Measurement and Analysis of Ultrapeer-based P2P Search Networks[*]

Boon Thau Loo[*] Joseph Hellerstein[*†] Ryan Huebsch[*] Scott Shenker[*‡] Ion Stoica[*]

[*]UC Berkeley [†]Intel Berkeley Research [‡]International Computer Science Institute

{boonloo, jmh, huebsch, istoica}@cs.berkeley.edu, shenker@icsi.berkeley.edu

Abstract

Unstructured Networks have been used extensively in P2P search systems today primarily for file sharing. These networks exploit heterogeneity in the network and offload most of the query processing load to more powerful nodes. As an alternative to unstructured networks, there have been recent proposals for using inverted indexes on structured networks for searching. These structured networks, otherwise known as distributed hash tables (DHTs), guarantee recall and are well suited for locating rare items. However, they may incur significant bandwidth for keyword-based searches. This paper performs a measurement study of Gnutella, a popular unstructured network used for file sharing. We focus primarily on studying Gnutella's search performance and recall, especially in light of recent ultrapeer enhancements. Our study reveals significant query overheads in Gnutella ultrapeers, and the presence of queries that may benefit from the use of DHTs. Based on our study, we propose the use of a hybrid search infrastructure to improve the search coverage for rare items and present some preliminary performance results.

1 Introduction

Gnutella [1] and Kazaa [6] have been widely used in file-sharing applications. These networks are often called *unstructured*, because nodes are organized in an ad-hoc fashion and queries are flooded in the network. Unstructured networks are effective for locating highly replicated items that can be retrieved with high probability in a small number of hops. They are less effective for rare items because recall is not guaranteed.

On the other hand, there have been proposals for using inverted indexes on distributed hash tables (DHTs) [13]. These networks guarantee perfect recall (in the absence of network failures), and are able to locate matches within a bounded number of hops. However, DHTs only provide exact-match queries, and may incur significant bandwidth for executing more complicated keyword-based searches.

There is no consensus on the best P2P design for searching. The contributions of this paper is as follows:

- Conduct a measurement study of Gnutella, an open-source unstructured P2P network for file-sharing. The measurement study examines the new ultrapeer-based topology, and analyzes the search performance and recall.
- Discuss the alternative of implementing searching using structured networks.
- Propose a hybrid search infrastructure which makes use of unstructured search techniques for popular items, and structured search techniques for rare items.
- Deploy the hybrid search infrastructure with the existing Gnutella network and present preliminary experimental results.

2 Measurements Methodology

In the measurement study on Gnutella, we made use of a crawler as well as a Gnutella client instrumentation software.

2.1 Gnutella Crawler

The crawler was used primarily to map out Gnutella's topology. Given Gnutella's recent upgrades to ultrapeers, we are interested to figure out the ratio of ultrapeer to leaf nodes, and the number of neighbors maintained by both ultrapeers and leaf nodes. The crawler works as follows: first, a connection is made to a bootstrapping node that is already part of the Gnutella network. Once connected, a *ping crawl* request is sent to this node. This ping crawl request is a standard Gnutella ping message with TTL of 2. This ping request travels one extra hop, allowing us to determine the neighbors of

[*]This research was funded in part by NSF awards (grant No. IIS-0205647 and grant No. IIS-0209109)

the crawled nodes. Using the neighbors obtained, we then proceed recursively with the crawl, building the Gnutella topology as the crawl proceeds.

We started 30 separate crawlers, bootstrapped from 30 ultrapeers running on Planetlab. The crawl was performed on 11 Oct 2003 and lasted 45 minutes.

2.2 Client Instrumentation Software

The client instrumentation software was modified from Limewire's [7] implementation of the Gnutella client software which we downloaded in July 2003. Our modified client participates in the Gnutella network as an ultrapeer or leaf node, and logs all incoming and outgoing Gnutella messages through the node. It also has the ability to inject queries into the network and gather the incoming results. We used this client in the following three setups:

- **A: Single node measurement.** We measured an ultrapeer and a leaf node over a period of 72 hours from 7-10 Oct on 2 Planetlab [9] nodes each. This allows us to measure the lifetimes and number of files shared of neighboring nodes. The measurements were obtained from 7-10 Oct.

