Peer-to-Peer Networking and Applications manuscript No.
(will be inserted by the editor)

Implications of the file names and user requested queries on Gnutella performance

Surendar Chandra · William Acosta

Received: date / Accepted: date

Abstract The Gnutella file sharing system allows a large number of peers to share their local files. However, it does not coordinate the way by which these shared objects are named or how they are searched by other users; such decisions are made independently by each peer. In this work, we investigate the practical performance implications of this design. We collected the shared filenames and user generated queries over a three-year period. We show the mismatch between these naming mechanisms. We show the fundamental limitations of Gnutella performance that cannot be addressed by improvements in overlays or by varying the search mechanisms alone. Based on our observations, we describe two practical approaches to improve Gnutella performance. We describe a mechanism to build the file term synopsis using the observed popularity of queries routed through the ultrapeer. We also describe a query transformation mechanism that improves the success rates for failed queries.

Keywords Gnutella · query · shared filenames · synopsis · query transformation

1 Introduction

Gnutella is a popular unstructured peer-to-peer (P2P) file sharing system; accounting for over 40% of all P2P users [4]. Gnutella users offer objects such as songs, movies and other documents for consumption by other Gnutella users. They also seek these shared objects from other Gnutella users.

Gnutella uses a distributed protocol. Each peer maintains network information about several other peers; recursively creating a P2P overlay without global coordination. The number of peers (node degree) maintained by any peer depends on its resource availability. Loo et al. [19] observed that ultrapeers using the LimeWire Gnutella client support between 30 and 75 leaf nodes and maintained overlay information for about 6 to 32 ultrapeer neighbors. Stutzbach et al. [27] also reported similar results. Also, peers frequently left the system without prior warning; several projects (e.g., [18,1]) developed overlays that were resilient to this node churn. Other work [3,11] developed overlays that reduced the amount of redundant paths in a distributed fashion.

Objects are accessed from other peers by first searching [29] for them using these P2P overlays. Gnutella ultrapeers search for objects by flooding all the neighbors. They use time-to-live (TTL) values to reduce the scope of the search query. Prior work [8] investigated mechanisms to automatically select these TTL values. Several projects (including [21,22]) improved the search mechanisms by reducing duplicate queries. Hybrid approaches [20,28,34] were also developed that first used non-scalable search mechanisms in the local neighborhood before using distributed hash table (DHT) based structured overlays for global search.

We focus our attention on another aspect of Gnutella that affects its real-world performance: the way that users name objects and how these shared objects are queried by users. Objects are independently named and queried by each peer without any global coordination. The Gnutella [17] protocol requires a match of all terms from the query against terms in the particular filename for a successful match; regular expressions are not supported. If many shared filenames matched typical user queries, then there is no need to investigate overlays that reach a large number of unique peers; a small number of peers can provide satisfactory query resolution. However, if there was a mismatch between the queries and the way objects were named, efficiently forwarding (from the network perspective) a particular query to all the peers will not by itself result in a successful query resolution.

First, we analyzed the shared filenames and user generated queries in Gnutella. For our analysis, we used our crawler to collect the names of shared objects: we collected 18.6M objects from 42K peers in October 2006, 21M filenames from 38K peers in April 2007 and 17M files shared

E-mail: surendar@acm.org · E-mail: william.acosta@utoledo.edu

by 34K peers in February 2008. During these data collection durations, we also used another modified Gnutella agent to collect between 200K to 230K queries per day that were routed through our data collection peer. We show that the query terms and filename terms exhibit a Zipf distribution. The long tail meant that over 99.5% of the objects were insufficiently replicated (less than 0.1% of the peers). Also, the relative popularity of a term in the filenames did not correlate with the term popularity in the queries. We observed that between 44.4% and 56% of the queries had no matching objects in the system regardless of the overlay or search mechanism used to locate the objects.

Based on these observations, we describe two mechanisms for improving Gnutella performance.

- Ultrapeers can distribute a synopsis of object terms from its leaf peers to other ultrapeers. These synopses are then used to decide whether to use the unstructured or structured component of hybrid P2P systems without further messages. We design a synopsis creation algorithm that dynamically adapts to the observed popular queries. For a 1,500 byte synopsis (1,250 terms in a Bloom filter representation), we show that our Query-Adaptive method improved the synopsis hit rates from 30% to about 60% (with a 3% false positive rate) over a file term only method.
- Queries fail because of a mismatch between the filenames and queries; we transform the failed queries to better match the objects available in the system. Our approach is relevant to the intent of the original query and uses the filenames from the leaf nodes of an ultrapeer. An overlay agnostic analysis shows that our transformation improves success rates from 44.4% to between 72.5% and 91.2%. Using our *Hybrid* mechanism, our transformation middleware subjectively produced relevant results for about 61% of the failed queries.

Next, Section 2 analyses the Gnutella shared filenames and user generated queries while Section 3 describes the implications of these observations on Gnutella performance. Based on our analysis, we describe two schemes to improve Gnutella performance in Section 4. We describe the related work in Section 5 and conclude in Section 6.

2 Analysis of objects names and queries

First, we analyze the aspects of the shared filenames and user generated queries that affect the query matching performance. We analyze the number of terms in queries and filenames as well as their popularity distribution. Though we collected Gnutella information over a three-year period, we only illustrate representative results for the rest of the paper.

2.1 Collecting information from Gnutella networks

Gnutella network is huge with significant node churn; it is difficult to collect information about all the online peers. Instead, we sampled the name of shared files from some peers that were online during our trace capture duration. We also collected queries that were routed through our modified Gnutella client. Next, we briefly describe the programs used to collect information about filenames and queries.

2.1.1 Capturing shared filenames

We developed a crawler to collect the name of shared files. Our program first performed a topology crawl to discover peers connected to the Gnutella network. We used the Gnutella *Crawler Ping* directive for fast crawls [13,27]. For each peer discovered by the topology crawl, we establish a direct connection and requested a list of its shared files using the *Browse Host* extension. The filenames were transmitted using an UTF-8 encoding. Some of the peers had already left the system between the topology crawl and listing request. Also, not all peers honor the request to list its shared filenames. We performed a crawl in October 2006 and discovered 18.6M objects from 41,910 peers. Our crawl in April 2007 discovered 21M objects shared by 37,572 peers while a crawl in February 2008 collected 17M files shared by 34K peers.

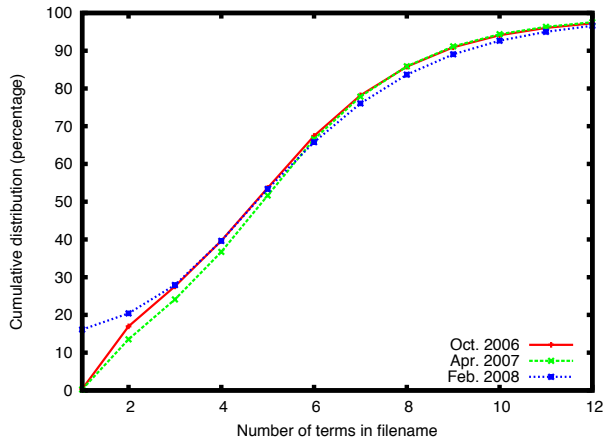
2.1.2 Capturing user generated queries

We modified the Phex (`http://www.phex.org/mambo/`) Gnutella client. Our client fully participated in the Gnutella network as an ultrapeer. All forwarded query messages were logged; our peer itself did not serve any documents. Queries were collected simultaneously with information about shared files in October 2006, April 2007 and in February 2008. The query traces correspond to about 200K queries per day.

