Using query transformation to improve Gnutella search performance

Surendar Chandra
surendar@acm.org

William Acosta
william.acosta@utoledo.edu

***Abstract*—Gnutella peers independently choose the way in which objects are named as well as queried. Using a long term analysis of the files shared and queries issued, we show that this flexibility leads to a mismatch between the way that objects were named and the way that users were issuing search queries. Thirty percent of the failed queries contained keywords that were not present in any file name while the remaining queries failed because no file name contained all the keywords in a particular query. Our earlier analysis of files shared in the popular iTunes music file sharing system showed that standardizing the file names to make them easier to search is not a viable alternative. Instead, we transform the queries to better match the objects available in the system. We investigated spell correction (using file name information from the neighborhood) as well as remove query keywords. We consider the results from the transformed query to be relevant to the intent of the original query if the transformed query used many of the original keywords and the number of matching files closely matched the number of matches for typical successful queries. Our approach is practical and uses information available within the immediate neighborhood of an ultra-peer. An overlay agnostic analysis shows that our transformation improves success rates from 45% to between 72.5% and 91.2%. Using our *Hybrid* mechanism as a Gnutella middleware, our transformation produced relevant results for about 61% of the failed queries.**

***Keywords*-unstructured peer-to-peer, query transformation**

I. Introduction

Gnutella is popular and accounted for over 40% of all P2P users [1]. However, its search performance [2] remains poor; between 2003 and 2006, the query success rate at a forwarding peer only increased from 3.5% to 6.9% [3].

In general, the performance of unstructured P2P systems depend on the network topology, search mechanisms that route queries over the overlay, annotations of shared objects as well as in the way that users request objects. A large body of prior research efforts (including [4], [5], [6]) investigated various overlay maintenance and search mechanisms that reduced the network overhead for finding shared objects.

Earlier [7], we investigated the object annotations and user generated queries on the Gnutella network. Note that objects in Gnutella were independently annotated by the content provider without any global coordination. We showed that over 44% of the queries had no matching objects regardless of the overlay or search mechanism used to locate the objects. In this paper, we focus on these failed queries.

Gnutella [8] requires a match of all query terms against terms in the particular file name. We show that 30% of the failed queries had at least one keyword that was not found in any of the file names. The keywords from the remaining failed queries was not all present in the same file name.

One approach to improve search performance is to force users to uniformly name objects; allowing other users to easily query for them. Earlier [9], we analyzed the annotations of objects shared by campus iTunes users. iTunes uses the Gracenote service to automatically annotate new objects that are imported from CDs; users are then allowed to customize the annotations. We observed that users exerted control over how they named the shared objects. For example, we observed over 1,452 different genre names from 239 users even though iTunes itself shipped with just 24 genres. Hence, it is unlikely that we can improve Gnutella search performance by standardizing the way that objects are named.

Another approach is to change the Gnutella matching semantics and successfully match files that contained any query keyword (instead of requiring a match for *all* keywords). Note that most queries [2] used less than four keywords. We observed that the median number of matching files using this policy was 24,233 files per query; this mechanism will exhibit low specificity.

We improve query success rates by transforming failed queries to better match the shared file names. We constrain our transformation using a notion of objects that are relevant to the intended matches for the original query. We define relevancy based on the number of original keywords as well as the distribution of the number of matching files. Note that our approach does not guarantee that the new query can be resolved. Given the dynamic nature of P2P systems, the challenge was to choose this transformation without requiring global knowledge. Our approach is practical, each peer uses object name information from its own peer neighborhood in order to choose the query keywords for the transformed query. First, we employed *Qgrams* [10] to perform approximate string matching and spell correction of query keywords. We only used the file names from the peer neighborhood which we show is more accurate than approaches that used a standard dictionary [2]. Next, we investigated mechanisms that selectively removed keywords from the original query. We considered random selection, term popularity and co-popularity as well as a *Hybrid* policy. Our evaluation used captured query traffic and shared files from the Gnutella network and was overlay agnostic. Our mechanism improved the overall query success rate from

45% to as much as 91%.

Finally, we implemented a middleware using our *Hybrid* query transformation mechanism. Using this middleware on the live Gnutella network, we successfully matched about 61% of the failed queries. A subjective analysis of the results indicated that failed queries received results that were relevant to the original query.

Next, Section II describes the trace data that is used for our analysis. Section III describes the system constraints while Section IV describes our query transformation mechanisms. We evaluate our mechanisms in Section V and describe our middleware in Section VI. Section VII discusses related work with concluding remarks in Section VIII.

II. System architecture

The Gnutella protocol [8] specifies mechanisms to tokenize and create sets of query keywords (Q_i) and file annotations (F_x). Terms are case insensitive and regular expressions are not supported. Query resolution requires matching all query keywords against annotations from a particular file (i.e., Q_i matches F_x iff $Q_i \subset F_x$). A query fails when no single file contains all the keywords from that query. Note that we do not consider an optional stricter and rarely used extension to the Gnutella protocol that allows peers to require that the match be present in the same number and order as in the query.

