# Distributed, Automatic File Description Tuning in Peer-to-Peer File-Sharing Systems

Dongmei Jia Wai Gen Yee Linh Thai Nguyen
*Illinois Institute of Technology*
*Chicago, IL 60616, USA*
*{jia, waigen, linhnt}@ir.iit.edu*

Ophir Frieder
*Georgetown University*
*Washington, DC 20057, USA*
*ophir@cs.georgetown.edu*

## Abstract

*Peers in peer-to-peer file-sharing systems cannot effectively share their files if they are poorly described. Terms one user employs to describe an instance of a file may not be those that are commonly associated with the file, making this instance difficult to locate. To alleviate this problem, a server can ask its peers for help in improving the description of files they have in common. We consider the design of a fully distributed, automatic system for the exchange of descriptive metadata. Experimental results show that the proposed techniques are effective in improving search accuracy with reasonable cost.*

## 1. Introduction

Peer-to-peer (P2P) file-sharing systems such as Limewire's Gnutella [1] and eMule's eDonkey client [2] are popular with millions of users sharing several petabytes of data daily [3]. Because of the scale of these systems, it is imperative that their search functionalities are accurate and efficient. Accurate search functionality, furthermore, relies on accurate data descriptions.

In these systems, each peer maintains its own repository of files identified by user-tuned descriptors. A descriptor is a metadata set which is composed of descriptive terms. Arriving queries are compared with these descriptors and if any descriptor contains all of the query terms, then a *match* occurs, and the descriptor and server identifier are returned to the client [4]. This query processing technique works well if the descriptors contain the terms necessary for relevant queries to match them.

Poorly described files, however, match few queries with several consequences: users are unable to share their content; some peers unintentionally develop a poorer "reputation" for sharing in the network; load among peers is unbalanced; query volume increases due to the high rate of failed queries; and fewer files overall are available to the participants.

This is a significant problem. Recent traffic analyses indicate that 91% of Gnutella messages are query messages and only 1% of them are query hit messages [5]. This indicates that almost no queries return results. For example, Figure 1 shows some of the results for the query "*Mozart clarinet*", issued during an arbitrary summer 2006 day using eMule. From the results, we see that the query, "*Infantil Mozart for Babies clarinet*," would not return the third result (i.e., the one with File ID 21295…) even though it likely refers to it.

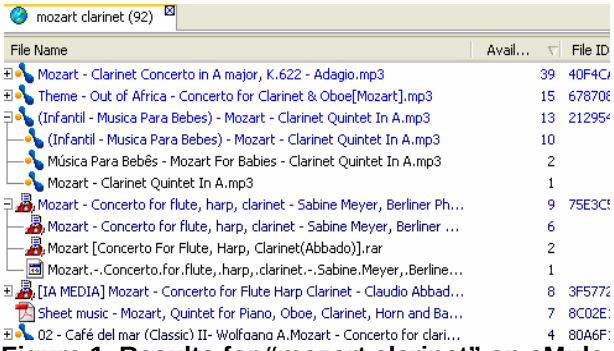


**Figure 1. Results for "mozart clarinet" on eMule.**

To address this problem, we propose a peer subsystem that automatically detects if a peer has a lower than desired level of system participation. In this event, the peer probes other peers' repositories for information that may be useful in improving local file descriptions, thereby improving its ability to match relevant queries.

We consider the design of the probing system and address the following issues:

- When to probe – how to determine the peer's participation level and use it to trigger a probe;
- What file to probe – how a peer selects a local file for which to conduct the probe;
- What should be done with probe results – how to use probe results to tune the probed file's descriptor.

## 2. Related Work

Query accuracy in P2P systems improves by increasing the number of relevant results (higher *recall*) or by decreasing the number of irrelevant results (higher *precision*) in a result set. These goals are typically accomplished by either query routing or query transformation techniques.

Query routing techniques take the form of either creating distributed indices that help locate shared data or by creating network topologies where peers with common interests are logically linked closely to each other in an

overlay network [6][7][8][9][10]. In the former case, the client must combine one or several indices that could be located on several peers for the locations of relevant content. In the latter case, queries are routed first to peers that are most likely to share relevant content.

Limewire's Gnutella attempts to improve the recall of hard-to-find files by increasing the time-to-live of queries that yield few results [1].

Index- and topology-building techniques assume that the desired data are accurately described. In P2P file-sharing systems, however, *user-tuned* data descriptions could be sparse or distributed over several peers, and only the aggregate description can appropriately match queries. Our proposed work can be viewed as an attempt to aggregate these descriptions as needed. Another problem with index- and topology-building techniques is that they often require reliable, DHT-based networks or centralized servers to function appropriately.

There is also a class of work that strives to improve search in P2P environments that share text documents [7][8]. Such systems generally do not suffer from poor data description because query matching is based on comparisons with a file's textual content (data are self-describing), not on its small, user-defined descriptors.