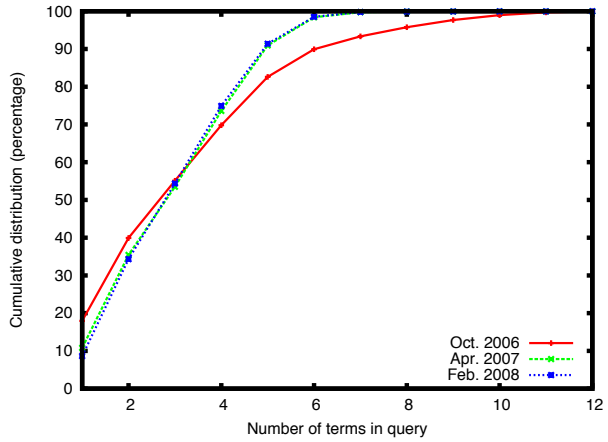
2.2 Number of terms in filenames and queries

First, we plot the cumulative distribution of the number of terms in the shared filenames as well as the user generated queries during the various trace durations in Figure 1. We tokenized the filenames and queries as specified by the Gnutella protocol [13]. Since Gnutella requires a match of all query terms against the filename, these numbers give an indication of the query success rates; a larger number of terms in the filename is likely to contain all the terms in the query.

From Figure 1(a), we observe that the number of terms in filenames were similar during the different data collection durations; 90% of the files had fewer than ten terms while the median number of terms was about five. During February 2008, about 15% of the files used just a single term; many of these files were data files (e.g., .jar, .exe files). The users need to know their exact names in order to access them.

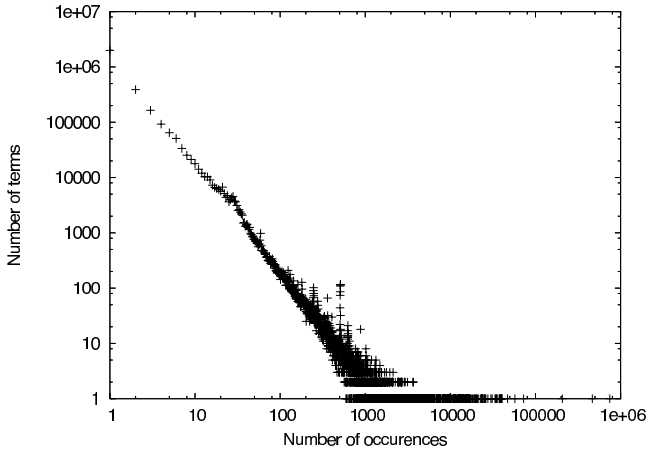


(a) Shared filenames

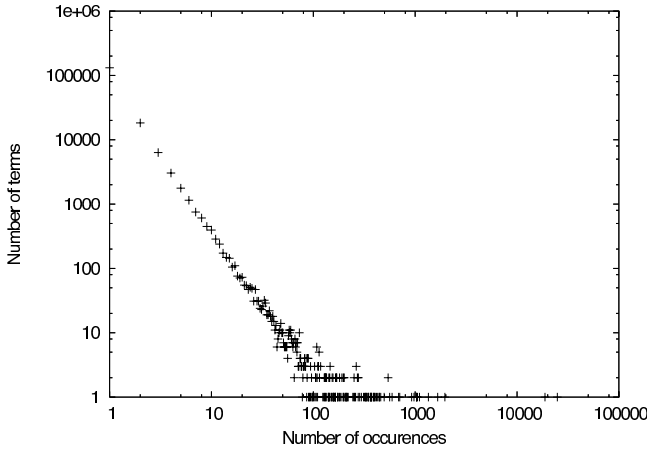


(b) User generated queries

Fig. 1 Distribution of number of terms



(a) Shared filenames



(b) User generated queries

Fig. 2 Term popularity distribution

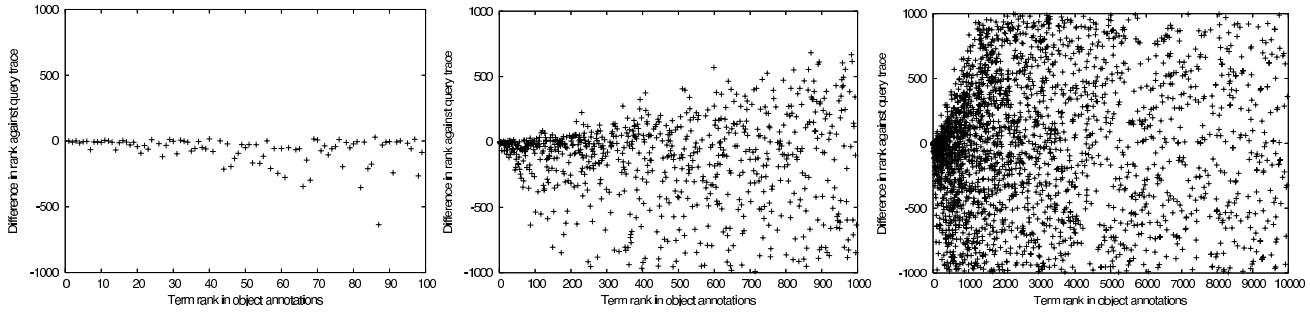
From Figure 1(b), we note that the median number of terms in the queries was about three in all the analysis durations. This compares well to the median number of five terms in filenames (Figure 1(a)). Overall, queries are becoming shorter: in October 2006, 81% of the queries contained five or less terms while the corresponding values in April 2007 and February 2008 were about 90%. Also, fewer queries have a single term in their queries: about 18% in October 2006 and about 10% in April 2007 and 8% in February 2008. For comparison, using Gnutella traces from May 2006, Zaharia et al. [33] reported that the average Gnutella object name and query had 8.47 terms and 3.91 terms, respectively. Yang et al. [32] reported a median of three and average of 3.74 query terms in July 2002 and a median of four and mean of 5.21 query terms by September 2003.

2.3 Popularity of terms in queries and filenames

Next, we analyze the popularity of terms in queries and filenames. In general, when the terms in the query and filenames were uniformly distributed and when the popularity rank of specific query terms match their ranks in filenames, we expect good query success.

For the traces from October 2006, we plot the cumulative distribution of the number of occurrences of terms in queries and filenames in Figure 2. The figures are plotted in a log-log scale. We observe a linear trend for the term popularity, indicating a Zipf distribution. This observation confirms prior assumptions that the popularity distribution of terms in queries and filenames might follow a Zipf distribution [20,34]. However, it differs from objects in the Kazaa network [15] which showed a significant deviation from Zipf like popularity.

Next, we investigate whether the popularity of specific terms in the filenames correlates to its popularity in the queries. Among the 233,000 queries, we discovered 83,000 unique query terms while among 18.6M files, there were 1.2M unique filename terms. We cross-referenced the list of query and filename terms. About 61,000 terms appeared in both queries and filenames. This represented about 75% of the query terms, but only 5% of the filename tokens. The majority of filename terms did not appear in any of the queries. This observation has significant implications for the synopsis generation mechanisms (Section 4.1).



(a) Top 100 object terms (b) Top 1,000 object terms (c) Top 10,000 object terms

Fig. 3 Difference between term rank in the filenames and queries

Next, we ranked query and filename terms based on their popularity. For the top 100, 1,000 and 10,000 filename terms, we plot the difference in rank between the query and filename in Figures 3(a), 3(b) and 3(c), respectively. A complete correlation between these measures mean that the same terms are similarly popular among the queries and filenames. Again, this metric has important implications on the synopsis generation (Section 4.1). We observe some correlation among the top 25 terms. However, less popular filename terms are uncorrelated with the query terms. Investigating the top 10,000 terms (Figure 3(c)) revealed a difference of -1,876 between the popularity rank of a term from the filename and its popularity rank among queries. Although, the term popularities followed a Zipf distribution (Figure 2), the relative popularity of a term in filenames did not correlate well with its popularity in the query terms.

