0040
txt	0.0050	0.0100	0.0002	0.0055	.0002	.0050	.0000	0.0035
xls	0.0060	0.0130	0.0002	0.0055	.0002	.0050	.0150	0.0180

Figure 4(a) demonstrates that there is room for varying the optimization criterion, and that a case can be made for a number of possible threshold values in the 37-49 range. Upon inspecting all the graphs, 43 emerges as the best candidate because it consistently achieves near-optimal results across all the 512-byte fragment experiments. A value of 21 yields the most consistent results for the 1024-, 2048- and 4096-byte fragments.

Table 2 summarizes the results for all the test sets using the chosen threshold values. Each cell provides the minimum or maximum misclassification rate observed for a fragment of a particular size. Note that the top row of the table (`rnd`) is the reference best-case scenario, and the other six sets produce very similar results.

Storage and Throughput On the average, storing a similarity digest along with the chosen parameters requires about 2.6% of the original source data. However, it is possible to shrink the on-disk representation down to 2.4% using standard zip compression. This strikes a good balance between accuracy and compactness of representation – a commodity server costing $5,000 can be equipped with 32 to 48 GB RAM, which would support the in-memory representation of 1.25 to 1.75 TB of data.

The current implementation is capable of generating SD hashes at the rate of approximately 30 MB/sec/core on a modern processor. Thus, the SD fingerprinting method can be applied during the imaging process and would be able to identify artifacts during target acquisition.

6. Conclusions

Our method for generating data fingerprints based on statistically-improbable features engages a generic entropy-based scheme to efficiently

select features from binary data. Prior work relies on randomized feature selection, which provides uneven coverage and produces relatively high false positives for low-entropy data. The method enables features with low information content to be filtered, thereby reducing false positives.

Experimental evaluations of the performance of the `sdhash` implementation on small (1 to 4 KB) data fragments from six common file types demonstrate the robustness of the method. The error rate as represented by misclassified fragments (including false positives and false negatives) does not exceed 0.0055, implying a correct classification rate of 0.9945. The success of the technique is due to the fact that the algorithm flags fragments that do not contain enough identifying features. The space requirements for the generated similarity digests do not exceed 2.6% of the source data, which makes it possible to maintain digests of images up to 1.5 TB in memory.

Our future research will refine and optimize the tool, and perform tests on large forensic images using digests generated from the NSRL set. We also plan to explore the capabilities of the tool by tuning its parameters for smaller and larger features. Along the same lines, we plan to add a preprocessing step that will recognize common header features that tend to produce false positives.

References

[1] B. Bloom, Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM*, vol. 13(7), pp. 422–426, 1970.

[2] S. Brin, J. Davis and H. Garcia-Molina, Copy detection mechanisms for digital documents, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 398–409, 1995.

[3] A. Broder, S. Glassman, M. Manasse and G. Zweig, Syntactic clustering of the web, *Computer Networks and ISDN Systems*, vol. 29(8-13), pp. 1157–1166, 1997.

[4] A. Broder and M. Mitzenmacher, Network applications of Bloom filters: A survey, *Internet Mathematics*, vol. 1(4), pp. 485–509, 2005.

[5] C. Cho, S. Lee, C. Tan and Y. Tan, Network forensics on packet fingerprints, *Proceedings of the Twenty-First IFIP Information Security Conference*, pp. 401–412, 2006.

[6] Digital Corpora, NPS Corpus (digitalcorpora.org/corpora/disk-images).

[7] J. Kornblum, Identifying almost identical files using context triggered piecewise hashing, *Digital Investigation*, vol. 3(S1), pp. S91–S97, 2006.

[8] U. Manber, Finding similar files in a large file system, *Proceedings of the USENIX Winter Technical Conference*, pp. 1–10, 1994.

[9] M. Mitzenmacher, Compressed Bloom filters, *IEEE/ACM Transactions on Networks*, vol. 10(5), pp. 604–612, 2002.

[10] National Institute of Standards and Technology, National Software Reference Library, Gaithersburg, Maryland (www.nsrl.nist.gov).

[11] M. Ponec, P. Giura, H. Bronnimann and J. Wein, Highly efficient techniques for network forensics, *Proceedings of the Fourteenth ACM Conference on Computer and Communications Security*, pp. 150–160, 2007.

[12] H. Pucha, D. Andersen and M. Kaminsky, Exploiting similarity for multi-source downloads using file handprints, *Proceedings of the Fourth USENIX Symposium on Networked Systems Design and Implementation*, pp. 15–28, 2007.

[13] M. Rabin, Fingerprinting by Random Polynomials, Technical Report TR1581, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, 1981.

[14] S. Rhea, K. Liang and E. Brewer, Value-based web caching, *Proceedings of the Twelfth International World Wide Web Conference*, pp. 619–628, 2003.

[15] V. Roussev, Building a better similarity trap with statistically improbable features, *Proceedings of the Forty-Second Hawaii International Conference on System Sciences*, pp. 1–10, 2009.

[16] V. Roussev, Hashing and data fingerprinting in digital forensics, *IEEE Security and Privacy*, vol. 7(2), pp. 49–55, 2009.

[17] V. Roussev, `sdhash`, New Orleans, Louisiana (roussev.net/sdhash).

[18] V. Roussev, Y. Chen, T. Bourg and G. Richard, `md5bloom`: Forensic filesystem hashing revisited, *Digital Investigation*, vol. 3(S), pp. S82–S90, 2006.

[19] V. Roussev, G. Richard and L. Marziale, Multi-resolution similarity hashing, *Digital Investigation*, vol. 4(S), pp. S105–S113, 2007.

[20] V. Roussev, G. Richard and L. Marziale, Class-aware similarity hashing for data classification, in *Research Advances in Digital Forensics IV*, I. Ray and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 101–113, 2008.

[21] S. Schleimer, D. Wilkerson and A. Aiken, Winnowing: Local algorithms for document fingerprinting, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 76–85, 2003.

[22] K. Shanmugasundaram, H. Bronnimann and N. Memon, Payload attribution via hierarchical Bloom filters, *Proceedings of the Eleventh ACM Conference on Computer and Communications Security*, pp. 31–41, 2004.