- **B: Multiple ultrapeers measurement.** We ran the client on Gnutella ultrapeers on 75 Planetlab (selected from the US, Canada and Europe) for three hours duration. The measurements were repeated and averaged over 3 experiments conducted on 12-13 Oct 2003. This setup allows us to examine the query processing overheads of different ultrapeers situated at different locations. It also provided a global view of the Gnutella network, and allows us to estimate a lower bound on the total number of queries being executed in the network.

- **C: Trace-driven ultrapeers.** We ran the client on 30 Gnutella nodes on Planetlab as above, and injected a total of 350 queries from each node derived from an earlier trace at 10 seconds interval. The measurements were repeated and averaged over 6 experiments conducted on 5th, 6th, 12th and 13th Oct 2003. This setup enables us to estimate the average recall of a typical Gnutella query.

2.3 Improvements to Gnutella

In this section, we describe briefly two recent improvements to the Gnutella protocol, namely the use of ultrapeers and dynamic flooding techniques.

2.3.1 Gnutella Topology with Ultrapeers

The Gnutella search protocol works as follows: each client searching for a file forwards the query request to each of its neighbors, which then forwards the request to other neighbors for a bounded number of hops (TTL). This flooding protocol is not scalable, as discussed in [11]. It also poses a strain on weak clients (such as those on modem links) who have to handle a large portion of the search traffic.

In an effort to address the scalability problems, both Gnutella and Kazaa utilizes *ultrapeers* [4]. This concept capitalizes on the excess resources of powerful nodes in the network, to carry the search traffic of less resource-rich *leaf nodes*.

Nodes regularly determine whether they are eligible to become an ultrapeer based on their uptime and resources. Once decided to be an ultrapeer, each ultrapeer will advertise itself as an ultrapeer to other connecting nodes. Ultrapeers perform the bulk of the query execution on behalf of a group of leaf nodes that they support. When a regular leaf node issues a query, it is forwarded to its one or more assigned ultrapeer. The ultrapeer will then broadcast the query among its ultrapeer neighbors up to a limited number of hops.

The ultrapeer builds an index of its leaves' contents as follows: Leaf nodes connect to one or more ultrapeers, and forward information on all shared files to the ultrapeer. The ultrapeer then responds to queries on behalf of leaf nodes, shielding them from the query traffic [1].

The crawlers discovered a total of 114597 nodes, out of which 22453 responded as ultrapeers[2]. Since these crawlers were ran separately, there were overlaps in the nodes they discovered. 70% of the nodes were discovered by more than one crawler.

65% of the nodes we connected to were either Bearshare [12] or Limewire [7] clients. 60% of nodes did not share any files. Ignoring nodes that reported excessive number of files (> 500000 files), the average number of files shared per node was 193. The total number of files shared was approximately 22 million.

Figure 1 shows the number of number of neighbors each contacted node maintained. We did not include nodes where we were not able to gather any neighbor information. Since we performed the crawl while

[1]There are other variants of this basic approach, such as the use of Query Routing Tables (QRP) [3]. The reader is referred to the article [4] for more details.

[2]The ratio of ultrapeers to nodes may be overstated. If we consider only nodes with degree ≥ 6 as ultrapeers, the number of ultrapeers reduces to 7000

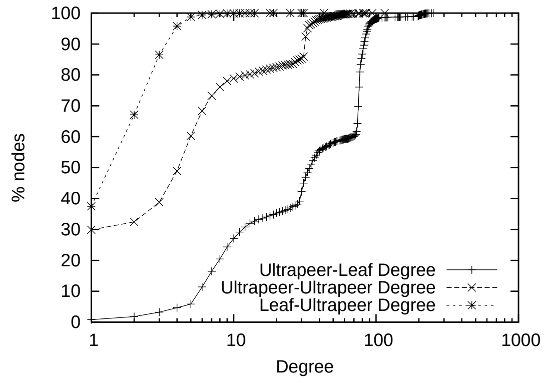

Figure 1: *Degree of Nodes*

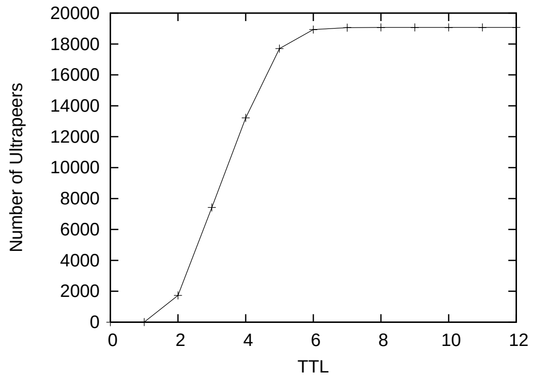

Figure 2: *Diameter of Network*

the network was in flux, there were many nodes that were still in the process of building up their neighbor list, and hence the number of nodes they connect to can range from 0 to as many as the maximum number of nodes they can connect to. The topology information gathered from the crawl was not a perfect snapshot of the Gnutella network. However, it can offer useful insights into how the network is organized.