2.3.1 Persistence of query term popularity

Finally, we investigate the temporal variations among popular query terms. We define two types of popularity: *overall popularity* (Q_t^p) which accounts for all prior occurrences of the term in the query trace at time t and *interval popularity* ($\hat{Q}_t^p$) which only counted the occurrences of the term in the current evaluation interval. Synopsis generation algorithms (Section 4.1) will recompute the synopsis during each of these intervals.

We used the Jaccard index to compare the similarity of the set of popular query terms to the previous set of popular query terms. Given two sets A and B, the Jaccard index of A and B is the ratio of the size of the intersection of A and B to the size of the union of A and B. The Jaccard index ranges between zero (dissimilar) and one (identical). We analyzed the set of 1,000 most popular query terms for one week of queries from April 2007 using a 60-minute evaluation interval. We computed $Jaccard(Q_t^p, Q_{t-1}^p)$ and $Jaccard(\hat{Q}_t^p, \hat{Q}_{t-1}^p)$ and plotted the results in Figure 4. After a short stabilization interval, the set of *popular terms* remained relatively constant with a high Jaccard similarity value of over 0.9. However, the set of *interval popular* terms were less similar with a Jaccard similarity value of about 0.4; many popular terms

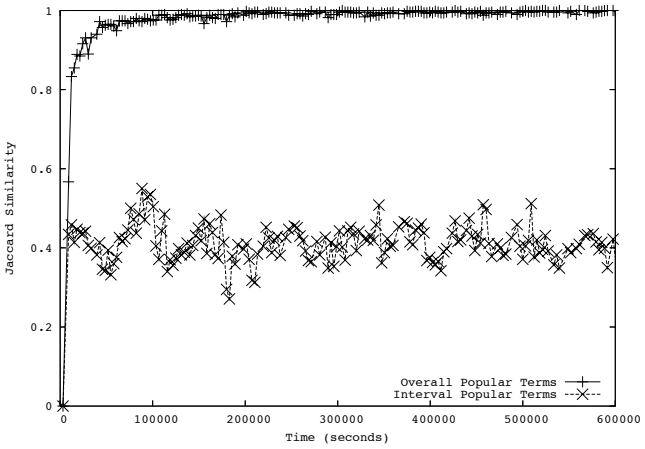


Fig. 4 Jaccard index of change in query term popularity

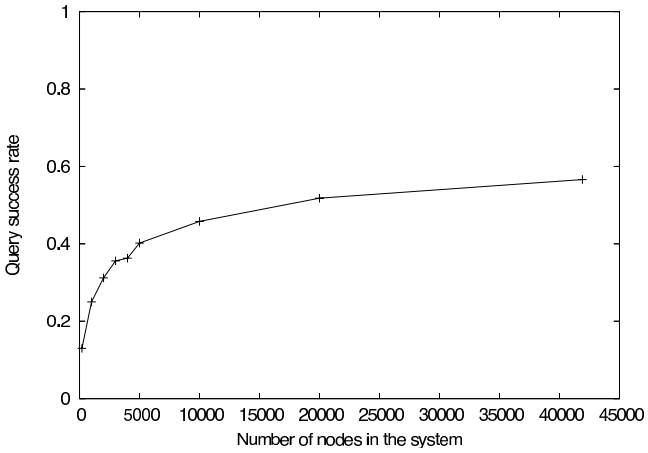
from one evaluation interval were unlikely to be popular in the next interval. In Section 4.1, our synopsis generation algorithm exploits these findings.

3 Implications on search performance

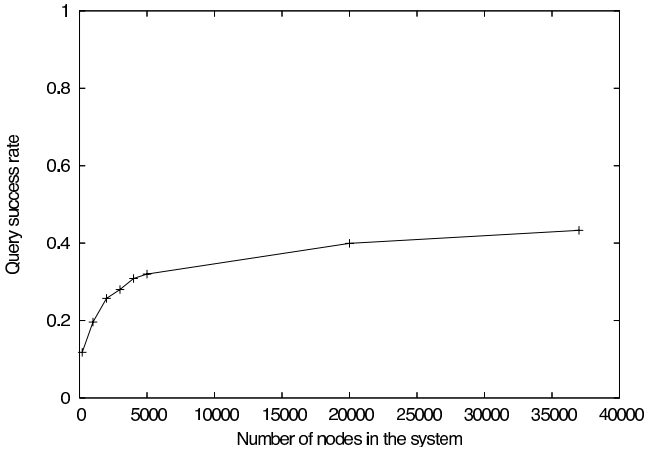
Next, we investigate the implications of our query and filename term observations on the Gnutella search performance. We used a query agnostic approach for our evaluation.

3.1 Query success rate

Query success rate is an important metric for evaluating P2P systems. Earlier [2], we analyzed the queries and responses that were routed through a monitoring client in order to quantify query success rates. We showed that between 2003 and 2006, the query success rate at a forwarding peer increased from 3.5% to 6.9%. Loo et al. [19] also analyzed the query success rates by injecting specific queries into the Gnutella overlay from various Planetlab locations. These analyses depended on the network overlay, query routing mechanisms as well as on the peers that were online during the evaluation.

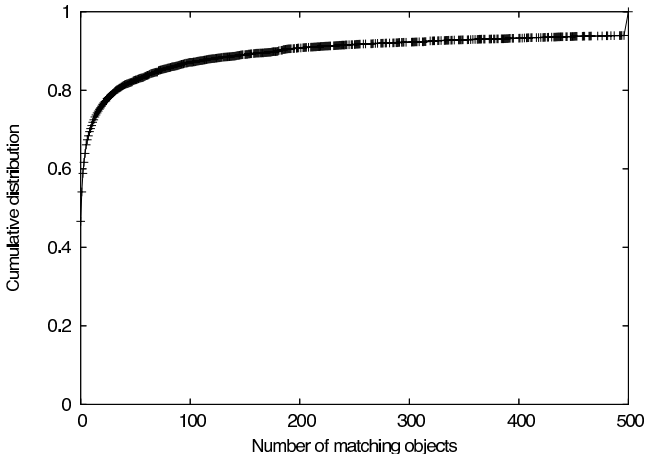


(a) October, 2006 Data

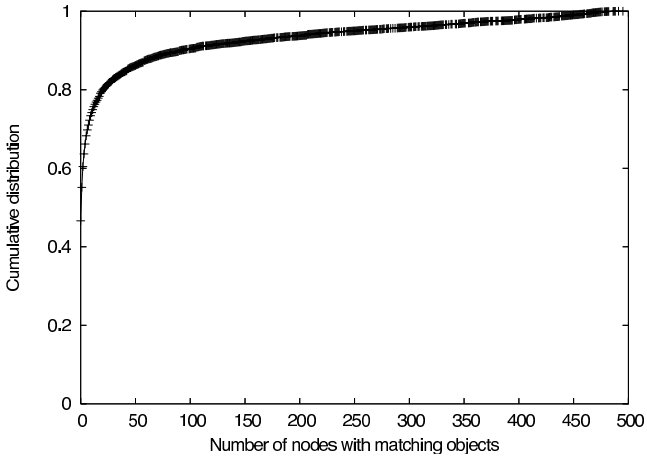


(b) April, 2007 Data

Fig. 5 Query success rates with nodes reached



(a) Number of objects per query



(b) Number of nodes

Fig. 6 Query matching with shared objects from October, 2006

Using the queries and filenames collected (Section 2), we performed an overlay agnostic evaluation of the query success rate. We evaluated the queries against filenames from varying number of peers. In a real deployment, the number of peers is dictated by the query TTL and the state of the particular overlay. We randomly chose peers groups of size 200, 1K, 2K, 3K, 4K, 5K, 10K, 20K and then all the peers from a particular trace. We evaluated a two hour trace of queries against all the filenames from the chosen peers. We reported the results using filenames captured in October 2006 and April 2007 in Figures 5(a) and 5(b), respectively. We observe that success rates improved when considering objects from a larger number of peers. However, the success rates did not improve significantly when using more than about 10,000 peers. Also, only 56% of the queries in 2006 and 44.4% of the queries in 2007 were resolved when using the shared objects from all the peers in our filename trace.