A. Empirical trace data for system analysis

For our analysis, we require the names of shared files as well as the queries issued by Gnutella users. We collected the file names using our file crawler (described in [11]). We crawled the Gnutella network and requested the list of shared files from each discovered peer. We collected the user issued queries by modifying the Phex [12] open source Gnutella client (described in [7]). Our modified client participated as a Gnutella ultra peer and logged all the queries that were routed through it. We simultaneously ran the file crawler and the query capturing client in order to ensure a consistent representation of the Gnutella system. We discovered approximately 20 million files shared by 37 thousand peers in April 2007 and 17 million files shared by 34 thousand peers in February 2008. The query traces correspond to about 200 thousand queries per day.

B. Nature of query failure in Gnutella

In [7], we analyzed the limits of query matching in Gnutella. Our current focus is on resolving those failed queries. First, we analyze the amount of query terms that do not appear in any file names - such queries cannot be resolved using improvements in query routing policies. In practice, some queries will also fail because of the limitations of the overlay, search mechanism or because a particular peer that was hosting the file was offline. Our analysis is agnostic to these concerns. Also, Gnutella requires all query terms be present for a successful match.

For our analysis, we first created a set of all file terms (F). For each user issued query, we calculate the number of terms in a particular query that did not match any term in F. For example, consider the keywords in a particular query $\{q_1, q_2, q_3\}$ and file name terms F:$\{q_1, f_1, f_2, \ldots, f_n\}$. This query will have two terms that did not match any file name terms (i.e., $\{q_2, q_3\}$). For our traces, we observed that 30% of the query keywords have no matching file terms. Improvements in network overlays and search mechanisms alone cannot resolve queries which contain these keywords.

III. Challenges in choosing the transformation

In this paper, we transform failed queries into another query that can be resolved in Gnutella. We prefer these transformed queries to match objects that were closely related to objects that were requested by the original query. Next, we describe the challenges encountered in transforming failed queries and our approach to address them.

A. Identifying related files

The transformed query should match files that were related to the intended results from the original query. However, it is not easy to divine the user's intention on what files they were searching when they issued a particular query. Identifying semantically related objects requires global knowledge and complex processing which are difficult in a P2P scenario. Hence, we define our notion of related objects by choosing the keywords in the transformed query from the original query. Our intuition has its limitations. Consider a failed query that we observed in our traces: "*one paul hardcastle your*". The user was likely searching for the song "*You're The One for Me*" by the obscure artist *Paul Hardcastle*. However, a transformation that removed the keyword *Hardcastle* is likely to match songs from the album *Stage One* by the popular artist, *Sean Paul*.

B. Limiting the scope of related files

The transformed query may match a few or millions of objects. If the result set was too large, many of the returned files were likely not relevant to the original query: i.e., these results might exhibit high recall but low precision. The challenge is to choose transformations that limits the number of files matched to a reasonable number. We consider transformations of failed queries that show similar matching behavior (in terms of the number of objects matched) to the original queries as a reasonable transformation. In order to understand what is reasonable, first we plotted the number of files that matched various queries in Figure 1. The data sets from 2007 and 2008 showed that about 55% of the queries did not match any file in the entire system. The median number of matching files was about twenty five. Hence, we choose transformations that match about twenty five files.

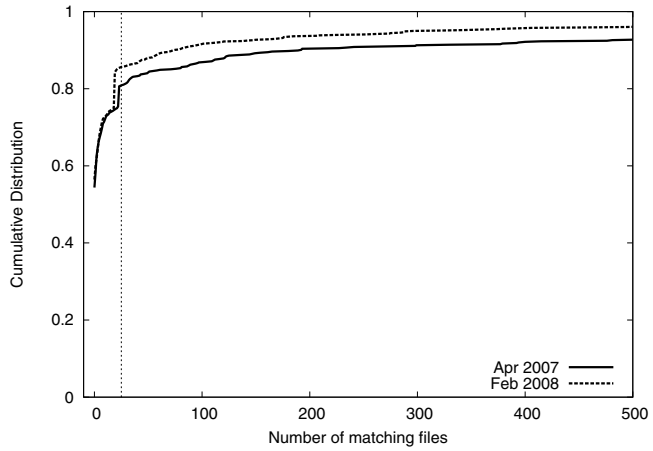


Figure 1. Matching files for queries issued by Gnutella users. Queries were matched against shared objects from 37K peers (2007) and 34K peers (2008) and is overlay and search agnostic

