Finding Content in File-Sharing Networks When You Can't Even Spell

Matei A. Zaharia[†], Amit Chandel[⋆], Stefan Saroiu[⋆], and Srinivasan Keshav[†]
[†]University of Waterloo and [⋆]University of Toronto

Abstract: *The query success rate in current file-sharing systems is low, for example, only 7-10% in Gnutella. An often-overlooked cause for this low recall is simply that keywords in queries and document descriptions are misspelled. Although many sophisticated approximate matching techniques have been developed by the Information Retrieval community, to our knowledge, they have not been used in popular P2P systems.*

We propose two approaches to improving query recall in file-sharing systems. For unstructured P2P networks, we show that "q-gram"-based approaches nearly double recall with little loss in precision. Unfortunately, such techniques cannot be used in structured P2P networks. Instead, we propose a simple alternative: encoding keywords using Soundex, a century-old phonetic algorithm for indexing names by their sound.

We evaluate both approaches on a trace of all queries and files in Gnutella over a period of a month. We find that misspellings are common in this trace, with 20% of file descriptions and 25% of queries containing at least one spelling error. We find that our approaches improve recall by 20-88% with little loss in precision when compared to a standard prefix-match approach. Given the endemic nature of misspellings, our findings suggest that techniques such as those described here ought to be part of any file-sharing system.

1 Introduction

The query success rate in file-sharing P2P networks is low. For example, only 7-10% of queries in Gnutella are successful. Although many factors contribute to this low recall rate, we believe that an often overlooked cause is simply that keywords in queries and document metadata are misspelled. Our findings show that 20% of file descriptions and 25% of queries in Gnutella are spelled incorrectly. The focus of our work is to improve the query success (i.e. recall) rate in such networks despite endemic misspellings.

There are many reasons why people spell incorrectly, such as errors in typing, and the inconsistency between spelling and pronunciation in the English language. Given these deep-seated root causes of misspelling, we are likely to see misspelled keywords in both queries and metadata for a long time to come.

Handling incorrect spelling is challenging. First, there are many ways in which words can be misspelled. Over the course of three months, Google users have misspelled "Britney Spears" in 592 different ways [1]. Second, given an incorrect spelling, it is sometimes hard to infer the intended word. For example, it is unclear whether "remeinder" is a misspelled version of the word "reminder" or the word "remainder". Third, there could be alternate correct spellings for the same word. While in the U.S. it is correct to spell "center", in Canada and the UK the correct spelling is "centre". Finally, a misspelled word may match a correctly spelled (but different) word – a misspelling of "eluded" is the rare word "elided". In this case, deciphering the user's intention is challenging, if not impossible.

We propose two approaches to dealing with misspellings suitable for unstructured and structured P2P networks, respectively. The first approach, for unstructured P2P-based file-sharing systems, is based on approximate string matching algorithms developed for information retrieval and computational biology (for a survey, see [3].) These algorithms can match two strings while allowing for a number of "errors", including misspellings. These techniques operate on "q-grams" of a keyword where a q-gram of a word is a substring of length q. For a given query, these techniques typically compute a relevance score for each of the document in the system. The document with the highest score is returned as the answer to the query. Since these techniques rely on accessing a large fraction of the corpus of documents in the system, such as all files sharing q-grams with the query, they are very costly for DHTs.

Instead, we propose a different and much simpler approach for DHTs: using a coding algorithm that suppresses spelling variations of keywords, such as Soundex [4]. Soundex maps different spellings of the same word to a single code. In this approach, inverted indices are maintained on Soundexed keywords. Each node uses the coding algorithm to convert its own content metadata to codes before inserting them into the index and convert its own queries before sending them to other nodes in the system. While less effective than sophisticated approximate matching techniques and prone to a high false-positive rate, this approach is simple, fast, and can be easily and incrementally deployed over any DHT. Moreover, we find that false-positives are easily eliminated by post-processing query results with a simple edit-distance-based filter.

Contributions We measure the prevalence of spelling errors in traces of several file-sharing systems: our own recent Gnutella and YouTube traces, an older Napster trace, and one downloaded from IMDB. We find that between 10% and 47% of user-entered queries and file metadata are misspelled in these traces.