Our approach bears some resemblance to work that either adds or replaces query terms to improve query accuracy [11]. However, in practice, there exists no standard mechanism (e.g., standard ontologies) for implementing these techniques, and it is unclear what impact this would have on query performance or cost. Moreover, because data descriptions are sparse in the first place, term transformation may result in unacceptable "semantic drift" – clearly incorrect transformations in descriptions. Another recent work improves query performance by removing terms from queries [12], but the resultant smaller query still needs to match with the terms in relevant descriptors.

We improve automatically query accuracy by tuning the description of shared data in a P2P environment. This transformation is possible and necessary due to the fact that, in P2P file-sharing systems, the data are not self-describing; so locating a file is wholly dependent on a user's ability to accurately describe it. Our system aims to supplement user descriptions.

Note that this work can also be more widely applied to the description of binary files in centralized environments, such as photo- or video- sharing Web sites, whose search capabilities also rely on user descriptions.

## 3. Query Processing Specification

In typical P2P file-sharing systems (e.g., Gnutella) peers collectively share a set of (binary) files by maintaining local replicas of them. Each replica is represented by a user-tuned descriptor, which also contains an identifying **key** (e.g., an MD5 of SHA-1 hash on the file's bits). All replicas of the same file naturally share the same key. A client's query is routed to all reachable servers until its time-to-live expires. A server compares each query to its local descriptors; a query *matches* a replica if the replica's descriptor contains all of the query's terms. On a match, the server returns its system identifier and the matching replica's descriptor. This information allows the client to distinguish and, if desired, download the associated file.

Formally, let $O$ be the set of files, $M$ be the set of all terms, and $P$ be the set of peers. Each file $o_i \in O$ has a key associated with it, denoted $k_i$, such that $k_i = k_j$ if and only if $o_i = o_j$.

Associated with each file $o_i$ is a set of terms, $T_i \subseteq M$, that *validly* describe it. Each term $t \in T_i$ has a strength of association with $o_i$, denoted $soa(t, o_i)$, where $0 \leq soa(t, o_i) \leq 1$ and $\sum_{t \in Ti} soa(t, o_i) = 1$. The strength of association $t$ has with $o_i$ describes the relative likelihood that it is used to describe $o_i$, assuming that all terms are independent. The distribution of *soa* values for a file $o_i$ is called the *natural term distribution* of $o_i$. Intuitively, an average person will describe $o_i$ with terms from $T_i$ with a distribution described by $o_i$'s natural term distribution.

A peer $p_j \in P$ is defined as a pair, $(R_j, g^j)$, where $R_j$ is the peer's set of replicas (i.e., its local repository) and $g^j$ is its unique identifier (e.g., its IP address). Each replica $r^i_j \in R_j$ is $p_j$'s copy of file $o_i \in O$ and has an associated locally maintained descriptor, $d(r^i_j) \subseteq M$, which is a multiset of terms. Each descriptor $d(r^i_j)$ also contains $k_j$, the key of file $o_i$. The maximum number of terms that a descriptor can contain is fixed.

A query $Q^i \subseteq T_i$ for file $o_i$ is also a multiset of terms. The terms in $Q^i$ follow $o_j$'s natural term distribution. When $Q^i$ arrives at a server $p_j$, the server returns *result set* $U^i_j = \{(d(r^i_j), g^j) \mid r^i_j \in R_j \text{ and } Q^i \subseteq d(r^i_j) \text{ and } Q^i \neq \emptyset\}$: membership in the result set requires that a result's descriptor contain all query terms, in accordance with the matching criterion.

The client that issues $Q^i$ receives result set $U^i = \cup_{pj \in P} U^i_j$, and groups individual results by key, forming $G = \{G_1, G_2, \ldots\}$, where $G_i = (d(G_i), k_i, l_i)$. $d(G_i) = \{\oplus d(r^i_j) \mid (d(r^i_j), g^j) \in U^i\}$ is the group's descriptor and is the multiset sum of all of the descriptors of the results contained in $G_i$. $k_i$ is the key of $o_i$ and uniquely identifies $G_i$. $l_i = \{g^j \mid (d(r^i_j), g^j) \in U^i\}$ is the list of servers that returned the results in $G_i$. In this definition, $\oplus$ denotes the multiset sum operation.

The client assigns a rank score to each group with function $F_i \in F$, defined as $F$: $2^M \times 2^M \times Z \times Z \rightarrow R^+$. If $F_i(d(G_j), Q, |G_j|, time_j) > F_i(d(G_k), Q, |G_k|, time_k)$, where $G_j$, $G_k$ are groups, then we say that $G_j$ is ranked higher than $G_k$ with respect to query $Q$. In these definitions, $|G_j|$ is the number of results contained in $G_j$ and $time_j$ is the creation time of the $G_j$ (i.e., the time when the first result in $G_j$ arrived at the client).