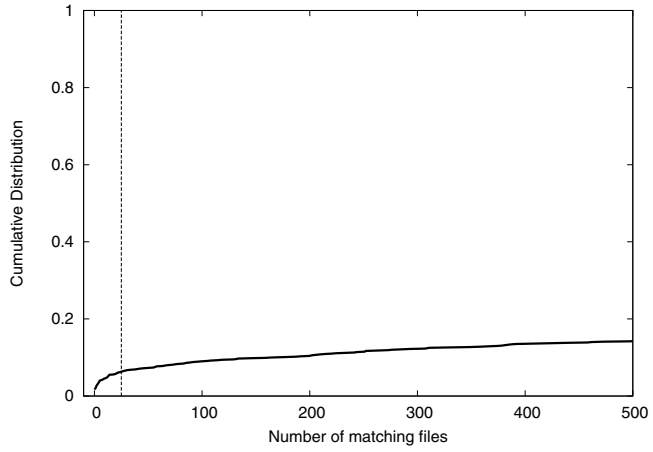


Figure 2. Matching files for considering *any* keywords as a match

We explain this notion using a *match any keyword* transformation. As compared to the Gnutella requirement to match all query keywords, consider a transformation that accepted a match of any query keyword (i.e. $|Q \cap F| \geq 1$, instead of $Q \subset F$). We plot the number of files that matched this transformation of failed queries from our 2007 data set in Figure 2. We note that the scope of this transformed query was far different than what was illustrated in Figure 1 with a median size of 24,233 files (not illustrated). Hence, we rejected this transformation and sought transformations whose median match size is roughly twenty five files.

C. Practical transformation

Finally, our transformation should be practical. Peers in unstructured P2P systems make independent decisions (regarding overlay maintenance, query routing etc.) without global knowledge. Unlike our offline analysis of file names collected from a large number of peers, it is difficult for individual peers to collect information about all shared objects. Hence, we only use information about files in the immediate peer neighborhood.

IV. Our query transformation

We consider three transformations: correct misspelt keywords, removing keywords as well as a hybrid approach.

1) Correcting misspelt keywords: Zaharia et al. [2] checked Gnutella queries using the *aspell* dictionary that was enhanced with the titles of all Wikipedia articles and found that 25% of Gnutella queries and 20% of shared files contained at least one keyword that was misspelt. However, earlier work [13], [7] showed that 17% of the queries were issued in a multi-lingual character set. Songs were sometimes named using slang terms (e.g., *Dat* instead of *That*) that were unlikely to be part of regular dictionaries. Also, query keywords and file name terms were observed to change with time perhaps, to account for the release of newer songs. An approach that built a static and customized dictionary will quickly become obsolete.

For each of the query keywords that did not match any file terms in the peer neighborhood, we correct misspelt keywords using Qgrams [10] of size three and an edit distance of one. We used the set of terms that occurred in the files in the peer neighborhood as the dictionary. If a suitable substitute term was identified, the query keyword was transformed to the new term.

2) Query subset by removing keywords: We also investigate transformations that create subsets of the original query. The number of matching files for the transformed query inversely depends on the cardinality of the transformed query. As discussed in Section III-B, we consider a small number of matches to be preferable. One approach to create a subset would be to issue sub-queries using each individual keyword from the original query. Using the positive responses, we can develop a client side mechanism that chooses the query subset that contains the most number of keywords and yet produces the smallest number of matches. However, as illustrated in Figure 2, each keyword can match a large number of files, making such an approach impractical (transmitting a large response places tremendous load on the Gnutella network). Hence, we desire mechanisms that chooses the subset without operating over a large number of matches. Zaharia et al. [2] found an average of 3.91 keywords per query. Our analysis of queries showed that most (over 99%) of the queries contained less than six terms. Hence, we explore subsets of size (n) one, two and three.

Random: We randomly selected n keywords from an original query of size k keywords. This approach did not require any information about files shared in the system.

Popular: This policy assumes that the presence of less popular file terms in the original query keywords affects the query success. We selected the n popular keywords based

on the number of occurrences of each of the query keyword in the shared files in the peer local neighborhood.

CO-Popular: This approach selects keywords that occurred together in file names. We first compute the co-occurrence value between each pair of keywords in the original query among file names. If the co-occurrence value for a pair of keywords was defined (i.e., > 0), then the pair of keywords was added to the transformed query. If all keywords from the original query string were added, then the term with the least number of occurrences in the file names was removed to create a new query of size n. On the other hand, if no keywords was selected for the transformed query (because all pairs of keywords had co-occurrence value of 0), then we chose the n popular keywords among file terms.

3) Hybrid: Finally, we tried a hybrid approach: we applied the *Spell* followed by the *CO-Popular* approach to the original failed query in order to create the transformed query.

A. Practical considerations

We used file name information from the peer neighborhood. The transformed query was then evaluated against the full set of shared files. Our approach is practical as it reduces the amount of unnecessary network traffic.