We present a Gnutella trace that captures all queries and all filenames of 22,457 peers over the course of one month. This gives us a baseline for a typical, realistic, file-sharing workload. Using this trace, we measure the recall rates of several matching algorithms[1]. In particular, we study Soundex and two state-of-the-art

[1]Our analysis assumes that all queries are conjunctive. While today's file-sharing networks support disjunctive queries, we believe that a conjunctive query model leads to more meaningful query recall estimates.

1. Retain the first letter of the string.
2. Remove all occurrences of the following letters, unless it is the first letter: a, e, h, i, o, u, w, y.
3. Assign numbers to remaining letters (after the first) as follows:
   - b, f, p, v = 1
   - c, g, j, k, q, s, x, z = 2
   - d, t = 3
   - l = 4
   - m, n = 5
   - r = 6
4. If two or more letters with the same number were adjacent in the original name, or adjacent except for any intervening h and w, then omit all but the first.
5. Return first 4 characters, right-padding with 0's when fewer than 4.

Figure 1. **The Soundex Algorithm.** *Soundex is a phonetic algorithm for indexing names by their sound when pronounced in English. It maps words to four-byte codes. For example, both words "Britney" and "Britny" are mapped to "B635", whereas "Brian" is mapped to "B650".*

approximate-matching algorithms: one based on TF-IDF weights [8] and one based on using the Jaccard similarity coefficient [7]. We find that these matching algorithms increase today's query success rates by 20%, 51%, 88% for Soundex, Jaccard's coefficient, and TF-IDF weights, respectively. Overall, we find that both approximate matching techniques greatly improve recall and are ideal for use in unstructured P2P networks. Similarly, we find that Soundex is simple, fast, and fairly effective, and leads to significant increases in DHT's recall rate. All three techniques achieve higher recall with little loss in precision (i.e. the number of relevant results to a query).

Roadmap The rest of the paper is organized as follows. In Section 2, we present a brief background of matching algorithms and we summarize the related work. In Section 3, we discuss our approach for handling incorrectly spelled words in file-sharing systems. Section 4 presents the methodology of our experiments, and Section 5 evaluates several algorithms handling misspellings. Finally, we conclude in Section 6.

2 Background and Related Work

This section presents a brief description of Soundex followed by an outline of two approximate matching algorithms from the information retrieval literature.

Soundex is nearly a century-old algorithm [4], with many variants in wide use. The most common variant of the algorithm uses four-byte codes – one letter followed by three digits. The letter is the first letter of the keyword or tag, and the numbers correspond roughly to the remaining consonants. By assigning the same number to similar sounding consonants, errors are suppressed. For example, 'd' and 't' are both encoded as '3', so the tag "rate" would match the keyword "rade" when they are both encoded with Soundex. Extending the Soundex algorithm to use codes longer than four bytes is common in practice [4]. Figure 1 describes the most-commonly used version of Soundex.

Q-grams are widely used to build approximate-string join capabilities on top of commercial databases [2]. The q-grams of a word are all its substrings of length q. For example, the 3-grams corresponding to the word 'hello' are 'hel', 'ell', and 'llo'. If two words share most of their q-grams, it is likely that one of them is a misspelled version of the other. For example, the word 'helllo' has 3-grams 'hel', 'ell', 'lll', and 'llo', suggesting a misspelling of 'hello'. We compared Soundex with two approximate matching algorithms over q-grams, TF-IDF and Jaccard, which are described next.

TF-IDF (term frequency - inverse document frequency) [8] measures the importance of a word in a document relative to a corpus. In a given document, a word's importance increases proportional to the number of times it appears in the document and inversely proportional to its frequency in the corpus. In this way, TF-IDF distinguishes between more important and less important words. A word occurring frequently in a particular document but rarely elsewhere in the corpus is thought to be "important" for that document. In our context, the TF-IDF weight of a 3-gram appearing in a file metadata (or in a query) is proportional to how frequently the 3-gram appears in the file's metadata (i.e., "term-frequency") and inversely proportional to how frequently the 3-gram appears in the entire corpus of all files' metadata (i.e., "document-frequency"). The TF-IDF score of a query relative to a file is the dot product of all TF-IDF weights of all 3-grams shared between the query and the file's metadata.