37% of the leaf nodes were connected to one ultrapeer, while almost all of them (96%) were connected to up at most 4 ultrapeers. In computing the out-degrees of ultrapeers, we took into consideration ultrapeers that reported at least 6 neighbor nodes (either leaves or ultrapeers). Most ultrapeers either had 6 ultrapeer neighbors and 75 leaf nodes, or 32 ultrapeer neighbors and 30 leaf nodes. The discrepancy in the out-degree of ultrapeers is due to recent improvements to the protocol that reduced the TTL of queries, while allowing dynamic querying (refer to section 2.3.2. These improvements requires an increase on the number of ultrapeer neighbors to be effective. The number of leaf nodes supported are reduced, presumably to offset the increased in ultrapeer neighbors. The increased number of ultrapeer neighbors from 6 to 32 also has the effect of reducing the likelihood of network partitions amongst the ultrapeers.

Figure 2 shows an estimation of the diameter of the Gnutella network of ultrapeers. We were able to gather neighbor information on approximately 19000 ultrapeers. For each of these ultrapeers, we compute the number of ultrapeers that the ultrapeer can reach within one hop, two hops, and so on until it connects to all possible ultrapeers that it can reach. The number of reachable ultrapeers is then averaged across all ultrapeers. The diameter of the network of ultrapeers is at most 12, and almost all ultrapeers can be reached in 6 hops. We expect the diameter of the network to be further reduced if more clients adopt the Gnutella client that supports up to 32 ultrapeer neighbors.

2.3.2 Dynamic Querying

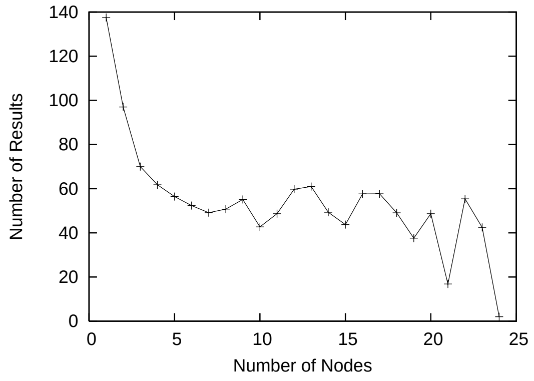

Figure 3: *Average Query Results Size against Number of Ultrapeers Visited.*

In unstructured networks, queries are executed by *moving the queries towards the data.* In order to locate rare items, the query would have to be flooded to a larger number of nodes. In Limewire's implementation of *dynamic querying* [2], a Gnutella ultrapeer floods the query on a few initial connections starting from a small default TTL (e.g. 2). If the number of results obtained from these initial connections is small, it may choose to increase the TTL for sending the query on subsequent connections. Dynamic querying is only effective if an ultrapeer maintains a large number of ultrapeer neighbors. To verify that dynamic querying has been used on our ultrapeers, we estimated the distance each query has traveled by observing 350 distinct Gnutella queries issued from 30 ultrapeers in setup

C. For each query, we counted the number of times the queries have been seen on the ultrapeers that we were measuring. The more ultrapeers the query had been seen on, the higher the chances that query had been flooded for a longer distance. We then computed the average number of query results returned, grouped by the number of ultrapeers seen. Figure 3 shows the correlation between the average number of query results returned and the number of ultrapeers the queries had visited. As the number of ultrapeers visited increased, the average number of query results obtained reduced. This indicates that queries that returned fewer results did traveled further and visited more nodes in the Gnutella network.

2.4 Gnutella's Search Characteristics

In this section, we examine the overheads of processing queries in an ultrapeer. We also used our measurements to compute the average query recall, bandwidth usage and latencies of Gnutella queries.