In commercial P2P file-sharing systems, such as various implementations of the Gnutella protocol, ranking is performed by *group size*, as a large group suggests relevance to a query and multiple sources can better ensure a quick, successful download:

$$F_G(d(G_j), Q, |G_j|, \text{time}_j) = |G_j|.$$

Descriptors in these systems are generally implemented via filenames, but a small amount of descriptive information may be embedded in the actual binary of the replica (e.g., ID3 data embedded in mp3 files [13]). Furthermore, when a file is downloaded, the descriptor of this new replica is initialized as a duplicate of one of the servers' in the result set, but can be tuned by the user as well.

The screenshot in Figure 1 exhibits the conjunctive matching criterion, result grouping by key ("File ID"), and ranking by group size ("Avail").

# 4. Probing

The goal of probing is to increase automatically a peer's *participation level* in the system, defined as the rate at which its shared files match incoming queries, to a level commensurate to both the user's desire and the perceived activity level in the network. Furthermore, this should be done in a way that is functional in a dynamic, unreliable P2P environment – specifically, the information needs of the system need to be minimal.

## 4.1. Implementation of probe queries

In the probing system, there are *file queries* and *probe queries*. A file query for a file $o_i$ is issued by a user and is composed of terms that s/he associates with $o_i$. A probe query for file $o_i$ is issued by the probing system and is a request to other peers for descriptor information on $o_i$. To unambiguously identify $o_i$, the probe query need only contain $k_i$, the key of $o_i$.

## 4.2. Steps in probing

There are three steps to probing: 1) First, some mechanism must trigger a peer's probe. 2) The peer must then select a local file to probe. 3) Probe results are then used to improve the description of the probed file.

**4.2.1. Probe triggering.** A peer should trigger probes at a rate commensurate to the user's desire and system activity levels. A peer's desire to share is modeled by $N_f$, the number of files in its share repository. A peer's matching level is modeled by $N_r$, its number of results that match incoming queries. System activity is modeled by $N_q$, the number of queries the peer has received. We integrate these factors in the following **triggering condition**: given a user-defined *desired* participation level $T$, if the following triggering condition holds, the peer performs a probe:

$$T > N_r / (N_f N_q) + N_p T,\ N_f,\ N_q > 0 \qquad (1)$$

where $N_p$ in the second term on the right hand side is the number of probes the peer has already issued. The first term on the right hand side of the triggering condition models a peer's *actual* participation level and can be understood as *the number of responses per shared file per incoming query*. When this value is 0, then the peer returns no responses, and when it is 1, the peer returns responses for every query and every file. Therefore, setting $T$ to 0 indicates that the client should never probe and setting $T$ to 1 indicates that a client should virtually always probe. Note that if $N_f = 0$ (i.e., the peer is a freeloader) or $N_q = 0$ (i.e., there is no activity in the system), then the triggering condition should naturally not be evaluated. Updates to the "$N$" counters and evaluation of the triggering condition are performed after each incoming query is processed.

The actual participation level models several sensible criteria. If the peer is busy (i.e., $N_r$ is high), then it is more difficult to trigger a probe as the peer does not wish to take on more work. If the system is busy (i.e., $N_q$ is high), then the rate of probing should increase commensurate with the perceived demand of other peers. If the peer does not have many files to share (i.e., $N_f$ is low), then it should not probe much as its rate of responses should be proportionately low.

The second term on the right hand side of the triggering condition is used to "reset" the condition after each probe. This ensures that a peer does not continually issue probes if one does not have an immediate impact. Alternative resetting techniques are possible, of course. For example, a peer could increase the threshold $T$ after all local files have been probed. The one in the triggering condition is designed for simplicity.

Note that the triggering condition requires only local variables and no global information. This is consistent with the design goal of the probing system being functional in a fully distributed P2P environment.

**4.2.2. Selecting a file to probe.** The choice of file for probing is made to increase a peer's participation level. The probing system must identify a file to probe that most furthers this goal. Below are possible file selection criteria, which we use in our experiments in Section 5:

- **Select the file $r^i$ that has been probed the least** (min $N_p^i$, where $N_p^i$ is the number of times $r^i$ has been probed)**.** With this selection criterion, we ensure that all files are selected in a round-robin way. This prevents a file from being probed repeatedly.
- **Select the file $r^i$ that has been in the most/least query responses ("hits")** (min/max $N_r^i$, where $N_r^i$ is the number of times $r^i$ has been returned as a response)**.** If we assume that $N_r^i$ is an indication of $r^i$'s popularity, then probing it further increases its match rate for peers who have yet to find it. If we assume

that $N_r^i$ is low due to poor description, then probing remedies the situation and increases $r^i$'s match rate.

- **Select the file $r^i$ with the smallest descriptor** (min $|d(r^i)|$, where $|d(r^i)|$ is the descriptor size of $r^i$)**.** The smaller a file's descriptor, the more likely a query will not retrieve it due to over-specification (i.e., a relevant query contains terms not in the descriptor). Probing this file alleviates this problem.

These criteria can be combined in various ways, as we demonstrate in Section 5.3. Combining the criteria is useful in breaking ties and in avoiding wasteful situations, such as having a particular file probed repeatedly.