The Jaccard similarity coefficient [7] compares the similarity of two sets and is defined as the size of the intersection divided by the size of the union of the sets: $J(A, B) = |A \cap B|/|A \cup B|$. Two identical sets have a Jaccard's coefficient of 1, and two completely dissimilar sets have a coefficient of 0. In our context, we use the Jaccard index to measure the similarity between the q-gram set of a query and the q-gram sets for the metadata corresponding to each file.

Our experiments use 3-grams to compute the TF-IDF weights and the Jaccard's coefficients between queries and file metadata. The most relevant answer to a query is the document whose tags had the highest score according to TF-IDF or to Jaccard's coefficient.

3 Our Approach

We propose using an approximate matching algorithm, such as the ones based on *q-grams*, to handle spelling errors in unstructured P2P systems. Although they can be computationally expensive and some require access to the entire corpus of data, our results show that they greatly improve recall for today's file-sharing workloads.

Unfortunately, approximate matching is not suitable for DHTs as such techniques typically require accessing all files whose metadata share at least one q-gram with the query. Such operations are very costly for DHTs.

Therefore, in the case of DHTs, we advocate using Soundex coding to suppress spelling variations both

when inserting file metadata into the inverted index of a file-sharing system and when querying it. This automatically allows for approximate matching and spelling error suppression. If there is a need for incremental deployment of our algorithm (as may be necessary in P2P systems where peer software versions are updated over a period of months to years) the system can temporarily store both Soundexed and non-Soundexed versions of a keyword. In this way, legacy clients could use unmodified keywords for search, and new clients would search initially on both Soundexed and non-Soundexed keywords, switching over to Soundex when the transition is complete. This approach supports legacy clients, albeit with lower recall. However, this lower recall rate creates an incentive to upgrade.

3.1 The Problem of False Positives

Soundex has an inherent problem of creating false positives, returning results that users do not really want. Such a problem is general to any approach that suppresses spelling variations. To see this, suppose the user issued a query of the form "Here Sun" wanting to retrieve metadata for the song "Here Comes the Sun". Now, the Soundexed query is "H600 S500", which also matches "Her Son". The original query would not have retrieved "Her Son", so coding has resulted in a false positive.

In general, any code that maps several keywords to a single codeword can create false positives. More precisely, consider a coding M : {Keyword} $\rightarrow$ {Codeword}. Denote the set of misspellings of a keyword K by $S(K)$, and the cardinality of this set by $|S(K)|$. Let $K1$ and $K2$ be distinct correctly spelled keywords, and let $M(K1) = X1$, and $M(K2) = X2$. Then, ideally,

- $X1 \neq X2$, that is, two keywords do not map to the same codeword;
- $\forall k \in S(K), M(k) = M(K)$, that is, every misspelling of a keyword maps to the same value as the keyword itself[2].

Such an ideal mapping can correct all misspellings. However, real-world mappings are non-ideal and suffer from two types of errors corresponding to the two conditions for M above:

- two keywords $K1$ and $K2$ (in addition to $S(K1)$ and $S(K2)$) may themselves map to the same value, i.e. $M(K1) = M(K2) = M(S(K1)) = M(S(K2))$. For example, both "Here" and "Her" map to "H600".
- an element $k \in S(K)$ may map to $M(K')$ instead of $M(K)$. This will result in searches for k actually searching on $M(K')$ thus finding titles with K' instead of K. For example, a search on "Sur" (instead of "Sun") would result in a search on "S600", which would find keywords like "Sir" instead of "Sun".

[2]Note that if $S(K1) \cap S(K2) \neq \emptyset$, then we have an inherent ambiguity in a misspelled keyword that is independent of the choice of M.

Both these cases lead to false positives and therefore lower query precision. The query answers contain results that are not relevant to the original query. Unfortunately, Soundex suffers from both types of errors, leading to a significant false positive rate.

In our experiments, we find that Soundex does indeed suffer from a high rate of false positives. However, sorting query results by their edit distance from words in the original query greatly increases the quality of the results. We find a very low false positive rate when a query result returned by Soundex has all its terms within an edit distance of one from the original query. These results are all presented in our evaluation. In the following section, we present the methodology of our experiments.