In general, the success rates for 2007 were lower than in 2006. This drop in success rate can be partially attributed to an increase in the number of queries with non-ASCII characters. The percentage of queries with non-ASCII characters increased from 8% in October 2006 to 17% in April, 2007. Analyzing Gnutella queries in December 2000 and January 2001, Sripanidkulchai [26] also observed that 17% of the queries contained non-ASCII characters. Similarly, Yang et al. [32] observed that 0.59% of queries issued in July 2002 were not in English. They also observed that by September 2003, 20.85% of the queries were not in English. Yet, our analysis of the shared filenames showed that the number of shared objects with non-ASCII characters in their filenames remained around 0.13%. An analysis of queries with non-ASCII characters against the full objects showed a success rate of only 0.08%; non-ASCII queries fail at a higher rate than English queries. In Section 4.2, we transform queries in order to improve the query matching performance.

3.2 Effective object replication ratios or object distribution

Object replication distribution is another important factor that affects the system performance; highly replicated objects are easier to search on the Gnutella networks [19].

One way to identify replicas is to consider objects that share the same name. For each object in the April 2007 trace,

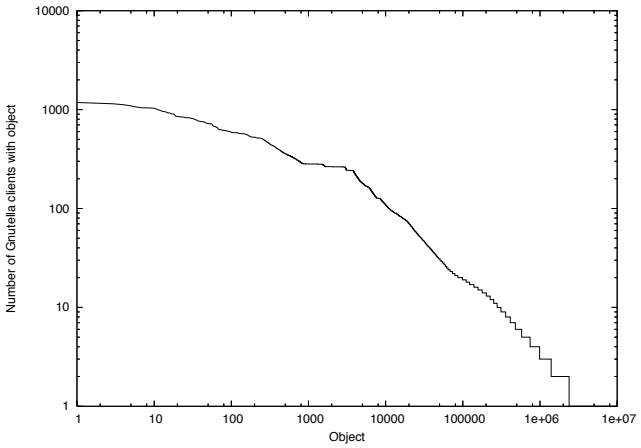


Fig. 7 Number of peers with same object name(April, 2007)

we plotted the number of replicas which contained files of the same name using a log-log scale in Figure 7. We note that the object distribution is linear in the log-log scale, signifying a Zipf distribution. A related analysis [7] of objects shared by iTunes users also exhibited a Zipfian distribution. About 5.7 million of the 8.1 million unique objects were available in a single peer or about 70.5% of the objects were not replicated in any of the 37,572 peers. Also, 99.5% of the objects were replicated in less than 0.1% (37) of the peers.

However, objects do not need to have the same name to be a replica. Next, we investigate the effective object replication ratios by measuring the number of matching objects for each query. We evaluated each query from a two hour representative interval against the full set of objects shared in October 2006 and plot the cumulative distribution of the number of matching objects as well as the number of peers with matching objects in Figure 6.

As reported in Section 3.1, 44% of the queries did not match any objects. From Figure 6(a), we observe that 88.3% of all successful queries had fewer than 500 matching files. Over 70% of successful queries were for objects with less than ten replicas (among 18.6M shared objects). The mean number of matching files per successful query was over 5,600 while the median was 18. Less than 1.4% of the successful queries had over 5,000 matching files. The mean was skewed high due to several queries with unusually large numbers of matching files. For example, single term queries such as "mp3" (over 4,000,000 matches), "zip" (over 800,000 matches), and "jpg" (over 700,000 matches) dominate the successful queries. The analysis of object traces from 2007 and 2008 yielded similar results (illustrated in Figure 11).

Next, for each query, we plot the cumulative distribution of the number of nodes with matching objects in Figure 6(b). 70% of the successful queries were resolved by fewer than ten nodes and over 90% of the queries were resolved by fewer than 100 nodes (among 42K nodes). Effectively, most Gnutella queries were for rare objects. Our synopsis mechanism incorporates these observations (Section 4.1).

4 Improving Gnutella performance

So far, we showed the implications of the observed queries and filenames on Gnutella performance. Next, we describe two mechanisms that leverage our understanding of queries and filenames to improve Gnutella performance. First, we describe a mechanism to create a synopsis that can improve the performance of a hybrid P2P search. Next, we focus on resolving failed queries by transforming them.

4.1 Improved synopsis for a hybrid P2P search

The flooding mechanism used by Gnutella is better suited for finding popular objects, while structured overlays are more efficient at locating rare objects. Hybrid P2P approaches [20, 28,34] leverage this observation to use a DHT based search for rare objects. Our analysis in Section 3.2 showed that many objects were poorly replicated for the user generated queries and hence are rare. Therefore, it is important to improve the performance of Hybrid P2P approaches.

Hybrid P2P approaches need to decide a priori the popularity of the objects that will match a particular query. Queries for popular objects are routed to the unstructured overlay while rare objects are looked up using the structured overlay. One way to know whether a query is for a popular object is to actually search a limited peer neighborhood using a TTL-limited flooding search. For rare objects, such a scheme can add significant search latency. Another approach is for each ultrapeer to maintain hints about objects stored in the local neighborhood. For example, the ultrapeer can collect terms from files shared by the leaf peers, create a *synopsis* using these terms and then share them with other ultrapeers. The other ultrapeers can locally match the query against a *synopsis* and decide whether to route the query to a particular ultrapeer without additional network messages. Note that Gnutella requires a match of all the query terms in a particular file. Hence, it is possible that a ultrapeer has different files that have all the individual query terms even though no single file had all the query terms; such a query will fail forcing the system to re-route the query to the structured overlay.

Bloom filters [5] are typically used to encode the chosen file terms into the *synopsis*. Since the synopsis is distributed to a large number of ultrapeers, they should be small in order to reduce the network overhead. As the synopsis size becomes constrained, the contents of the synopsis must be carefully chosen. Synopsis should include filename terms that will be popular in future queries. Prior approaches [34] considered the popular filename terms for the synopsis. However, our earlier analysis in Section 2.3 showed the mismatch between popular filename and query terms; choosing the popular file terms is not always the best option while constructing the *synopsis*. Hence, we describe our Query-Adaptive algorithm that chooses filename terms for the synopsis using queries routed through the ultrapeer.

4.1.1 Query-Adaptive algorithm to choose filename terms

Our algorithm is tuned more towards representing filename terms that will match popular queries. In Section 2.3.1, we showed that the overall popular query terms remained consistent while the interval popular query terms exhibited variations in popularity. Each ultrapeers tracks the queries that were forwarded through it in order to create a list of the overall popular query terms (Q_t^p), query terms which were popular in the current interval ($\hat{Q}_t^p$) as well as query terms which were popular in the prior interval ($\hat{Q}_{t-1}^p$). It also collects filename terms (F) from its leaf peers. Our algorithm chooses the overall query terms that matched the filename terms ($Q_t^p \cap F$). If there was still room for more terms in the synopsis, it chooses filename terms that matched the interval popular query terms that were also popular in the prior interval ($(\hat{Q}_t^p \cap \hat{Q}_{t-1}^p) \cap F$). These selected filename terms were encoded using a Bloom filter.