2.4.1 Ultrapeer Query Processing

Using query logs obtained from 70 ultrapeers in setup B, an ultrapeer received 18 queries per second on average. We observed a significant amount of wasted work due to the flooding-based protocol used by Gnutella:

- 33% of queries on each ultrapeer received were duplicate queries forwarded by more than one neighbor.
- The ultrapeer processed 63 queries per hour on behalf of leaf nodes. This constituted only 0.1% of the queries that the ultrapeer had received. This shows that a bulk of query processing load that an ultrapeer handles are from neighboring ultrapeers and not leaf nodes.
- Leaf nodes of the ultrapeer contributed to the results of 6% of the distinct queries received by the ultrapeer.
- 5% of the distinct queries forwarded by an ultrapeer to ultrapeer neighbor received query results from them. This shows that a large portion of forwarded query traffic did not perform useful work of getting query results.

2.4.2 Search Coverage

To figure out the average recall of a query, we would have to take a complete snapshot of all the files in the network at the time when the query was issued. We can approximate this by issuing the same query simultaneously from different Gnutella ultrapeers. In setup C, we issued a total of 350 distinct queries from 30 ultrapeers. Each query was simultaneously issued from all 30 ultrapeers. Hence, a total of 10500 queries were issued. In figure 4, *Average Results* shows the CDF of the number of results returned by all 10500 queries averaged over 6 experiments. *Total Results* shows for each distinct query, the total number of results returned from all 30 ultrapeers (computed based on the union of results obtained for each distinct query). We considered each file unique by filename, IP address and reported file size. Figure 5 shows the same graph for only queries that returned 20 results or less.

Averaging over six experimental runs, 45% of queries issued obtained 10 or less results, while 18% obtained no results. Based on the union of results from all 30 ultrapeers, the percentages reduced to 27% and 6% of queries respectively. Hence, there were 13% of queries that *should have* received results from the Gnutella network at the time the query was

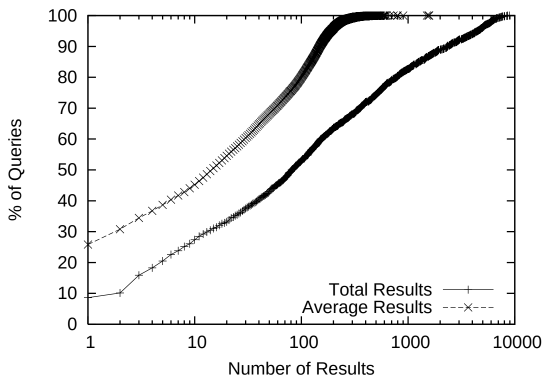

Figure 4: *CDF of Query Results from 10500 queries on 30 Ultrapeers*

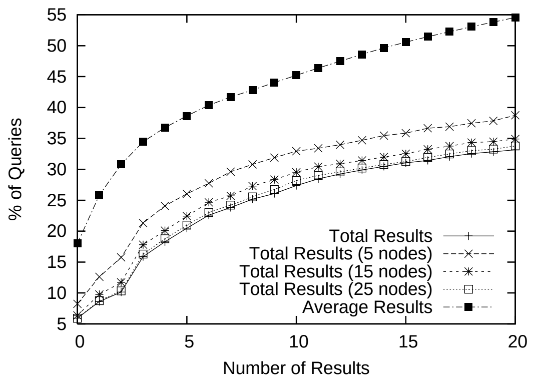

Figure 5: *Same graph as figure 4 (≤ 20 results).*

issued. In figure 5, we show the gains in coverage using the results from 5, 15 and 25 ultrapeers. As we increase the number of ultrapeer results used, there would be diminishing returns in terms of additional coverage.

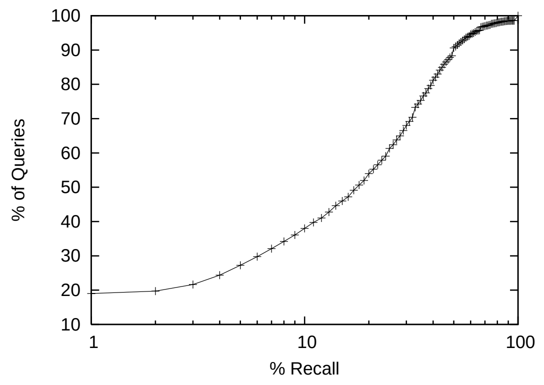

Figure 6: *Query Recall CDF.*

Figure 6 shows the query recall CDF of all 10500 queries averaged over 6 experiments. We define the recall of a query as the number of distinct results returned for that query, as a percentage of the total number of distinct results obtained from all 30 ultrapeers. For queries that obtained 1 or more result, the average per query recall was 23%. In a separate analysis of the results, we do not take into account duplicates in the results, by counting only the number of unique file names in the results obtained. I.e. we assume that users are only interested in any *one* copy of the file (identified by filename), regardless of where it is located and how large that item may be. The average recall of queries increased to 28% under this assumption.