**4.2.3. Applying probe results to the local descriptor.** Once the probe results for file $o_i$ are returned to the client, they are grouped into a multiset. The client then selects terms from this multiset to add to $d(r^i)$. We consider four possibilities in term selection, each of which takes a different approach to increasing participation level:

- **random** – randomly select terms from the multiset. The goal of random term selection is to create new combinations of terms in descriptors that may allow a greater variety of queries to be matched. The selection process continues until $d(r^i)$ is full.
- **weighted random** – randomly select terms from the multiset based their relative frequencies. Weighted random is similar to random, but prioritizes more frequent terms.
- **most/least frequent** – select instances of terms in order of their frequencies (or inverse frequencies) in the multiset. Only terms that are not already in $d(r^i)$ are considered to maximize term variety. The selection process continues until $d(r^i)$ is full or there are no more terms to select. Most-frequent term selection's goal is to increase the likelihood of matching based on a consensus of what terms are strongly associated with a file. Least-frequent term selection's goal is to match queries that may contain "outlier" terms.

We do not replace terms already in the descriptor as this may interfere with the local user's naming conventions. Many other term selection possibilities exist, but our current goal is to demonstrate the general impact of term selection with these canonical alternatives.

## 5. Experimental Results

We simulate the performance of a P2P file-sharing system to test the large-scale performance of our methods. In accordance with the accepted model described in [14] and observations presented in [15], we include in our experimental model *interest categories* $C$, a partitioning of $O$ into sets $C_i \in C$, where $C_i \subseteq O$, and $\cup_i C_i = O$. Interest categories are used to model constraints on user interests.

Each category $C_i$ has popularity, $b_i$, with a Zipf-skewed distribution. At initialization, each peer $p_j$ is assigned some interests $I_j \subseteq C$, based on $b_i$.

Each file $o$ within each instance of an interest category varies in popularity, which is also skewed using a Zipf distribution. This re-skewing of popularities models individual user interests and governs the likelihood that a peer who has interest in the category that contains $o^m$ is initialized with a replica of $o^m$. Each replica, $r_j^m$, allocated at initialization has a randomly initialized descriptor subject to $o^m$'s natural term distribution. Peer $p_j$'s interest categories also constrain its searches; $p_j$ only searches files from $\cup_n L_n$, where $L_n \in I_j$.

We use Web data to simulate our language model (i.e., term distributions and interest categories). Web data are a convenient choice because they constitute a grouping of terms into documents (we use terms' relative frequencies within a document to simulate the natural term distribution for a file) and a grouping of documents into domains (we use domains to simulate interest categories).

Our data consist of an arbitrary set of 1,000 Web documents from the TREC 2GB Web track (WT2G). These documents come from 37 Web domains. Terms are stemmed, and markup and stop words are removed. The final data set contains approximately 800,000 terms, some 37,000 of which are unique. We also conducted experiments using other data sets with other data distributions, but, due to space constraints, we only present a representative subset of our results. The data used for all experiments are found on our Web site [16]. The other experimental results are available on request.

**Table 1. Query length distribution.**

| Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Prob. | .28 | .30 | .18 | .13 | .05 | .03 | .02 | .01 |

**Table 2. Parameters used in the simulation.**

| Parameter | Value(s) |
|---|---|
| Num. Peers | 1,000 |
| Num. Queries | 10,000 |
| Max. descriptor size (terms) | 20 |
| Num. terms in initial descriptors | 3-10 |
| Num. categories of interest per peer | 2-5 |
| Num. files per peer at initialization | 10-30 |
| Num. trials per experiment | 10 |

Terms for a query are picked randomly based on the desired file's natural term distribution, similar to a technique described in [17]. The query length distribution shown in Table 1 was derived from observations of query logs we collected over several days on the Gnutella network in the Spring and Summer of 2006 using our Gnutella network crawling tool [18]. Without loss of generality, queries are flooded to all peers. The simulation parameters listed in Table 2 are based on observations of

real-world P2P file-sharing systems and are comparable to the parameters used in the literature.

Although other behavior is possible, we assume that the user identifies and downloads the desired result group with a probability $1/r$, where $r \geq 1$ is its position in the ranked set of results. If the desired result is not in the result set, $r = \infty$ and the top-ranked result is selected. We use the group size ranking function described in Section 3. We have also tried other ranking functions (e.g., cosine similarity), but group size ranking results are representative.

Performance is measured using a metric known as mean reciprocal rank score (MRR) [19]. MRR is appropriate for known-item search, which P2P file-sharing queries are accepted as being. (See evidence of this in [20].) MRR assumes that there is a single, identifiable desired result, and uses the ranking of this result in the result set to compute an accuracy score. The score for an individual query is the reciprocal of the rank at which the desired result is returned. MRR is defined as

$$MRR = \frac{\sum_{i=1}^{N_q} \frac{1}{rank_i}}{N_q} \qquad (2)$$

where $N_q$ is the number of queries issued by the client and $rank_i$ is the rank of the desired file in query $i$'s result set.