4.1.2 Performance evaluation

We examined the ability of our Query-Adaptive synopsis to effectively represent the content of a set of leaf peers. We selected 64 random peers from the April 2007 data to serve as leaf peers. We aggregated the set of files shared by these peers to create our synopsis. On average, these filenames yielded about 20,000 unique terms. We set the synopsis limit to 5,000, 2,500, and 1,250 terms. This represents 25%, 12.5%, and 6.25% of the unique filename terms from the leaf peers, respectively. Note that when using seven hash functions, synopses of 5,000, 2,500 and 1,250 terms can be represented by Bloom filters [5] of size 5,991, 2,995 and 1,498 bytes, respectively with a 1% false positive rate (`http://hur.st/bloomfilter`). Incidentally, the maximum packet size in typical Ethernet networks is 1,500 bytes; these Bloom filters can be sent in four, two or one packets, respectively.

From the query trace from April 2007, we selected queries that matched at least one file in our leaf peers. For these queries, we plot the true positive hit rate using our synopsis in Figure 8. We varied the synopsis recalculation interval from 15 minutes to one hour. We observed that a 5,000 term synopsis achieved 80% true positive hit rate. The synopsis evaluation interval did not significantly affect the performance of the synopsis when the limit was over 2,500 terms. For 1,250 terms, our synopsis achieved 60% true positive hit rate when the evaluation interval was 60 minutes. However, when the interval was reduced to 15 or 30 minutes, we achieved 40% true positive hit rate.

Prior synopsis construction approaches only considered the popular filename terms [34]. We plot the true positive hit rate for our approach and the filename popularity based approaches in Figure 9. For all term limit settings, our Query-Adaptive synopsis outperformed the traditional file popularity based synopsis. When the term limit was low (1,250 terms), our Query-Adaptive synopsis produced a true positive hit rate nearly double that of the filename-based synopsis. When the term limit was 5,000 terms, our synopsis had a narrower improvement in the true positive hit rate.

	15 min	30 min	60 min
1250 Terms	2.9%	2.8%	2.7%
2500 Terms	6.7%	6.6%	6.8%
5000 Terms	11.9%	11.7%	11.6%

Table 1 False positive rate for Query-Adaptive synopsis algorithm

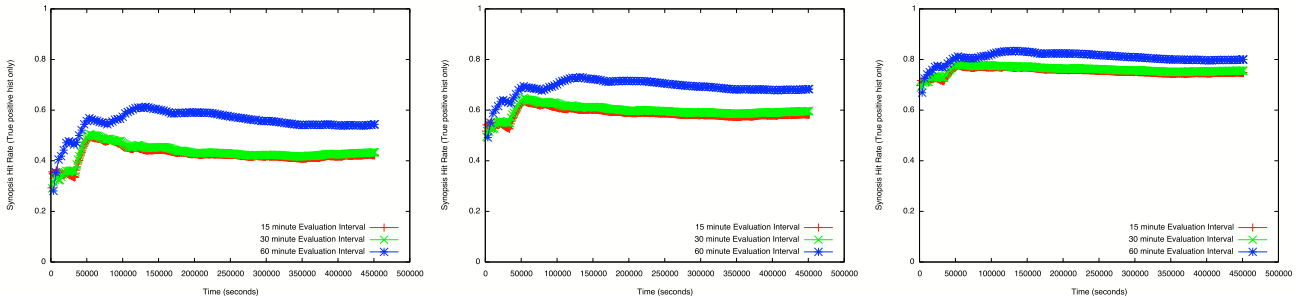
Next, we tabulate the false positive rates for all the queries in our trace in Table 1. For all evaluation intervals, the false positive rate increased as the synopsis term count increased. The reason for this is that the synopsis aggregated terms from multiple peers. All terms from a given query may have existed in files from the leaf set, but not necessarily in the same file. As the term count for the synopsis increased, the probability of this event occurring also increases. Nevertheless, the false positive rate for the synopsis was reasonable (11.9%) even when the term limit was relatively large (5,000 terms). When the term limit was 1,250 terms, the false positive rate was less than 3%. This suggests that a synopsis with a small term limit and a reasonable evaluation interval (60 minutes) can achieve a good true positive hit rate (60%) as well as a low false positive rate.

4.2 Query transformation for failed queries

Next, we address another area of poor performance in Gnutella; its query resolution. Gnutella [17] requires that all query terms be present in a particular file. One approach to improve search performance is to force users to standardize on the naming of shared filenames. This could allow other users to easily query for them. Earlier [7], we analyzed the annotations of objects shared by 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 even though iTunes itself shipped with just 24 genres. It is likely that Gnutella users will also want to customize the filenames; standardizing the way that files are named is not a viable option.

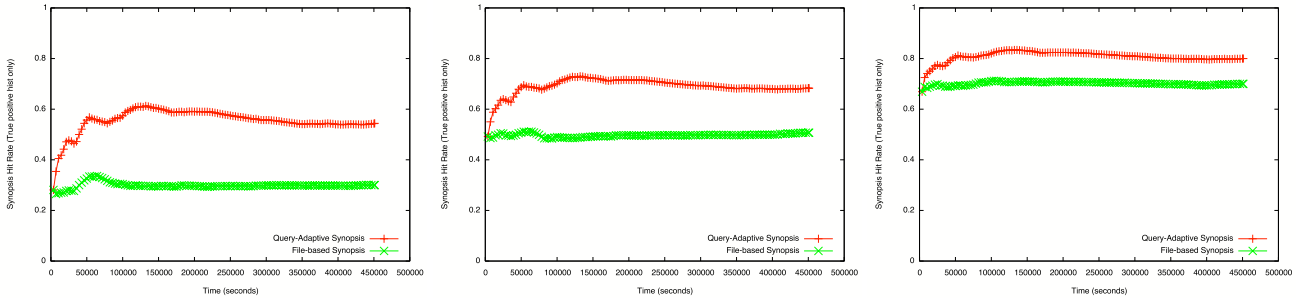
Another approach is to change the Gnutella matching semantics and successfully match files that contained any query keyword. For the failed queries from the 2007 data set, we plotted the number of files that matched queries using this policy in Figure 10. We observed that the median number was 24,233 matching files per query; this mechanism will exhibit low specificity.

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



(a) 1250 Terms in Synopsis (b) 2500 Terms in Synopsis (c) 5000 Terms in Synopsis

Fig. 8 True positive hit rate for Query-Adaptive synopsis



(a) 1250 Terms in Synopsis (b) 2500 Terms in Synopsis (c) 5000 Terms in Synopsis

Fig. 9 True positive hit rates of file popularity and Query-Adaptive

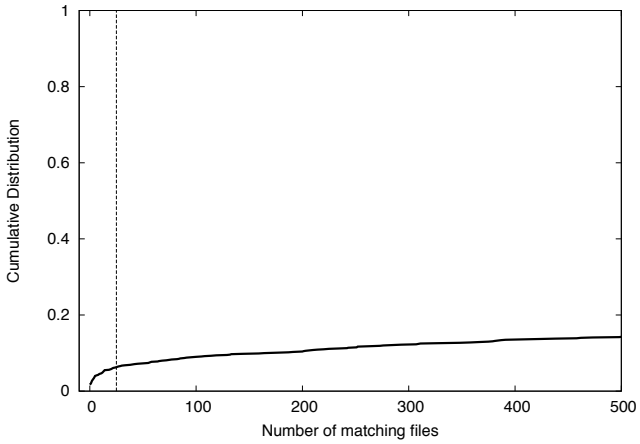


Fig. 10 Matching files for considering *any* keywords as a match

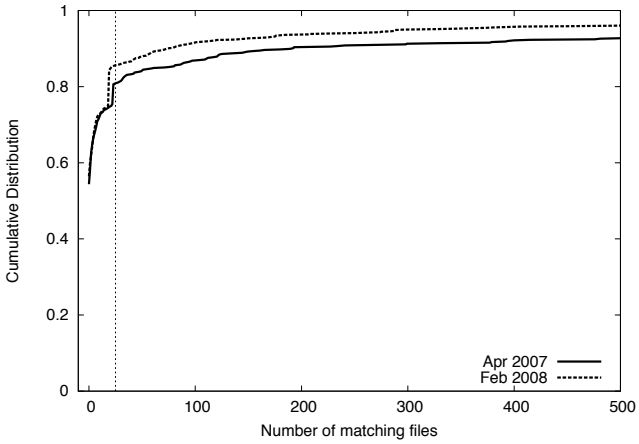


Fig. 11 Matching files for queries issued by Gnutella users. Queries were matched against shared objects from 37K peers (2007) and 34K peers (2008) and is overlay and search agnostic

requested by the original query. The challenges encountered in transforming failed queries are:

– **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*.

– **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 limit 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 11. The data sets from 2007 and 2008 showed that the median number of matching files was about twenty five. Hence, we choose transformations that match about twenty five files.

– **Practicality:** Finally, the transformation should be practical. Peers make independent decisions (regarding overlay maintenance, query routing etc.) without global coordination. Unlike our offline analysis of filenames collected from a large number of peers (Section 2), 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.

4.2.1 Our query transformation

We resolve failed queries by transforming them into queries that can successfully locate objects. We used filename information from the peer neighborhood. We consider three transformations: correct misspelt keywords, removing keywords as well as a hybrid approach.

1. **Correcting misspelt keywords:** Zaharia et al. [33] 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, in Section 3.1, we showed that 17% of the queries were issued in a multi-lingual character set. Sometimes, songs were named using slang terms (e.g., *Dat* instead of *That*) that were unlikely to be part of regular dictionaries. Also, query keywords and filename 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 become obsolete.
   For each of the query keywords that did not match any filename terms in the peer neighborhood, we correct misspelt keywords using Qgrams [14] 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 consider 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 4.2, 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 10, each keyword can match a large number of files, making such an approach impractical (transmitting a large response places a 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. [33] found an average of 3.91 keywords per query. Our analysis of queries (Section 2.2) showed that most (over 99%) of the queries contained less than six terms. Hence, we explore subsets of size (n) one, two and three.
3. **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.
4. **Popular:** This policy assumes that the presence of less popular file terms in the original query 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.
5. **CO-Popular:** This approach selects keywords that occurred together in filenames. We first compute the co-occurrence value between each pair of keywords in the original query among filenames. 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 filenames 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.
6. **Hybrid:** Finally, we applied the *Spell* followed by the *CO-Popular* approach to the original failed query in order to create the transformed query.

4.2.2 Results

We analyze the performance of *Spell*, *Random*, *Popular*, *CO-Popular* and *Hybrid* query transformations using the 2007 data set (Section 2). 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; Section 4.2.2.4 analyzes the scalability of this choice of 64 peers.

4.2.2.1 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 12. We used the dictionary information of filenames from a 64

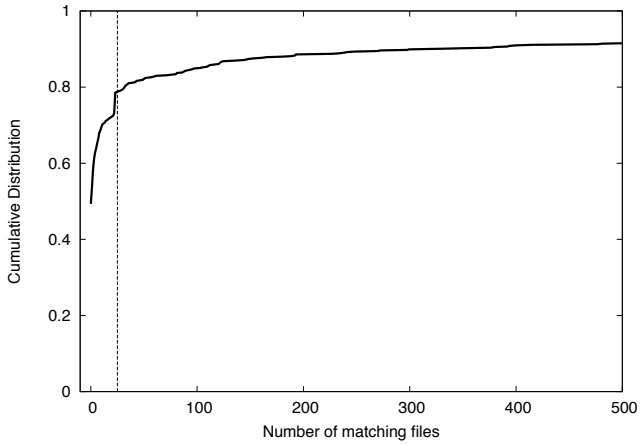


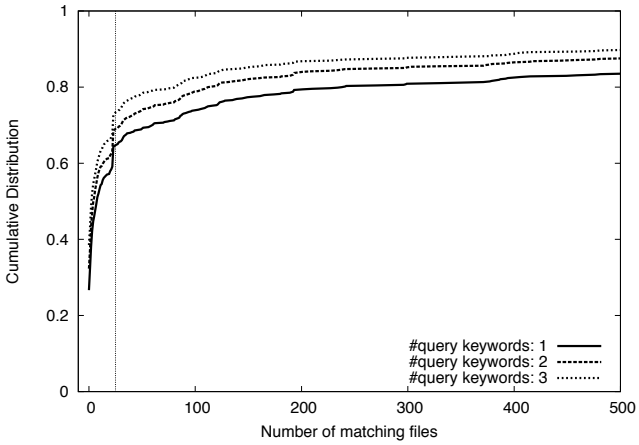
Fig. 12 Matching files for correcting misspelt keywords of failed queries

peer neighborhood. The results were similar to the number of matching files for the original query (Figure 11). About 50% of the failed queries continue to fail even with the transformation. 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. [33] reported an improvement of 51% of query success rates using an approximate matching algorithm.

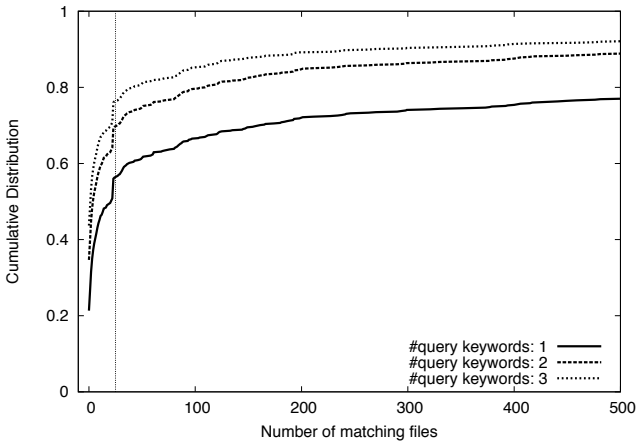
4.2.2.2 Query subset by removing keywords We analyze transformation mechanisms that remove keywords from failed queries. We varied the number of keywords in the transformed query between one and three. We plot the number of matching files for transforming failed queries using *Random*, *Popular* and *CO-Popular* in Figures 13(a), 13(b) and 13(c), respectively.

From Figure 13(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 filenames shared in Gnutella; matched files for the transformed query might not be relevant to the intent of the original query.

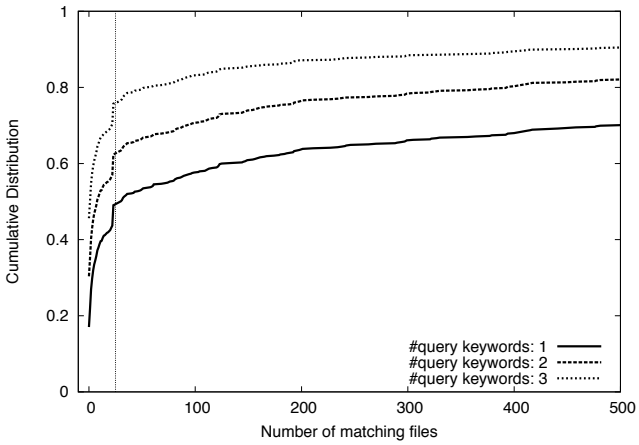
Next, we investigate the *Popular* transformation mechanism in Figure 13(b). We note that 21.33%, 34.66% and 43.70% of the failed queries were not resolved by the transformation of choosing one, two or three of the most popular keywords, respectively. This translates to matching 88.63%, 80.94% and 75.97% 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%,



(a) *Random*



(b) *Popular*



(c) *Co-Popular*

Fig. 13 Matching files for transforming failed queries

54.20% and 58.42% of the successfully transformed queries matched twenty five or less objects.