We repeated the same experiments using 30 leaf nodes to issue the queries. 13% of the queries did not receive any query results. That percentage decreased to 6% when we took the union of results from all 30 leaf nodes. The average query recall was 25%. The leaf nodes that we used have better search coverage as each of them had 4 ultrapeer parents each.

2.4.3 Per Query Bandwidth Usage

Empirically, we can estimate the average cost of each query. First, we estimate the total query rate that is being supported by the entire Gnutella network. We made use of setup B which involved running 70 ultrapeers for 3 hours duration. Averaging over 3 experimental runs, there were 96 distinct queries being seen on all ultrapeers per second. This means that as a lower bound, 96 queries per second ($TOTAL\_QUERY$) were being issued in the entire network. We round up this value to 100 queries per second. Each ultrapeer forwarded an average of 18 queries per second ($Q\_FORWARD$). The crawl in section 2.3.1 revealed that there were roughly on the order of 10000 ultrapeers in the network. The average bandwidth cost per query is hence approximated by $(SIZE\_MSG * NUM\_ULTRAPEERS \times Q\_FORWARD)/TOTAL\_QUERY = (80B * 10000 * 18)/(100 queries/second) \approx$ **140KB**.

2.4.4 Query Latency

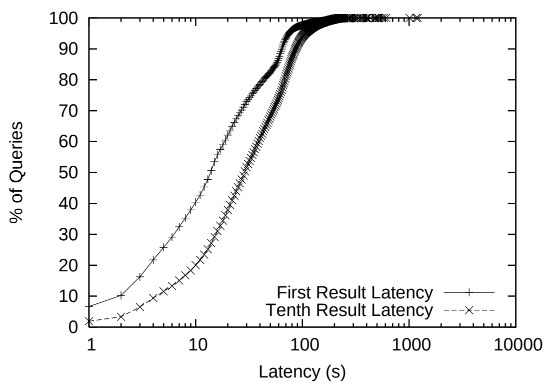

Figure 7: *Latency of Gnutella Results*

We measured the latency of the 10800 Gnutella queries issued from ultrapeers in setup C. Figure 7 shows the latency of these queries. The average time to receive the first result is 25 seconds, and the median is 14 seconds. The average time to get the tenth tuple is 39 seconds, and the median is 28 seconds.

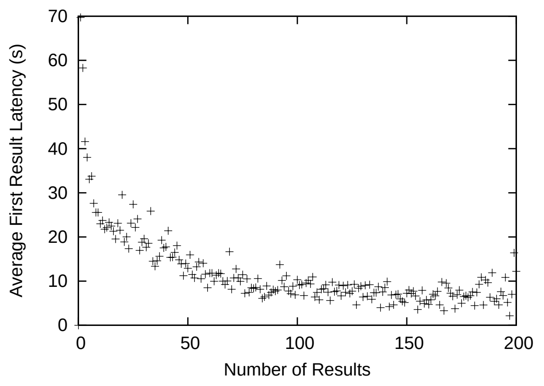

Figure 8: *Correlation of Number of Query Results and Latency*

Figure 8 shows the correlation of the number of results obtained from a query against the latency of the first result. This is done by grouping queries by the number of results obtained and computing the average latency to obtain the first result. Queries that returned few results tend to have longer latencies. For queries that received a single result, the average latency was 71 seconds. The average latency levels off at around 6-8 seconds as the number of results increases.

2.4.5 Summary

We summarize our findings as follows:

- Gnutella has a two-tiered topology, consisting of more powerful ultrapeer nodes that are supporting weaker leaf nodes. Most ultrapeers either had 6 ultrapeers and 32 leaf nodes, or 32 ultrapeers and 30 leaf nodes.

- Gnutella is effective for highly replicated items. Dynamic querying reduces the cost of these queries. Our measurements showed that queries that return many results have short response times.

- 45% of queries returned 10 or less results, and 13% of queries did not receive any results even though there were items that would satisfy these queries in the network. These queries would benefit from the use of DHTs as recall would be guaranteed.

- While increasing the TTL would improve the recall of queries, this would improve bandwidth costs per query and would clearly not scale.