For reference, we also present precision and recall, which have slightly different definitions than they do in traditional Information Retrieval (IR) since file replicas exist in P2P and not in IR. Let $S_A$ be the set of replicas of the desired file, and $S_R$ be the result set of the query. Precision and recall are defined as:

$$precision = \frac{|S_A \cap S_R|}{|S_R|} \qquad recall = \frac{|S_A \cap S_R|}{|S_A|} \qquad (3)$$

These more traditional IR metrics are useful in roughly diagnosing the performance of query processing and in generalizing the presented performance to other domains.

## 5.1. Applying probe results to the local descriptor

We consider how a client applies probe results to the local descriptor first to streamline our presentation: once a preferred probe result handling technique is established, we will use it in the rest of our experiments. Further, the probe result handling technique is intuitively independent of probe file selection and triggering, so we can handle it separately.

In Figure 2, we present MRR where probes are triggered randomly for randomly selected files at an aggregate rate of 5,000 probes for 10,000 file queries. Weighted random (wrand) outperforms the alternatives by two to seven percent. Random (rand) and least frequent (lfreq) term selection suffer from selecting too many weakly associated terms. Resultant descriptors are ineffective in matching queries. It could be even worse in the case that a few fake or incorrect descriptors exist along with correct ones, as their chance of selecting bad descriptive terms is definitely higher compared with that of wrand. Most frequent term selection (mfreq) creates homogenous descriptors, decreasing the variety of queries that can be matched. By avoiding these problems, weighted random term selection performs best.

The statistical significance of these results was confirmed via *t*-tests at the 95% significance level. Because weighted random outperforms the alternatives, we use it as the default probe result term selection technique in the rest of the discussion.

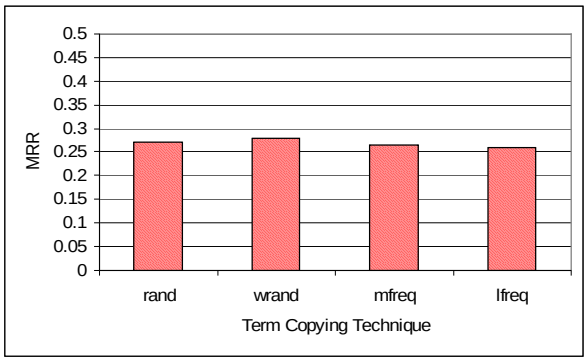


**Figure 2. MRR with various term selection techniques.**

## 5.2. Probe triggering techniques

We now consider the impact of various alternative triggering conditions. The three alternatives we consider are:

1. No probing (base case).
2. Random probing at a uniform rate.
3. Using the triggering condition from Section 4.2.1.

With random probing, we perform 5,000 *probe queries* over 10,000 *file queries* by assigning to all peers an appropriate random probability of probing right after each query. To trigger 5,000 probe queries using the triggering condition, we manually tune $T$.

Experimental results shown in Figure 3 clearly indicate that probing improves query accuracy. Random probing, denoted "random", increases MRR by 20% over the non-probing base case, denoted "noprobe". Probing using the triggering condition increases MRR by 30% (denoted "T5K") over the base case. That the triggering condition results in an improvement in accuracy indicates that probe activity should be directed at peers that are underutilized and not wasted on appropriately participating peers.

Much of this accuracy improvement is due to the improved ability for longer queries to retrieve desired results. Longer descriptors alleviate the problem that longer

queries have of result over-specification as shown in Figure 4: MRR increases by about 20% for two-term and by about 1000% for eight-term queries. Probing slightly reduces the performance of single-term queries because they are already very unselective and increasing the sizes of their result sets through longer descriptors leads to the inclusion of many superfluous results. These results serve to obscure the desired one. Given the small performance loss in this case, particularly as compared with the gains, overall performance is still improved.