CO-Popular transformation (Figure 13(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

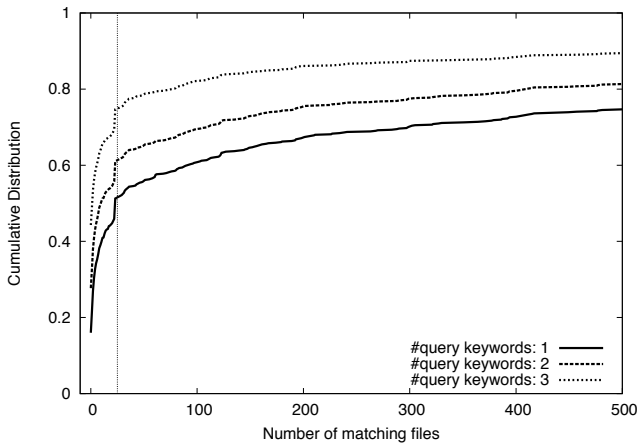


Fig. 14 Matching files for *Hybrid* transformation of failed queries

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.

4.2.2.3 Hybrid approach (Spell+CO-Popular) Finally, we analyze the behavior of a hybrid approach. From Figure 14, 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.

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 14 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 4.2.3.

4.2.2.4 Scalability analysis In the previous section, we chose a neighborhood size of 64 for the various query transformation mechanisms. Typical Gnutella ultrapeers maintained between 30 and 75 leaf peers [27]; 64 is a reasonable number of leaf peers. Hence, an ultrapeer node will already maintain overlay information for about 64 peers within a single hop on the P2P overlay. Our experiments showed that the list of shared filenames 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.

Table 2 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%

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 2. 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.

4.2.3 Query transformation middleware

So far, we maintained a large number of original keywords in the transformed query and showed that the number of matching files appear similar to the number of matching files for the original query (Section 4.2). 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 target our middleware at 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 filename. The weight of the edge corresponds to the number of times the terms co-occur in a filename. The ultrapeer also computes the Qgrams for each file term for the *Spell* component. In our earlier work [2], we showed that the success rate at the ultrapeer was low; about 6.9% of the forwarded queries yielded any query responses. 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.

4.2.3.1 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.

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 responses 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.

5 Related work

The ease of locating objects is a fundamental requirement of unstructured P2P file-sharing systems. Earlier work focused on improving Gnutella overlays that allowed search mechanisms to reach more newer peers while reducing the amount of network messages required to maintain the overlay. Law et al. [18] explored the use of expander graphs as overlays. Wouhaybi et al. [30] constructed low-diameter resilient topologies. Acosta et al. [1] described a practical overlay that exhibited expander like properties. Other efforts focused on improving the search mechanisms. Chang et al. [8] minimize the number of messages sent by the Gnutella flood mechanism. They describe two mechanisms: a dynamic programming mechanism that selected the query TTL when the probability distribution of the object locations was known and a randomized algorithm when the distribution was unknown. Lv et al. [21] describe a random walk that forwarded the queries to one randomly selected neighbor. This approach traded the number of query messages for the response time required to locate objects. Rhea et al. [25] reduced the number of query messages while maintaining a low response time. They used probabilistic routing mechanisms that routed queries using a potential function that estimated routes which lead to matching contents. Each peer maintained a compact summary of shared files in the peer neighborhood in a bloom filter. Yang et al. [31] presented a number of techniques to improve P2P searches. They used 500 random queries from a collection of 500,000 queries collected in May 2001 to validate their system. Chawathe et al. [9] employed one-hop replication, topology adaptation and a search mechanism based on random walks to improve the search performance of Gnutella. We address a related problem of understanding how end users named and accessed objects. Our observations impact the behavior of the overlay generation and search mechanisms. On the other hand, our approach can fully exploit any advances in overlay maintenance or search mechanisms.

Chawathe et al. [9] captured Gnutella query traffic and offered for download the objects that matched the top 50 queries. Using these objects, they showed that more than 50% of the requests were for objects with more than 100 replicas and approximately 80% of the requests were for files with more than 80 replicas. This analysis approach can evaluate the popularity of the objects matching the popular queries. However, it does not give an insight to the popularity of all the objects with respect to the query workload. Our

analysis examined the distribution of matching objects for each query and found that only 17% of the queries had more than 100 matching objects.

Loo et al. [20] motivated the need for hybrid P2P approaches using captured Gnutella query traffic. By capturing traces at 30 Gnutella ultrapeers, they contained information about 670K objects and 230K queries. They selected 700 successful queries from these traces and replayed them from several different locations and times in order to analyze the Gnutella search performance. They showed that 6% of queries received no results. Since only queries with successful responses were selected to be replayed, the 6% of queries with no results represent a failure of Gnutella to locate the desired objects. Their analysis does not take into account the fact that some queries may have no matching objects in the system. By contrast, our approach to evaluating query performance is independent of the P2P overlay or search mechanism used and thus provides a more complete representation of query behavior.

Fessant et al. [12] analyzed the eDonkey network to show that the objects exhibited a Zipf distribution. Similarly Zhao et al. [35] illustrated that Gnutella objects exhibited a Zipf distribution. Yang et al. [32] captured Gnutella queries in July 2002 and September 2003. They observed similar behavior to our query data. They observed higher instances of non-English characters in queries. Their data collection location in Asia might affect this parameter.

Hybrid P2P systems [20,28,34] improve search performance by leveraging both structured and unstructured mechanisms. Chen et al. [10] weighed the query terms in order to compute a difficulty rank for the query. Easy queries were flooded while hard queries were routed to a structured DHT. They distribute statistics about relative popularity of terms in the shared files to all peers. They use these statistics to compute the difficulty rank of each term. Similarly, Zaharia et al. [34] collected global statistics about popular file terms and used these statistics to decide the search mechanism. Several projects use the synopsis to obtain one-hop search latency for queries that matched against the synopsis. For example, Cai et al [6] pro-actively distributed the synopsis. Peers store these synopses to achieve a one-hop search latency to a particular advertising peer. Our approach considered the popularity of query terms when creating the synopsis. Our approach yielded synopses that described the relevant content at each peer better than file-based approaches.

Qiao et al. [24] evaluated the search performance of unreplicated objects. In order to ensure that no other replicas existed in the system, they synthetically created objects with 50-byte long random string filenames and inserted them into the Gnutella network. They found that Gnutella achieved only a 1.5% success rate for finding these objects. In practice, an object may have few replicas and yet might match many queries. For example, an object named "britney-spears-live-concert-for-my-cousins-birthday.mp3" is likely a unique object. However, searches for "britney spears concert", "cousins birthday" etc. will match this object. We examined actual queries issued by users in order to determine if the query targeted a rare or popular object.

Pucha et. al. [23] examined the byte-level similarity of files shared in BitTorrent. They found byte-level similarities even though the filenames did not indicate that the objects were identical. They exploited this similarity to improve BitTorrent performance. We only consider the original terms in the object names but transform queries in order to improve their matching performance. Jia et al. [16] used the file description from other peers in order to enhance the local annotation to improve the P2P search mechanisms. We consider the original filename annotations while transforming the queries themselves to match the relevant objects.