V. Results

We analyze the performance of *Spell*, *Random*, *Popular*, *CO-Popular* and *Hybrid* query transformation mechanisms using the 2007 data set (Section II-A). The results from analyzing the 2008 data set were similar to the 2007 data set and are not shown for brevity. We transform each failed query and evaluate the number of files that match the transformed query against files from all the peers. We used a neighborhood size of 64 peers, the typical number of leaf peers connected to a Gnutella ultrapeer [14]. Section V-D analyzes the scalability of this choice of 64 peers.

A. Correcting misspelt keywords

First, we plot the number of matching files for correcting the spelling of keywords of failed queries (55% of the original queries) in Figure 3. We used the dictionary information of file names from a 64 peer neighborhood. The results were similar to the number of matching files for the original query (Figure 1). We observe that 50% of the failed queries continue to fail even with the transformation. In other words, 72.5% of the original queries returned some results (45% from the original query and 27.5% for the transformed query). 80% of the successfully transformed queries returned results of twenty five files or less. Note that Zaharia et al. [2] reported an improvement of 51% of query success rates using an approximate matching algorithm.

B. Query subset by removing keywords

We analyze transformation mechanisms that remove keywords from failed queries. We varied the number of keywords in the transformed query between one and three. We plot the number of matching files for transforming failed queries using *Random*, *Popular* and *CO-Popular* policies in Figures 4(a), 4(b) and 4(c), respectively.

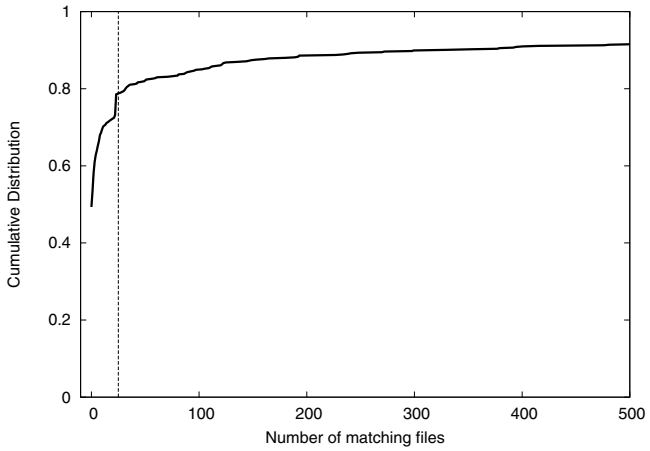
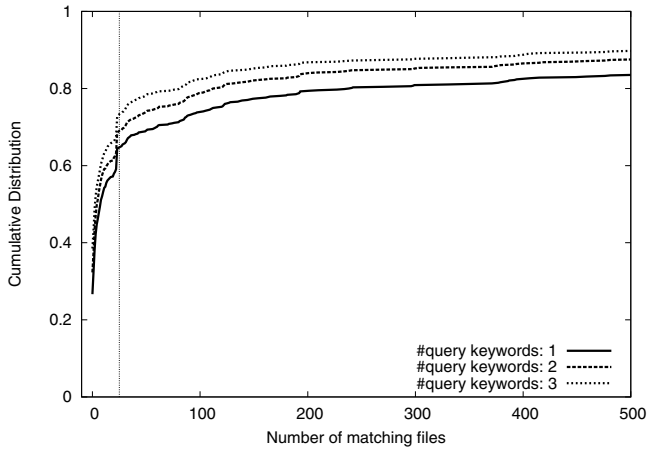


Figure 3. Matching files for correcting misspelt keywords of failed queries

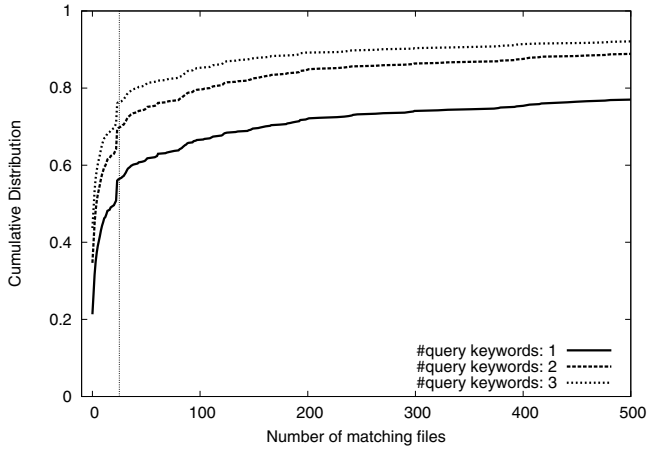
From Figure 4(a), we note that the *Random* mechanism did not successfully resolve 26.6%, 32.2% and 38.5% of the originally failed queries for choosing one, two and three keywords for the transformed query, respectively. Thus, 40.37%, 37.3% and 33.83% of the original queries were additionally successful because of this transformation, leading to success rates of 85.37%, 82.3% and 78.83% for choosing one, two or three keywords in the transformed query, respectively. Also note that 52.41%, 54.49% and 57.11% of the successfully transformed queries matched about twenty five files or less. Note that the *Random* policy does not consider files names shared in Gnutella; matched files for the transformed query might not be relevant to the intent of the original query.