3 DHT-based Keyword Search

Given that flooding-based approaches in unstructured networks incur high overheads, we examine the use of DHTs for searching. DHT-based searching is done by publishing traditional inverted files indexed by DHTs (also called *inverted indexes*). Boolean search proceeds by hashing via the DHT to contact all sites that host a keyword in the query, and executing a distributed join of the postings lists of matching files. DHTs guarantee full recall if there are no network failures, but may use significant bandwidth for distributed joins.

In the subsequent sections, we describe our implementation of DHT-based keyword search for file sharing, as well as address the issues of network churn and substring queries.

3.1 Implementation using PIER

We implemented file-sharing search using PIER (Peer-to-Peer Information Exchange and Retrieval) [21], a P2P relational query engine over DHTs. Using PIER, information of each file is published as a `FileInformation(docID, filename, location, filesize, other fields)` tuple indexed by *docID*. The docID is a unique identifier of the file. To enable searching by keyword, we construct an *inverted index* as follows: for each keyword in the filename, a *posting list* of all the unique identifiers of all files containing the keyword. This is stored as `Posting(keyword, docID)` tuples indexed by *keyword*.

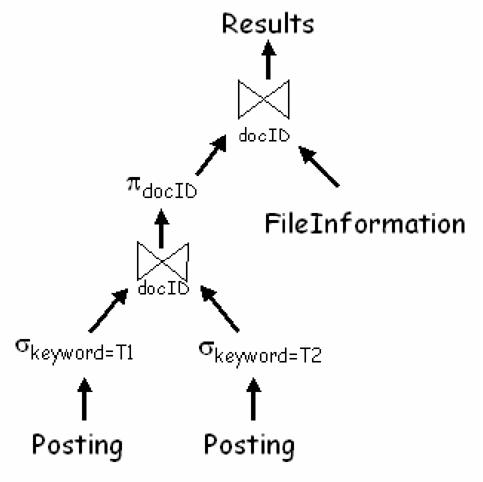

Figure 9: *Query Plan of Symmetric Hash Join for a two-term query T1 and T2.*

Queries involving more than one term are executed by intersecting the respective posting lists. The query plan for a two-term query is shown in figure 9. The query proceeds by sending the query to the nodes responsible for the keywords in the query. The node that hosts for the first keyword in the query plan will send the matching tuples to the node that hosts for the next keyword. The node receiving the matching tuples will perform a *symmetric hash join (SHJ)* between the incoming tuples and its local matching tuples and the results are sent to the next node. The matching docIDs are sent back to the query node, which fetches all or a subset of the complete file information based on the docIDs.

A slight variant of the above algorithm shown in figure 10 redundantly stores the filename with each posting tuple. This incurs publishing overheads but makes queries cheap, by avoiding a distributed join.

3.2 Network Churn

A major disadvantage of using DHTs for indexing file-sharing workloads is the fact that nodes in these networks join and leave the network at high fre-

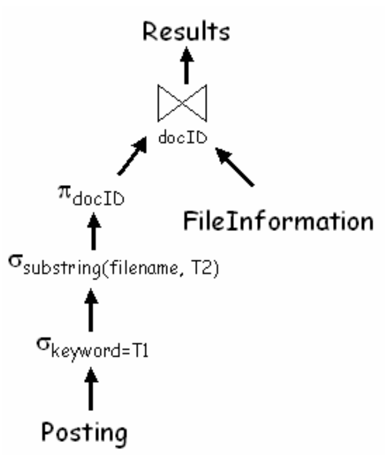

Figure 10: *Query Plan for Join Index for a two-term query T1 and T2.*

quency, making the DHT maintenance overheads high. To determine how severe network churn would be, we measured the connection lifetimes of leaf nodes and ultrapeers.

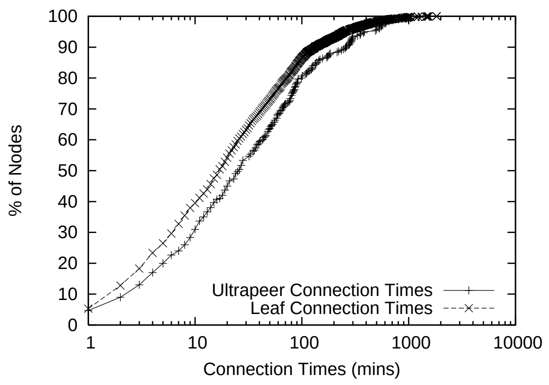

Figure 11: *Connection Lifetime of Nodes*

Using setup A, we measured the connection times of neighbors while running an ultrapeer and leaf node over 72 hours. The connection time of a neighbor node is defined as the time the neighbor node stays connected to our instrumented node. It is a lower bound on the session time (the time a node stays on the Gnutella network), as the neighbor nodes may have simply dropped their connections to our instrumented node. When our instrumented node was terminated after 72 hours, there were still some outstanding connections. We conservatively estimated the session lifetimes of these outstanding connections to be the duration when the connections was established until our experiments ended.