Audio search services such as Shazam (`shazam.com`) and Midomi (`midomi.com`) use techniques such as Multimodal Adaptive Recognition System (MARS) to create finger prints of songs. In a traditional Gnutella network, users request objects using keywords. These keywords might not match the standard annotation of these finger prints. An extension of our work can transform queries to locate finger prints of relevant objects.

Zaharia et al. [33] showed that many of the query keywords and file terms were misspelt by checking them against standard dictionaries. We showed that 17% of the queries were issued in a multi-lingual character set. Songs were also officially named using slang terms. These terms were unlikely to be part of regular dictionaries. We used the file terms from the peer neighborhood for our dictionary.

6 Conclusion

Gnutella networks allow the peers to independently decide on their naming and overlay maintenance mechanisms. This flexibility has important impact on the global system behavior. In this article, we empirically analyzed the way that typical peers were naming objects and the way in which they were requesting objects from other peers. We show a fundamental mis-match which affects the performance of prior efforts that focused on the overlay and search mechanisms. We present two techniques that can improve search performance. First, we change the way that filename terms were chosen for a synopsis. We also transform queries to better match the way objects were named. Users are likely to share these same files using other P2P mechanisms and search for them using similar queries; our observations might be applicable to other P2P applications as well.

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

References

1. Acosta, W., Chandra, S.: Improving search using a fault-tolerant overlay in unstructured P2P systems. In: IEEE ICPP '07, p. 5. Xian, China (2007). DOI http://doi.ieeecomputersociety.org/10.1109/ICPP.2007.48
2. Acosta, W., Chandra, S.: Trace driven analysis of the long term evolution of gnutella peer-to-peer traffic. In: Passive and Active Measurment Conference (PAM'07) (2007)
3. Adamic, L.A., Lukose, R.M., Puniyani, A.R., Huberman, B.A.: Search in power-law networks. Physical Review E **64** (2001)
4. Bangeman, E.: Study: Bittorrent sees big growth, limewire still ♯1 p2p app. Ars Technica (2008). URL `arstechnica.com/old/content/2008/04/study-bittorren-sees-big-growth-limewire-still-1-p2p-app.ars`
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970). DOI http://doi.acm.org/10.1145/362686.362692
6. Cai, H., Gu, P., Wang, J.: ASAP: An advertisement-based search algorithm for unstructured peer-to-peer systems. In: IEEE ICPP'07. Xian, China (2007)
7. Chandra, S., Yu, X.: An empirical analysis of serendipitous media sharing among campus-wide wireless users. ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMCCAP) **7**(1), 23 (2011)
8. Chang, N., Liu, M.: Revisiting the TTL-based controlled flooding search: optimality and randomization. In: Mobicom '04, pp. 85–99 (2004). DOI http://doi.acm.org/10.1145/1023720.1023730
9. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making gnutella-like p2p systems scalable. In: SIGCOMM '03, pp. 407–418 (2003). DOI http://doi.acm.org/10.1145/863955.864000
10. Chen, H., Jin, H., Liu, Y., Ni, L.M.: Difficulty-aware hybrid search in peer-to-peer networks. In: IEEE International Conference on Parallel Processing (ICPP'07) (2007)
11. Cohen, E., Shenker, S.: Replication strategies in unstructured peer-to-peer networks. In: SIGCOMM 2002, pp. 177–190 (2002). DOI http://doi.acm.org/10.1145/633025.633043
12. Fessant, F.L., Handurukande, S., Kermarrec, A.M., Massouli, L.: Clustering in peer-to-peer file sharing workloads. In: IPTPS'04. San Diego, CA (2004)
13. The gnutella protocol specification v0.6. `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`
14. Gravano, L., Ipeirotis, P.G., Jagadish, H., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: Very Large Data Bases (VLDB '01). Rome, Italy (2001)
15. Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 314–329 (2003). DOI http://doi.acm.org/10.1145/945445.945475
16. Jia, D., Yee, W.G., Nguyen, L.T., Frieder, O.: Distributed, automatic file description tuning in peer-to-peer file-sharing systems. In: IEEE P2P '07, pp. 167–176 (2007)
17. Klingberg, T., Manfredi, R.: Gnutella 0.6. `rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html` (2002)
18. Law, C., Siu, K.Y.: Distributed construction of random expander networks. In: IEEE INFOCOM'03, vol. 3, pp. 2133–2143 (2003)
19. Loo, B.T., Hellerstein, J.M., Huebsch, R., Shenker, S., Stoica, I.: Enhancing p2p file-sharing with an internet-scale query processor. In: Very Large Data Bases Conference (VLDB'04), pp. 432–443 (2004)
20. Loo, B.T., Huebsch, R., Stoica, I., Hellerstein, J.M.: The case for a hybrid p2p search infrastructure. In: IPTPS '04, pp. 141–150. San Diego, CA (2004)
21. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: International conf. on Supercomputing, pp. 84–95 (2002). DOI http://doi.acm.org/10.1145/514191.514206
22. Massoulie, L., Merrer, E.L., Kermarrec, A.M., Ganesh, A.: Peer counting and sampling in overlay networks: random walk methods. In: PODC '06, pp. 123–132 (2006). DOI http://doi.acm.org/10.1145/1146381.1146402
23. Pucha, H., Andersen, D.G., Kaminsky, M.: Exploiting similarity for multi-source downloads using file handprints. In: Proc. 4th USENIX NSDI. Cambridge, MA (2007)
24. Qiao, Y., Bustamante, F.E.: Structured and unstructured overlays under the microscope: a measurement-based view of two p2p systems that people use. In: USENIX Annual Technical Conference, pp. 31–31 (2006)
25. Rhea, S.C., Kubiatowicz, J.: Probabilistic location and routing. In: IEEE INFOCOM '02, vol. 3, pp. 1248–1257 (2002)
26. Sripanidkulchai, K.: The popularity of gnutella queries and its implications on scalability. `http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html` (2001)
27. Stutzbach, D., Rejaie, R., Sen, S.: Characterizing unstructured overlay topologies in modern p2p file-sharing systems. IEEE/ACM Trans. Netw. **16**(2), 267–280 (2008). DOI http://dx.doi.org/10.1109/TNET.2007.900406
28. Tang, C., Dwarkadas, S.: Hybrid global-local indexing peer-to-peer information retrieval. In: NSDI'04 (2004)
29. Tsoumakos, D., Roussopoulos, N.: Analysis and comparison of P2P search methods. In: ACM InfoScale '06, p. 25 (2006). DOI http://doi.acm.org/10.1145/1146847.1146872
30. Wouhaybi, R.H., Campbell, A.T.: Phenix: Supporting reilient low-diameter peer-to-peer topologies. In: IEEE INFOCOM'04, vol. 1, p. 119 (2004)
31. Yang, B., Garcia-Molina, H.: Improving search in peer-to-peer networks. In: IEEE ICDCS '02, p. 5 (2002)
32. Yang, C.C., Kwok, J.S.: Changes in queries in gnutella peer-to-peer networks. J. Inf. Sci. **31**(2), 124–135 (2005). DOI http://dx.doi.org/10.1177/0165551505050789
33. Zaharia, M.A., Chandel, A., Saroiu, S., Keshav, S.: Finding content in file-sharing networks when you can't even spell. In: IPTPS'07. Bellevue, WA (2007)
34. Zaharia, M.A., Keshav, S.: Gossip-based search selection in hybrid peer-to-peer networks. In: IPTPS '06. Santa Barbara, CA (2006)
35. Zhao, S., Stutzbach, D., Rejaie, R.: Characterizing files in the modern gnutella network: A measurement study. In: Proceedings of SPIE/ACM Multimedia Computing and Networking, vol. 6071. San Jose, CA (2006)