4 Methodology

We do not need to simulate a file-sharing system's entire functionality to evaluate the effectiveness of approximate matching algorithms. Instead, we merely need to evaluate the algorithms on traces of file sharing queries and replies. In the remainder of this section, we present the file-sharing datasets and the coding algorithms used in our experiments.

4.1 File-Sharing Datasets

The ideal dataset for studying query success rates in a file-sharing system should include all queries and all content stored in the system. Obviously, this is impossible for a large-scale file-sharing system with node churn. We found that we could not even reuse any previously collected datasets. Many datasets anonymize their queries and file metadata, making it impossible to experiment with misspellings. In the remaining datasets, the set of peers whose queries are captured and the set of peers whose file metadata is recorded differ. This makes it impossible to accurately measure query success rates.

In our work, to avoid underestimation of query success rates, we measured the query and the result streams for the same set of peers. While this approach only approximates the entire system's query success rate, it does capture the "true" query success rate for this subset of participants.

Our Gnutella Trace To measure query success rate, we instrumented several Limewire (Gnutella) ultrapeers to record all queries and query results, as well as all file metadata stored at their (peer) leaves. Our trace captures 22,457 peers over the course of one month, starting on May 18th, 2006 and ending on June 16th, 2006. Our dataset includes a total of 66,128 files and 1,830,079 queries.

Other Traces Besides our Gnutella trace, we analyzed three other datasets. Although these datasets were inadequate for studying query success rates, they do allow us to gauge the prevalence of misspelled keywords in different information-sharing systems. These datasets were:

• file names from Napster collected in 2002 [5];

• all movie titles in the IMDB database whose main language is English produced since 1950 (we downloaded this dataset from the IMDB website [6]);

• all clips found by an exhaustive crawl of YouTube's website in July 2006.

We convert metadata in these datasets into keywords using a code snippet from the Limewire software. This code uses 13 different characters to separate metadata into keywords. It also handles accents and Unicode characters adequately. In this way, our experiments match the behavior of a DHT client using the same query and metadata handling code as a Gnutella Limewire client.

4.2 Matching Algorithms

We evaluate five different matching algorithms over our Gnutella trace: exact-match, prefix-match, Jaccard's coefficient, TF-IDF, and Soundex. Today's file-sharing systems (whether using an unstructured overlay or a DHT) use either a prefix-match or an exact-match algorithm to answer queries. In our experiments, these matching algorithms serve as a baseline to measure how effective the more sophisticated, approximate matching algorithms work. Next, we briefly describe each of our matching algorithms:

Exact-Match: A file is included in a query's result only if every keyword in a query *exactly matches* one keyword in the file's metadata. This matching algorithm corresponds to the search behavior of generic file-sharing systems.

Prefix-Match: A file is included in a query's result only if every keyword in a query *matches the prefix* of one keyword in the file's metadata. This matching algorithm corresponds to the search behavior in many of today's unstructured file-sharing systems.

Jaccard's Coefficient: This algorithm assigns a similarity score to each file by computing Jaccard's coefficient [7] between the 3-grams of the query and the 3-grams of the file's metadata. A file is included in a query's result only if its score is at least 0.4. Section 5.2 will present our rationale for choosing this threshold.

TF-IDF: Similar to Jaccard's coefficient algorithm, except we use TF-IDF weights to measure similarity. We use a threshold score of 0.6 to include a file in a query's result. Section 5.2 will present our rationale for choosing this threshold.

Soundex: We use the Soundex algorithm presented in Figure 1 to map keywords to 4-byte codewords. A file is included in a query's result only if every codeword in a query *exactly matches* one codeword in the file's metadata. As discussed earlier, Soundex suffers from false positives. To alleviate this problem, we add a simple heuristic to Soundex – we drop results returned by Soundex other than those where every keyword in the query is within an edit distance of 0 or 1 from a keyword in the file's metadata. We use the term *Soundex + edit distance* to refer to this algorithm.

	Gnutella queries	*Gnutella files*	*Napster files*	*IMDB files*	*YouTube files*
# unique titles	1,830,079	66,128	444,467	119,857	150,512
# of unique keywords	299,004	34,397	208,700	54,927	132,478
avg. # keywords per title	3.91	8.47	4.84	3.45	12.26
# 4-byte soundex codes	14,115	5,732	9,698	4,375	9,731
avg. # keywords per 4-byte soundex	21.18	6.00	20.48	12.82	13.62

Table 1. **High-level statistics of the datasets.** *Section 4.1 describes the methodology for collecting these datasets.*

4.3 High-Level Trace Characterization

In this section, we present a high-level characterization of our trace. Table 1 shows summary statistics.