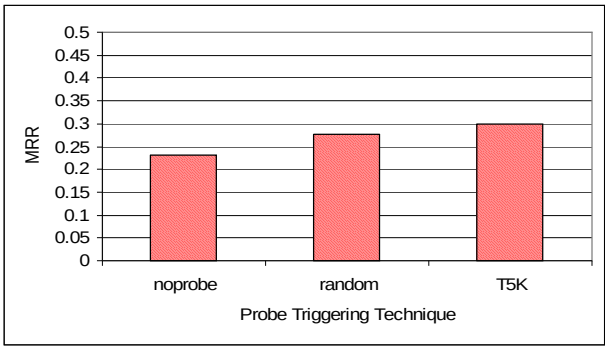


**Figure 3. MRR with various probe triggering techniques.**

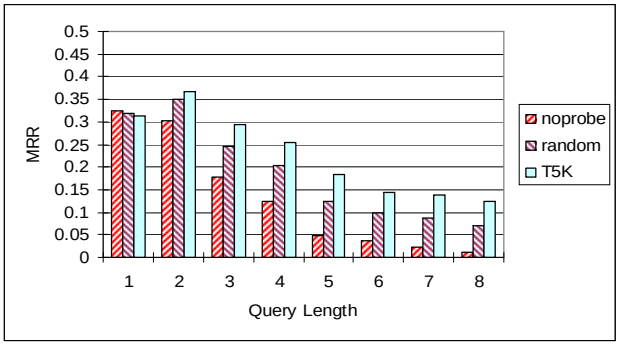


**Figure 4. MRR for various query lengths.**

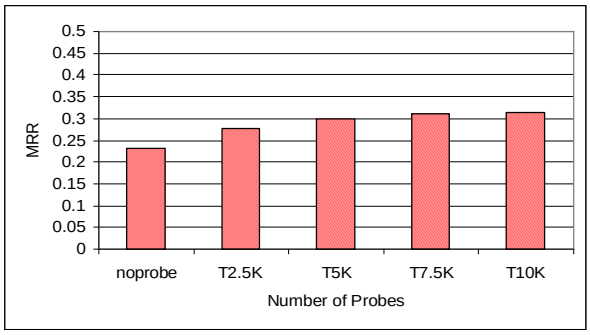


**Figure 5. Effect of various probing rates on MRR.**

To test the impact of probe rate, we varied $T$ so that 2,500, 5,000, 7,500, and 10,000 probe queries were issued per 10,000 file queries. As shown in Figure 5, MRR increases with more probes, but at a decreasing rate. This indicates that probing should be controlled, as its benefit diminishes. We analyze probing costs and benefits in more detail in cost Section 5.4.

Another indication of the effectiveness of probing is how it allows peers to tune their participation levels. In Table 3, we see that the average actual participation level (defined in Section 4.2.1) increases with probe rate by 30% to 94%. That the increase in standard deviation in actual participation level (29%) is relatively low compared with the increased averages (94%) is indicative of system responsiveness to user control.

**Table 3. Mean and standard deviation of actual participation level ($N_r/(N_fN_q)$) with various probing rates. Values are $\times 10^{-2}$.**

| | Noprobe | T2.5K | T5K | T7.5K | T10K |
|---|---|---|---|---|---|
| mean | 0.262 | 0.343 | 0.412 | 0.469 | 0.508 |
| std dev | 0.097 | 0.093 | 0.104 | 0.115 | 0.125 |

### 5.3. Probe file selection techniques

We now consider the impact of five different file selection techniques based on the combinations of the criteria discussed in Section 4.2.2. We use number of query hits ($N_r^i$) and number of probes (i.e., round-robin) ($N_p^i$) as our primary criteria:

- **Rand** – Randomly select a file to probe (base case).
- **LPF** – least popular first: Select file $r^i$ with the minimum number of query hits (min $N_r^i$). On a tie, use the minimum descriptor size (min $|d(r^i)|$).
- **MPF** – most popular first: Same as LPF, but with *max* $N_r^i$ instead.
- **RR-LPF** – round-robin-LPF: Select file $r^i$ with the minimum number of probe queries (min $N_p^i$). On the first tie use the minimum number of query hits (min $N_r^i$). On the second tie, use the minimum descriptor size (min $|d(r^i)|$).
- **RR-MPF** – round-robin-MPF: Same as RR-LPF, but with *max* $N_r^i$ instead.

In these experiments, probes are triggered randomly at a rate of 5,000 probe queries per 10,000 file queries.

In Table 4, we compare each file selection technique with the base case of random selection. The "Cost" column indicates the total number of responses to both probe and file queries. The "Pct. Contained" column indicates the percentage of queries whose result sets contain the desired result. Only RR-MPF outperforms the random base case in all the metrics.

The two round-robin techniques outperform those based on popularity because the uniform distribution of probes reduces the likelihood that probes are wasted on unpopular files (LPF) or that popular files are over-probed (MPF). LPF probing increases the MRR of queries for

unpopular files (not shown), but without a noticeable overall difference because few queries are for these files. Over-probing a file (e.g., a popular file) with MPF is wasteful and counterproductive because additional probes for a file, which could be better allocated to another file, yield only marginal information and increases the likelihood that the file matches irrelevant queries. Consequently, both precision and recall when using MPF are worse than with rand. Cost increases with MPF because probes return many results and increase the recall of common file queries.

**Table 4. Comparison of various file selection techniques against Rand on different metrics. Comparators indicate how well each file selection criteria compares with random file selection.**

| | MRR | Cost | Recall | Prec. | Pct. Contained |
|---|---|---|---|---|---|
| Rand | = | = | = | = | = |
| LPF | < | < | < | < | > |
| MPF | < | > | < | < | < |
| RR-LPF | < | < | > | < | > |
| RR-MPF | > | < | > | > | > |

The round-robin techniques smooth out the problems associated with using popularity a primary file selection criterion. RR-MPF prioritizes the more popular files by using popularity as a tie-breaker and therefore has a better impact on all the metrics for the reasons stated above: MRR increases by 11% and cost decreases by 10% over rand. The increase in accuracy is supported by the fact that RR-MPF increases both precision and recall. This is an indication that the right files are being probed. The higher cost of rand is due to the fact that it is more likely to repeatedly probe popular files because there are many of them. RR-LPF works worse than RR-MPF because the former focuses too much on unpopular files and introducing unpopular results into queries for popular files decreases precision. Random selection works reasonably because it ensures that there is no correlation among the choices of the peers (e.g., not all peer probe the same file.)