Next, we investigate the *Popular* transformation mechanism in Figure 4(b). We note that 21.33%, 34.66% and 43.70% of the failed queries were not resolved by the transformation of choosing one, two or three of the most popular keywords, respectively. This translates to matching 88.63%, 80.94% and 75.97% of the queries using the original query and the transformed query for choosing one, two and three of the most popular terms, respectively. Also note that 45.01%, 54.20% and 58.42% of the successfully transformed queries matched twenty five or less objects.

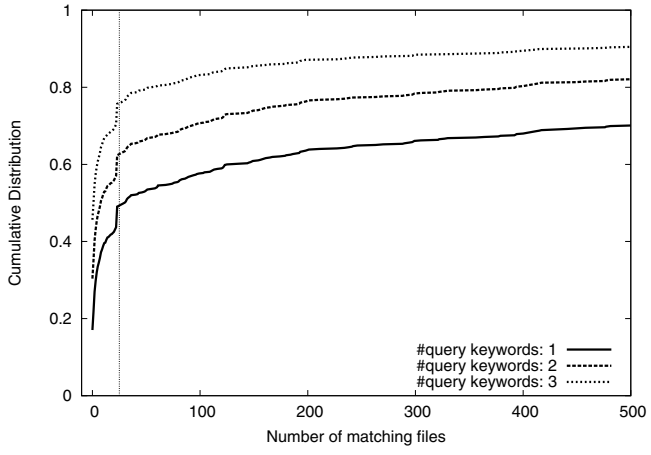
CO-Popular transformation (Figure 4(c)) showed that 17.04%, 30.37% and 45.63% of the failed queries were not resolved by the transformation to choose one, two or three co-popular keywords, respectively. Overall, 90.63%, 83.30% and 74.90% of the queries were matched, both using the original query as well as transforming them to choose one, two and three keywords, respectively. It should be noted that 39.28%, 46.80% and 56.15% of the successfully transformed queries matched twenty five or less objects.



(a) *Random*



(b) *Popular*



(c) *Co-Popular*

Figure 4. Matching files for transforming failed queries

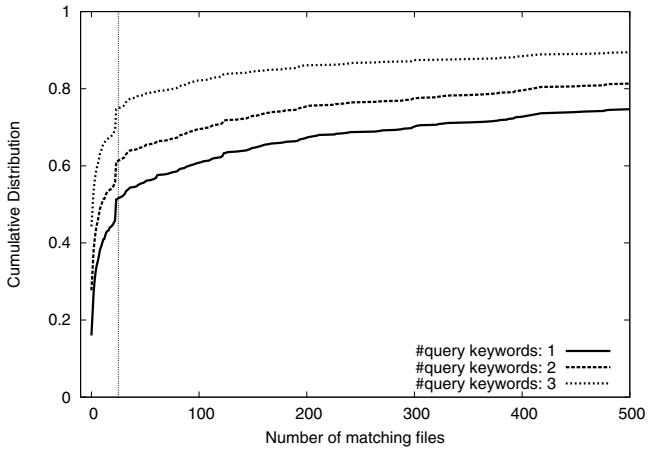


Figure 5. Matching files for *Hybrid* transformation of failed queries

C. Hybrid approach (Spell+CO-Popular)

Finally, we analyze the behavior of the hybrid *Spell+CO-Popular* approach. From Figure 5, we note that 16.00%, 27.70% and 44.30% of the queries were not resolved by the hybrid transformation of failed queries: first by spell checking and then by a *CO-Popular* scheme that chose one, two or three keywords, respectively. In effect, the success rate of queries improves from the original 44% of queries to 91.20%, 84.77% and 75.64% with a hybrid transformation using one, two and three keywords, respectively. We also note that 42.68%, 46.93% and 55.31% of the transformed failed queries resulted in twenty five or less matches for choosing one, two and three keywords, respectively.

Summary of results

Transformed queries with more keywords were likely to be similar to the original query. Our analysis showed that policies that chose three keywords resulted in an improvement of the success rate from 45% to about 73%, 79%, 76%, 75% and 76% for using the *Spell*, *Random*, *Popular*, *CO-Popular* and *Hybrid* mechanisms, respectively. Also, assuming that results which produced fewer matches (less than twenty five files) were more likely to be relevant to the intent of the original query, we observed success rates of about 57%, 58%, 56% and 55% for the transformed queries using the *Random*, *Popular*, *CO-Popular* and *Hybrid* mechanisms, respectively. Anecdotally, the *Hybrid* policy appeared to produce more relevant responses. Also, Figure 5 shows that for a scheme that chose three keywords, 89.33% of the successfully transformed queries resulted in less than 500 matches, a slightly better percentage than other policies. We further explore this policy in Section VI.