Not surprisingly, queries are shorter than file metadata. A query in Gnutella has an average of 3.91 keywords, whereas files in Gnutella, MP3s in Napster, and video titles from IMDB have over 8, 4, and 3 keywords per title, respectively. Since YouTube's titles are descriptions of content entered by users, they are significantly longer: the average content title on YouTube has over 12 keywords (the longest YouTube title has 105 keywords). This suggests an interesting trend – people enter increasingly longer content descriptors to label their content. Thus the need for better search tools that can find relevant content is growing in these content-sharing systems.

The number of unique entries in a file-sharing DHT serving this data is captured by the number of unique keywords in the file datasets in Table 1. While the number of keywords stored by each DHT participant decreases with the use of Soundex, the number of values stored for each keyword increases proportionally. This can potentially create a load imbalance if the number of keywords is not much higher than the number of DHT participants.

5 Evaluation

In this section, we present our evaluation of algorithms used to suppress spelling variations in file-sharing workloads. We start by analyzing the prevalence of spelling errors in such workloads. Next, we evaluate query recall rates based on using Soundex and two approximate matching algorithms: one based on the Jaccard coefficient and one based on TF-IDF.

5.1 How Common Are Spelling Errors?

We evaluated the frequency of misspellings by using the "aspell" spell checker version 0.50.5. Since many of our entries contain proper names, we have enhanced the aspell's dictionary with a list of all words found in the titles of all Wikipedia articles[3]. We also configured

[3]Of course, we are only $estimating$ the prevalence of misspellings, because defining a list of correct spellings is inherently difficult when proper names are allowed. Our choice (Aspell + Wikipedia) may not be perfect, but it gives a reasonable idea of the prevalence of alternate spellings.

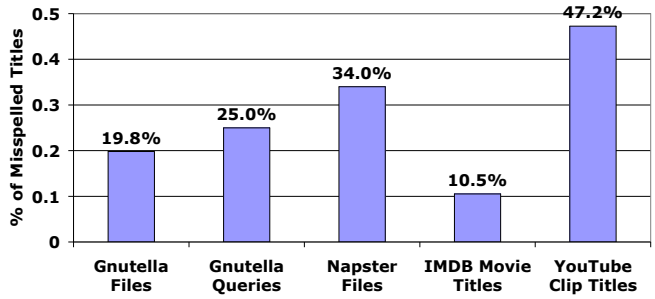

Figure 2. **Fraction of misspelled titles in file-sharing workloads.** *A title is misspelled if at least one of its words is incorrectly spelled.*

aspell to ignore accents when checking words.

Figure 2 shows how frequently misspellings occur in file-sharing datasets. We find that 25% of Gnutella queries and 20% of Gnutella file metadata contain at least one keyword that is misspelled. In Napster, 34% of all MP3 file metadata are incorrectly spelled. The rate of movie titles containing at least one incorrectly spelled keyword is very low in IMDB: 10.5%. The lower rate of misspellings in IMDB is consistent with the fact that there is editorial oversight on the content. Finally, it appears that almost half (47.2%) of all clips' titles posted on YouTube are incorrectly spelled. Unlike Napster and Gnutella, YouTube's titles are true file metadata; they are descriptions entered by users when posting their clips. Since these descriptions are longer than P2P's filenames (by a factor of three on average), the chance of having an incorrectly spelled word is much higher.

5.2 Can Unsuccessful Queries Become Successful by Handling Spelling Errors?

In this subsection, we investigate whether unsuccessful queries in today's file-sharing P2P systems could become successful by fixing spelling errors. We start by separating queries with answers using an exact-match algorithm from those with no answers. For each query, we compute the Jaccard's coefficient and the TF-IDF between the query and every file metadata in our dataset. Figure 3 shows the distribution of the highest Jaccard's coefficients (on the left) and the highest TF-IDF weights (on the right) for all queries.