A two-tail *t*-test at a 95% significance level verified the statistical significance of these results. The difference between RR-MPF and the base case is statistically significant as the calculated $t$ value (2.12) exceeds the $t$ value threshold (1.96).

### 5.4. Cost analysis

We define cost as the number of query responses because it is a rough estimate of both client processing and network load. Because there are two types of queries – probe queries for descriptors and file queries for files – there are two types of query responses, and two cost components.

Our strategy to reduce cost is to use server-side *Bernoulli sampling* on the result set for each query. That is, for each matching file query result, the server decides to return it to the client with a fixed probability $P_r$, $0 \le P_r \le 1$. This type of sampling is expected to preserve the overall distribution of terms and results in the result set with a predictable reduction in cost by a factor $P_r$. We define the sampling rate of probe queries similarly at $P_p$. The question is what effect such sampling has on query accuracy.

In Figure 6 and Figure 7, we see the impact that sampling rates have on cost and accuracy. Peers perform probe queries using the T5K threshold described above. Cost is lower than the no probing case in three combinations of $P_p$ and $P_r$, respectively: (.50, .25), (.25, .50), (.25, .25). However, as shown in Figure 7, for all combinations of $P_p$ and $P_r$, MRR is higher than without probing. This indicates that, although accuracy always improves, it is also possible to simultaneously improve both accuracy and cost. For example, when $P_p = .25$ and $P_r = .25$, cost is *half* that of the no probing case with similar MRR. When $P_p = .25$ and $P_r = .5$, cost is about 15% lower than that of no probing, but MRR is about 18% higher.

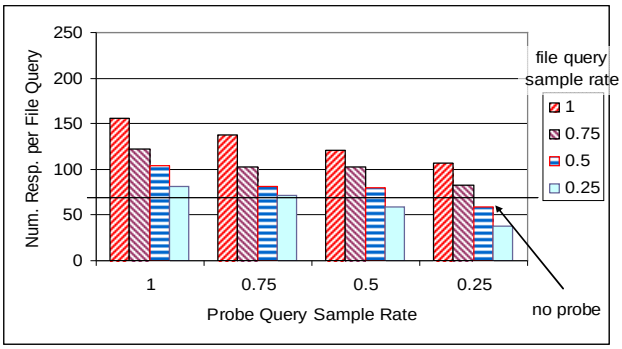


**Figure 6. Total per-file-query cost for different file and probe query sampling rates.**

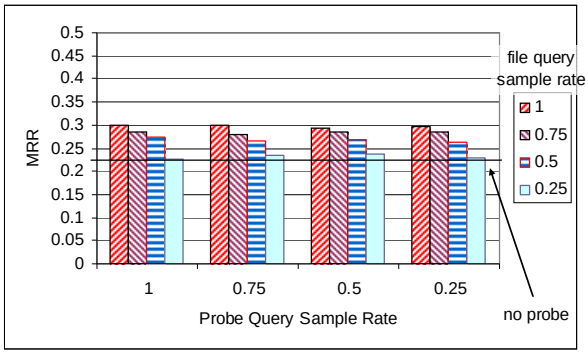


**Figure 7. MRR for different file and probe query sampling rates.**

The reason for this positive performance/cost behavior is due to the fact that probing can increase both query recall and precision as shown in Figure 8. The higher quality results are more able to resist the negative effect

that sampling has on accuracy because the desired result is less likely to be sampled out of the result set.

Specifically, for a query to contribute to MRR, its result set must contain at least one instance of the desired result. The probability this result is *not* in a result set $U$ is $(1 - \text{precision}(U))^{|U|}$. Probing increases both $|U|$ and precision($U$), while sampling decreases $|U|$ but leaves precision($U$) unchanged. Probing therefore increases both $U$'s MRR and its ability to withstand sampling.

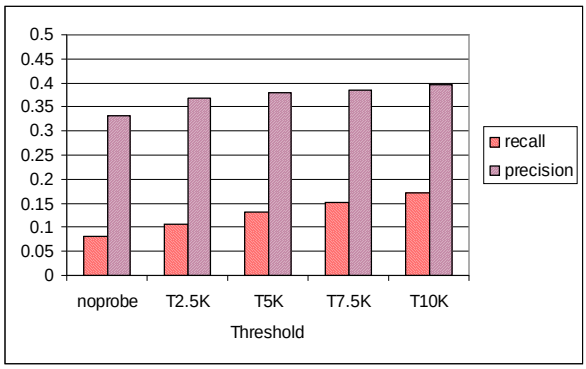


**Figure 8. Recall and precision with various probing rates.**

**5.4.1. Sampling in practice.** From the experimental results shown in Figure 6 and Figure 7, it is clear that reducing the probe query sampling rate does relatively little to hurt MRR (especially compared with file query sampling), but significantly reduces cost. A heuristic for probing rates, therefore, is to fix $P_p$ at a low value and to progressively decrease $P_r$ until the user indicates that a query has failed (i.e., by not following a query with a download). For failed queries, $P_r$ is restored to a higher value.