D. Scalability analysis of query term transformation

In the previous section, we chose a neighborhood size of 64 for the various query transformation mechanisms. 64

Table I
SUCCESS RATES (MATCHING ANY FILE) FOR VARIOUS NEIGHBORHOOD SIZES. *Random* DID NOT USE NEIGHBORHOOD INFO

Query transformation	Neighborhood Size	Success rate for failed queries		
Spell	64	50.07%		
	200	51.11%		
	400	47.85%		
		Number of keywords		
		n = 1	n = 2	n = 3
Popular	64	78.67%	65.33%	56.30%
	200	77.48%	63.70%	55.26%
	400	78.67%	65.04%	55.70%
CO-Popular	64	82.96%	69.63%	54.37%
	200	83.26%	69.48%	56.00%
	400	84.89%	71.26%	57.04%
Hybrid	64	84%	72.30%	55.70%
	200	84.15%	72.15%	58.52%
	400	83.45%	74.22%	59.56%

is the typical number of leaf peers maintained by Gnutella ultra-peers [14]. Hence, an ultra-peer node will already maintain overlay information for about 64 peers within a single hop on the P2P overlay. Also, our prior work [7] showed that most peers shared less than 400 files. Our experiments showed that the list of shared files names was approximately 15 KB per peer or about 970 KB for 64 peers; a reasonable storage overhead. On the other hand, choosing a larger number of peers might yield a better query match.

Next, we examined the effect of varying the peer neighborhood size on the various transformation mechanisms. We randomly sampled the set of peers to create groups of 64, 200 and 400 peers. We then evaluated the failed queries using *Spell*, *Popular*, *CO-Popular* and *Hybrid* mechanisms for various keyword sizes (using the neighborhood peer groups) and tabulated the success rates in Table I. Note that *Random* does not use any peer neighborhood information. We observe a small improvement for using larger neighborhood sizes; perhaps attributable to using different group members for each neighborhood size. For example, with the *Hybrid* mechanism, the success rate of 55.7% using 64 peers only modestly improved to 59.56% using 400 peers. Increasing the number of peers adds to the computational complexity of the transformation mechanisms. Hence, we continue to use a peer neighborhood size of 64.

VI. QUERY TRANSFORMATION MIDDLEWARE

So far, we analyzed the behavior of the transformation mechanisms using query and file name data collected from the Gnutella network. However, those analyses did not address the following concerns:

1) *Practicality:* An important design goal of our system was that the transformation mechanisms should be practical to implement in an unstructured P2P scenario. Though Section V-D showed that the choice of peer neighborhood of 64 was practical, we require further implementation experience to show the viability of our mechanisms.
2) *Relevancy of results:* Another concern is that the results from a transformed query might not actually be relevant to the original query. So far, we used the metric that a) maintaining a large number of original keywords in the transformed query and that b) the number of matching files appear similar to the number of matching files for the original query (Section III-B). This is insufficient to show the relevancy of the transformation mechanisms.

Subjective analysis of the matching files for the transformed queries can offer proof of our relevancy claims. Hence, we implement our *Hybrid* mechanism as a middleware. We subjectively analyzed the results and report the performance of the transformed queries.

We target our middleware to ultrapeer clients which are responsible for the bulk of query routing in Gnutella. We used a *Hybrid* mechanism that chose three keywords for our transformation. The middleware operates on top of the P2P overlay maintenance and query routing mechanism. The middleware layer collects information about the files shared by neighboring leaf peers during the peer handshake process of establishing a Gnutella connection. Using the list of file annotations, the middleware computes the co-occurrence counts between terms as well as the popularity distribution of terms. We store this information using a weighted graph with each vertex representing a term in the file annotations. An edge exists between two vertices if the corresponding terms appear together in a file name. The weight of the edge corresponds to the number of times the terms co-occur in a file name. The ultrapeer also computes the Qgrams for each file term for the *Spell* component. In our earlier work [3], we showed that the success rate at the ultrapeer was low; about 6.9% of the forwarded queries yielded any query response messages. Hence, we transform all incoming queries and issue them simultaneously as the original query in the Gnutella network. When the original query matches any items, we discard responses for the transformed query.

A. Experimental Results

For our experiments, we monitored the Gnutella network for failed queries (that were issued by some actual Gnutella user). We then transformed those queries using our *Hybrid* mechanism and then manually evaluated any matches for the transformed query. For this experiment, we used 100 queries and the results were subjectively evaluated by the authors.

Ignoring bogus peers: During our experiments, we discovered that some queries that we expected to fail were actually returning matching results. The matching files were always in the form of "<random prefix>-<query string>-<random suffix>". We suspected that some peers were positively responding to all queries irrespective

of whether they had matching files. To test our hypothesis, we generated random strings and observed that a small set of peers were always matching those queries. We identified these *bogus* peers by first issuing random query strings and then ignored results from those peers.