Figure 11 shows the connection lifetimes of ultrapeer and leaf neighbors. The lifetimes of leaves were observed using our instrumented ultrapeer, and the lifetimes of ultrapeers were observed using our instrumented leaf node. The average session lifetimes of a leaf node and ultrapeer measured were 58 minutes and 93 minutes respectively. The median session lifetimes were 13 minutes and 17 minutes respectively.

As ultrapeers have longer average session lifetimes than leaf nodes (1.5 times more), it is conceivable that they can be used as DHT nodes. The short median lifetimes of both leaf nodes and ultrapeer nodes would result the published inverted indexes published becoming stale quickly. This problem can be adverted by not publishing file information from short-lived nodes. For example, if we do not consider leaf nodes whose lifetimes are less than 5 minutes, the average and median lifetimes of remaining leaf nodes becomes 90 minutes and 35 minutes respectively. Ignoring leaf nodes whose lifetimes are less than 10 minutes increases the average and median lifetimes to 106 minutes and 47 minutes. This means that for a DHT-based search engine to reduce the probability of indexing stale information, file information on short-lived leaf nodes should not be indexed.

3.3 Substring Queries

Substring queries using DHTs are more expensive as they would require expensive distributed joins based on n-grams of keywords and filenames. Recent Gnutella proposals to make use of Query Routing Protocol (QRP) [10, 3] would require the computation of bloom-filter summaries (QRP tables) to be cached at neighboring nodes. This would also make it harder to support substring matches without increasing the sizes of the QRP tables. In general, using either QRP or indexing via DHTs would compromise the ability to perform cheap substring queries.

4 Hybrid Search Infrastructure

DHT-based keyword searching guarantees perfect recall in the absence of network failures. However, it may incur expensive distributed joins, which can be avoided if join indexes are built (at a higher publishing costs).

Existing unstructured networks incur high flooding overheads but are sufficient for locating highly replicated files. Our experimental studies showed that the existing Gnutella network failed to locate any query results in 15% of the queries issued. Flooding the network for these queries would be too expensive. Replacing all ultrapeer nodes with a DHT-based network would increase the recall of queries, but may be too expensive if many queries involve the shipping of large intermediate posting lists.

One solution would be a *hybrid search infrastructure* in which DHTs are used for locating rare files at low costs, while preserving the existing unstructured search technologies for searching highly replicated files. We explore this infrastructure in the remaining of this section.

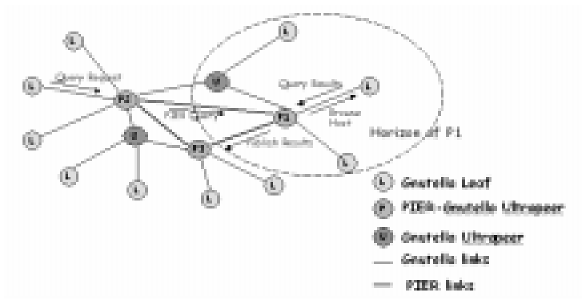

Figure 12: *Hybrid PIER-Gnutella*

Figure 12 shows a design of the hybrid search network. In such a network, a subset of the ultrapeers are *hybrid* nodes. They are hybrid because they utilize both structured and unstructured search techniques. These hybrid nodes are organized together using a DHT. Flooding-based schemes is used for searching highly replicated files, while rare files utilizes a DHT-based approach.

4.1 Preliminary Live Measurements

As a preliminary feasibility study of this system, we deployed 50 PIER ultrapeers on Planetlab [9]. These PIER nodes participate in the Gnutella network as ultrapeers. We call these nodes *PIER-Gnutella ultrapeers* or *PG ultrapeers* in short. These PG ultrapeers gather file information of nodes within their search horizon through two basic mechanisms. Firstly, a more active probing approach which involves issuing browse-host commands to leaf nodes. Secondly, a more passive approach which involves monitoring query results that passes through the ultrapeer. These PG ultrapeers then identify rare files based on heuristics for selective publishing. Queries that failed to locate any files using the regular Gnutella network over a period of time would be executed as a PIER query by the nearest PG ultrapeer.