This figure shows that successful queries have higher scores than unsuccessful ones. Nevertheless, 9.8% of unsuccessful queries have very high Jaccard's coefficients (0.4 or higher). Similarly, 12.5% of unsuccessful queries have high TF-IDF weights (0.6 or higher). This demonstrates that many unsuccessful queries could become successful using sophisticated approximate matching algorithms. This suggests that fixing spelling errors could help increase the success rate of queries in today's file-sharing workloads.

In our next experiments, we include a file in a query's result only if its Jaccard's coefficient is at least 0.4 or its TF-IDF weight is at least 0.6. Figure 3 shows that these threshold settings are reasonable: many successful queries based on exact-match have scores higher than our threshold, while most unsuccessful queries have lower scores. Also, in all our empirical observations, we have found these threshold settings to be adequate.

5.3 What is the Recall Rate of Our Algorithms?

We now turn our attention to the main thesis of our work – the improvement in the recall rate by use of coding algorithms. Figure 4 shows improvement in the recall rate of our matching algorithms. For Gnutella queries with no coding, the measured query success rate is only 7.3%. Using prefix-match, the query success rate is 8.5%. Using approximate matching algorithms, we find a recall of 12.9% for Jaccard's coefficient and 16% for TF-IDF. With Soundex, the recall rate increases significantly, to over 23.5%. Adding the edit distance heuristic to handle false positives leads to a recall rate of 10.2%. This experiment illustrates that both approximate matching algorithms and Soundex help improve recall despite spelling errors in today's file-sharing workloads.

While these results demonstrate that our matching algorithms are effective at increasing query success rates, it is unclear whether their query answers are relevant or not. To see this, note that a trivial algorithm that returns the same answer to every query would exhibit a query success rate of 100% while producing irrelevant results. Therefore, we have to look beyond the query success rate to evaluate the true effectiveness of different coding algorithms.

5.4 Are the Query Answers Relevant?

Measuring whether query answers are relevant is challenging. There are no known ways to judge whether an answer is relevant in an algorithmic and systematic manner. Instead, we need to rely on human subjects to determine whether answers are relevant.

For each experiment, we take a sample of 100 queries and query answers and we manually determine whether an answer is relevant or not. Each of these paper's authors (four in total) performed this experiment in isolation, and we report our results combined. We only examined successful queries that would be left unanswered by an exact-match algorithm, such as the one used in a generic DHT.

Figure 5 shows that at least two thirds of the answers returned by the approximate matching algorithms contain relevant entries. As we have anticipated, we find that Soundex has a high rate of false positives: only 13% of answers are relevant. However, using a simple heuristic, such as to filter out all Soundex results unless their edit distance is at most one, is very effective. The query answers are as relevant as the ones returned by sophisticated approximate matching algorithms.

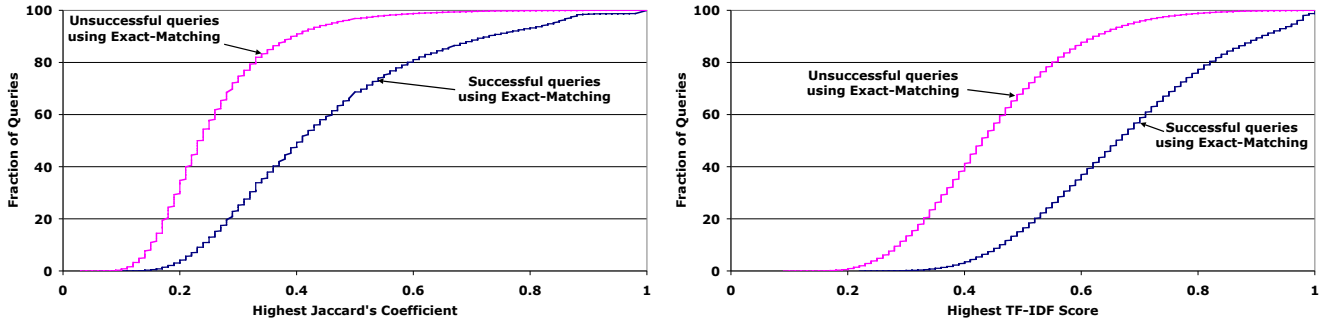

Figure 3. **CDF of Approximate Matching Algorithms' Scores.** *We separate successful queries using exact-matching from unsuccessful queries using exact-matching and we compute their highest Jaccard's coefficient (on the left) and TF-IDF weight (on the right). This figure shows that while successful queries have high scores, there are many unsuccessful queries with high scores as well.*