About 61% of the failed queries returned positive results using the *Hybrid* transformation. Subjective evaluation of the results showed that many of the responses were relevant to the original query. First we show a successful transformation: the failed query "*barbara streisen woman love*" was transformed by *Hybrid* mechanism as "*barbara woman love*". The user appeared to have misspelt "*Streisand*" as "*streisen*". The *Spell* component did not have enough local files to correct this mis-spelling. On the other hand, the *CO-Popular* component removed a good keyword. Matching results for the transformed query included: *barbara streisand - woman in love.mp3*, *barbara streisand & beegees - wild flower - woman in love.mp3*, *barbara striesand - i am a woman in love.mp3*, *Bee Gees & Barbara Streisand - Woman In Love.mp3*, *Barbara Streisand - I am a Woman In Love.mp3*, all of which appear to be relevant for the original query. On the other hand, the query "*one hardcastle paul your*" was transformed into "*one paul your*" which matched *Lil Jon ft. E40 & Sean Paul - Snap Your Fingers.mp3*. As discussed earlier, the user was likely searching for the song "*You're The One for Me*" by the artist *Paul Hardcastle* and not the song by the artist *Sean Paul*. The number of matching files for transformations varied between one and ninety one.

VII. Related Work

Prior work focused on improving Gnutella overlays which in turn allowed search mechanisms to reach many unique peers. Law et al. [15] used expander graphs as overlays. Acosta et al. [6] developed a practical overlay that exhibited expander like properties. Wouhaybi et al. [16] constructed low-diameter resilient topologies. Other efforts focused on improving the search mechanisms. Lv et al. [4] traded off the number of query messages for the object location time using a random walk. Rhea et al. [17] used probabilistic routing mechanisms that routed queries using a potential function that estimated routes which lead to matching contents. Chawathe et al. [18] employed one-hop replication, topology adaptation and a search mechanism based on random walks to improve Gnutella search performance. We address an orthogonal issue which concerned matching the user issued queries with the names of the shared files. Our approach can fully exploit any advances in overlay maintenance or search mechanisms. Guo et al. [19] developed a content abundant cluster mechanism to route queries to nodes with a large number of objects. Our query transformation can make an object match more likely among the stored objects.

Structured P2P systems such as Chord [20] and Pastry [21] maintain overlays and route messages between peers based on a distributed hash table. The precise identifier of the object of interest is required in order to enjoy the search performance of structured P2P systems. Several mechanisms have been proposed to support keyword searches in structured P2P systems. Loo et al. [22] maintained an inverted index of the file terms for each shared file. Tang et al. [23] also described a term-based search mechanism which was built on top of a structured P2P system. Each peer in the system was assigned a set of terms and maintained an index of all files that matched those terms. Queries were resolved by obtaining the list of files that matched each of the query keywords. As we described in Section III-B, the number of matching files for each query keyword can be large.

Hybrid systems [22] optimized the DHTs using join indices. The join index for a term is a list of all files containing the term. Queries can be forwarded to a peer that is responsible for maintaining the index to any of the query keywords. This peer locally matches the remaining query terms to each of these matching files. In Section II-A, we showed that 30% of the failed queries in Gnutella contained keywords that were not present in any single shared file. Unless the query was routed based on a keyword that was not available in any file, one can incorporate our transformation techniques and locally match using fewer keywords.

In our earlier work [7], we analyzed the shared files and user issued queries in Gnutella. We showed that the query keywords and file terms exhibited Zipf like long tail distribution. However, the relative popularities did not correlate well with each other. This had significant implications for the synopsis used in a hybrid P2P [22]. In [7], we dynamically adapted the synopsis and selected terms for the synopsis to reflect popular terms in both the query workload and file distribution. In our position paper [11], we identified the temporal mismatch of the object and query popularity distributions. In this paper, we address the problem by removing query keywords that affect the search performance.

Pucha et al. [24] observed byte-level similarity in BitTorrent even though the file names did not indicate that the objects were identical. Gnutella searches only use query keywords to locate objects; we improve the search performance using these keywords. Audio search services such as Shazam (shazam.com) and Midomi (midomi.com) use Multimodal Adaptive Recognition System (MARS) to create song finger prints. An extension of our work can transform queries to locate the finger prints of relevant objects.

Zaharia et al. [2] showed that many query keywords and file terms were misspelt by checking them against standard dictionaries. However, about 17% of typical queries were issued in a multi-lingual character set [13], [7]. Songs were also officially named using slang terms (e.g., *Dat* instead of *That*) that were unlikely to be part of regular dictionaries. Also, we observer [7] temporal variation of query keywords and file name terms (to account for the release of newer songs). We dynamically built the dictionary using file terms from the peer neighborhood.