## 6. Conclusion

Given the conjunctive matching criterion of today's P2P file-sharing systems, poor data description limits query accuracy. Probe queries help solve this problem by automatically tuning local descriptors using those of other peers. Our experimental findings demonstrate that it is possible to improve query accuracy up to 30% at a tunable cost by probing for better file descriptions and sampling result sets. Furthermore, this is accomplished in a fully distributed fashion and is therefore suitable for general P2P environments. Ours is the first work we know of that automatically tunes the descriptions of shared data and could be extended beyond the P2P environment to describing all non-self-describing binary data.

We are now considering ways of better controlling exactly where probes are directed (i.e., more or less popular files). We are also developing models to help in tuning probing threshold and sampling values manner. We are also considering different ways of sampling the network to yield cardinality estimations on shared data for the sake of tuning sampling rates [21]. Finally, we are building test data from traces from the Gnutella network using a tool we recently developed [18] – no such data currently exists [22], esp. the lack of relevance judgment between queries and shared files in Gnutella network.

## 7. References


[1] Limewire. www.limewire.org.

[2] eMule Project Home Page, www.emule-project.net.

[3] Slyck.com P2P File-sharing Statistics. slyck.com/stats.php.

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

[5] S. Ciraci, I. Korpeoglu, and O. Ulusoy. Characterizing Gnutella Network Properties for Peer-to-Peer Network Simulation. In *Intl. Symp. On Comp. Info. Sys.*, 2005.

[6] B. Loo, J Hellerstein, R Huebsch, S Shenker and I Stoica, Enhancing P2P File-sharing with an Internet-Scale Query Processor, In *Proc. VLDB Conf.*, 2004.

[7] F. M. Cuenca-Acuna and T. D. Nguyen, Text-based content search and retrieval in ad hoc P2P communities, In *Proc. Intl. Wkshp Peer-to-Peer Comp*, 2005.

[8] J. Lu and J. Callan. User modeling for full-text federated search in peer-to-peer networks. In *Proc. SIGIR*, 2006.

[9] Y. Shao, R. Wang. BuddyNet: History-based P2P Search. In *Proc. ECIR*, 2005.

[10] K. Sripanidkulchai, B. Maggs, H. Zhang. Efficient content location using interest-based locality in peer to peer systems. In *Proc. INFOCOM*, 2003.

[11] M. M. Masud, I. Kiringa, A. Kementsietsidis, Don't Mind Your Vocabulary: Data Sharing Across Heterogeneous Peers, In *Proc. of* CoopIS, 2005.

[12] W. G. Yee, L. T. Nguyen, O. Frieder. Masked Queries for Search Accuracy in Peer-to-Peer File-Sharing Systems, In *Proc. IEEE IPDPS*, 2007.

[13] M. Nilsson. Id3v2 web site. www.id3.org.

[14] M. T. Schlosser, T. E. Condie, and S. D. Kamvar, Simulating a file-sharing p2p network, In *Proc. Wkshp. Semantics in Peer-to-Peer and Grid Comp.*, May 2003.

[15] S. Saroiu, P. K. Gummadi, and S. D. Gribble, A measurement study of peer-to-peer file sharing systems, In *Proc. Multimedia Computing and Networking*, Jan. 2002.

[16] PIRS Research Group Data, ir.iit.edu/~waigen/proj/pirs/data/.

[17] L. Azzopardi, M. de Rijke. Automatic construction of known-item finding test beds. In *Proc. SIGIR*, 2006.

[18] L. T. Nguyen, W. G. Yee, D. Jia, and O. Frieder, A Tool for Information Retrieval Research in Peer-to-Peer File Sharing Systems, In Proc. *IEEE ICDE*, 2007.

[19] E. Voorhees and D. Tice, The TREC-8 Question Answering Track Evaluation, In *Proc. of the Eighth Text REtrieval Conference*, 1999.

[20] L. T. Nguyen, D. Jia, W. G. Yee, and O. Frieder, An Analysis of Peer-to-Peer File-Sharing System Queries. To appear *Proc. SIGIR*, 2007.

[21] N. Ntarmos, P. Triantafillou, G. Weikum. Counting at Large: Efficient Cardinality Estimation in Internet Scale Data Networks. In *Proc. IEEE ICDE*, 2006.

[22] H. Nottelmann, K. Aberer, J. Callan, and W. Nejdl, CIKM 2005 P2PIR Workshop Report, In *SIGIR Forum*, 2006.