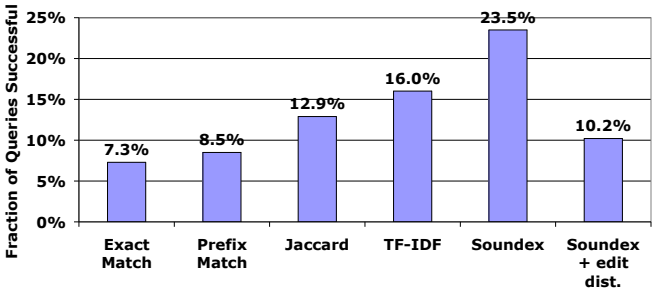

Figure 4. **Fraction of successful queries.** *This figure shows that a simple coding algorithm like Soundex handles spelling errors in a file-sharing workload as effectively as sophisticated, approximate matching algorithms.*

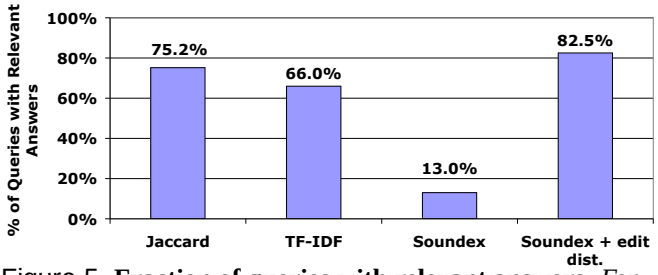

Figure 5. **Fraction of queries with relevant answers.** *For each matching algorithm, we selected 100 queries with no answers based on an exact-match algorithm and we manually determine whether an answer is relevant. Each bar represents the average of four experiments, performed by each of this paper's authors.*

6 Conclusions

The focus of our work has been to improve query recall without loss of precision in file-sharing systems with endemic misspellings. We have shown that spelling errors are common in content workloads. We present several datasets collected over four years (from the Napster dataset collected in 2002 [5] to the YouTube dataset collected in 2006) and in all these datasets, we find a significant fraction of content descriptors containing spelling errors. These findings suggest that dealing with spelling errors cannot be overlooked in today's Internet content sharing systems.

We have evaluated the performance of several matching algorithms over these datasets. We found that these matching algorithms increase today's query success rates by 51%, 88%, and 20% for Jaccard's coefficient, TF-IDF weights, and Soundex + edit distance, respectively. At the same time, the quality of these algorithms' query answers is high: 75.2%, 66%, and 82.5% of the previously unsuccessful queries have relevant answers based on the Jaccard's coefficient, TF-IDF weights, and Soundex + edit distance, respectively.

Overall, we find that approximate matching algorithms based on q-grams can deal with spelling errors effectively and are well-suited to unstructured P2P networks. Unfortunately, because of their reliance on q-grams, these techniques are too costly for DHTs. Instead, we find that Soundex, a simple, fast, and efficient algorithm, together with an edit distance heuristic, works surprisingly well – the query recall rate improves with no loss of query precision. Soundex can be easily deployed over most DHTs today. Our work provides effective solutions for dealing with misspellings in both structured and unstructured P2P systems.

Acknowledgments

We would like to thank Krishna Gummadi, Jeremy Knight, Alec Wolman, and the anonymous reviewers for their comments and feedback.

References

[1] Google. Britney Spears spelling correction. `http://www.google.com/jobs/britney.html`.

[2] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proc. of VLDB*, 2001.

[3] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[4] R. Parsons. Soundex – the true story, October 2005. `http://west-penwith.org.uk/misc/soundex.htm`.

[5] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of MMCN*, January 2002.

[6] The Internet Movies Database. `http://www.imdb.com`.

[7] Wikipedia. Jaccard Index. `http://en.wikipedia.org/wiki/Jaccard_index`.

[8] Wikipedia. Tf-idf. `http://en.wikipedia.org/wiki/Tf-idf`.