VIII. Discussion

The Gnutella network continues to be plagued by poor search performance. Prior work focused on improving the network query mechanisms. In this work, we show that the failure stems from a mismatch between the way different users name and query for objects. Improvements in network overlays and search mechanisms alone cannot address this problem. We introduce simple mechanisms to transform the failed queries. We carefully describe mechanisms to keep the transformations relevant to the intent of the original query. Through extensive simulations, we show that we can achieve significant improvements in query performance. Through a middleware, we show that our approach is practical. Through subjective analysis, we show hints that the approach is relevant. Future work should build better transformation mechanisms using information retrieval techniques that are amenable to the distributed nature of peer to peer systems. More sophisticated information retrieval techniques are likely to further improve on our results.

Acknowledgment

This work was supported in part by the U.S. National Science Foundation (CNS-0447671).

References

[1] E. Bangeman, "Study: Bittorrent sees big growth, limewire still ♯1 p2p app," Ars Technica, Apr. 2008. [Online]. Available: arstechnica.com/old/content/2008/04/study-bittorren-sees-big-growth-limewire-still-1-p2p-app.ars

[2] M. A. Zaharia, A. Chandel, S. Saroiu, and S. Keshav, "Finding content in file-sharing networks when you can't even spell," in *IPTPS'07*, Bellevue, WA, Feb. 2007.

[3] W. Acosta and S. Chandra, "Trace driven analysis of the long term evolution of gnutella peer-to-peer traffic," in *Passive and Active Measurment Conference (PAM'07)*, 2007.

[4] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *International conf. on Supercomputing*, 2002, pp. 84–95.

[5] L. Massoulie, E. L. Merrer, A.-M. Kermarrec, and A. Ganesh, "Peer counting and sampling in overlay networks: random walk methods," in *PODC '06*, Jul. 2006, pp. 123–132.

[6] W. Acosta and S. Chandra, "Improving search using a fault-tolerant overlay in unstructured p2p systems," in *IEEE ICPP '07*, Xian, China, Sep. 2007, p. 5.

[7] ——, "Understanding the practical limits of the gnutella p2p system: An analysis of query terms and object name distributions," in *ACM/SPIE: Multimedia Computing and Networking (MMCN'08)*, vol. 6818, San Jose, CA, Jan. 2008.

[8] T. Klingberg and R. Manfredi, "Gnutella 0.6," rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html, Jun. 2002.

[9] S. Chandra and X. Yu, "Share with thy neighbors," in *ACM/SPIE Multimedia Computing and Networking (MMCN 2007)*, San Jose, CA, Jan. 2007.

[10] L. Gravano, P. G. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate string joins in a database (almost) for free," in *Very Large Data Bases (VLDB '01)*, Rome, Italy, Sep. 2001.

[11] W. Acosta and S. Chandra, "On the need for query-centric unstructured peer-to-peer overlays," in *IEEE Hot Topics in Peer-to-Peer Systems (HotP2P '08)*, Miami, FL, Apr. 2008.

[12] "The phex gnutella client," http://phex.kouk.de.

[13] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability," http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html, Feb. 2001.

[14] A. H. Rasti, D. Stutzbach, and R. Rejaie, "On the long-term evolution of the two-tier gnutella overlay," in *IEEE Global Internet*, 2006.

[15] C. Law and K.-Y. Siu, "Distributed construction of random expander networks," in *INFOCOM'03*, vol. 3, Mar. 2003, pp. 2133–2143.

[16] R. H. Wouhaybi and A. T. Campbell, "Phenix: Supporting reilient low-diameter peer-to-peer topologies," in *INFOCOM'04*, vol. 1, Mar. 2004, p. 119.

[17] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *INFOCOM '02*, vol. 3, Jun. 2002, pp. 1248–1257.

[18] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM '03*, 2003, pp. 407–418.

[19] L. Guo, S. Jiang, L. Xiao, and X. Zhang, "Exploiting content localities for efficient search in p2p systems," in *Symp. on Dist. Comp. (DISC '04)*, Amsterdam, Netherlands, Oct. 2004.

[20] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: scalable peer-to-peer lookup service for internet applications," in *SIGCOMM*, Aug. '01, pp. 149–160.

[21] A. Rowstron and P. Drushel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Middleware 2001*, Nov. 2001, pp. 329–350.

[22] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein, "The case for a hybrid p2p search infrastructure," in *IPTPS '04*.

[23] C. Tang and S. Dwarkadas, "Hybrid global-local indexing peer-to-peer information retrieval," in *NSDI'04*, Mar. 2004.

[24] H. Pucha, D. G. Andersen, and M. Kaminsky, "Exploiting similarity for multi-source downloads using file handprints," in *Proc. 4th USENIX NSDI*, Cambridge, MA, Apr. 2007.