We show some preliminary results of measurements obtained from these nodes over a period of one hour, where the first 30 minutes are used to populate files in the DHT, and the latter 30 minutes are used to execute Gnutella queries while publishing was still in progress.

Experiment	**Files**	**Queries**
SHJ	196135	225
JI	203315	315

Table 1: Experimental Setup. Total files published and queries issued over a one hour period.

Experiment	**Median**	**Avg**
SHJ	2s	6s
JI	4s	8s

Table 2: Join Algorithms Overheads

Table 1 shows the experimental setup for each join algorithm described in section 3.1. The number of files published and queries issued differed in the two experiments because they depended on the Gnutella load during the time the experiment was carried out. Based on our Gnutella measurements, there is evidence that queries that return few results typically contain rare files in their results set. Hence, as a basic heuristic, on top of publishing the files shared by the immediate leaf nodes, we only published files that belong to queries that return fewer than 20 results. The overall publishing rate was approximately 1 file per second per node. Queries from leaf nodes of the PG ultrapeers that obtained no results after 60 seconds are re-executed as PIER queries. The query duration was set to 300 seconds, and we discarded both Gnutella and PIER results after the query duration. In view of possible copyright infringements, we did not forward results computed by PIER onto the Gnutella network.

Re-executing the query in PIER reduced the number of queries that returned no results by 21% and 18% respectively for the SHJ and JI experiments. The coverage can be improved by running more nodes over a longer period of time to populate the files in the DHT sufficiently.

Table 2 shows the average and median latencies to receive the first result for both the join algorithms. Both join algorithms have better latency numbers compared with that of Gnutella measured in section 2.4.4.

Experiment	Bandwidth	Publish Cost
SHJ	$20KB$	3.8KB
JI	$850B$	4.3KB
Gnutella	140KB	0KB

Table 3: Join Algorithms Overheads

4.2 Bandwidth Usage

To examine the feasibility of this system as it scales up in both data and queries, we measure the bandwidth consumption per query and publishing costs per file.

Table 3 shows the cost of publishing a file measured from our experiments. Storing the filename redundantly using the JI algorithm increased publishing overheads by 10.5%. Publshing at 1 file per second, each ultrapeer incurs 3.8-4.3KBps of bandwidth. To compute bandwidth usage per query, we aggregated the network messages used to compute all matching docIDs per query. We did not take into account the costs of shipping the result tuples back to the query node. Since we were issuing queries for rare files, we expect the number of results to be small and this overhead to be minimal.

The bandwidth usage per query was measured to be approximately **20KB** when SHJ was used. The bulk of the bandwidth costs was dominated by shipping of the Posting tuples between nodes. Currently, we reordered the join such that keywords with fewer posting tuples are intersected earlier. JI avoids a distributed join, and the bandwidth cost is dominated by the query message, which is **750 bytes** on average. A further 100 bytes is incurred by the DHT to identify the node to route the query to. The bandwidth cost of a JI query stays constant even as more files are published in the system.

An unstructured network like Gnutella incurs zero publishing costs (if we ignore the costs of leaf nodes sending information on their files to the ultrapeer). As computed in section 2.4.3, the average cost of a Gnutella query is $140KB$.

4.3 Summary

Overall, our experimental results were encouraging. Despite publishing only 1 file per second, and the use of only 50 hybrid nodes, there was a 20% reduction of leaf node queries that returned no results. On the other hand, relative to the average Gnutella query, searching for rare files using DHTs incur low bandwidth costs and low latencies.

5 Related Work

There has been previous work done on analyzing Gnutella [19, 25]. This work focuses on the behavior of nodes inside the Gnutella file-sharing system, and show that there is significant heterogeneity in bandwidth, availability and transfer rates of nodes. This provided motivation for the use of ultrapeers. However, this work do not take into account the latest improvements in Gnutella. They also do not focus on recall and search latency. The effects on using ultrapeers have been studied in previous work [5, 14]. However, none of these results are based on measurements taken from an actual ultrapeer-based network. A more recent work [18] modeled file-sharing workloads based on the Kazaa P2P download traffic, and observed non-Zipfian distributions in the popularity distribution of content. Their study provided several useful insights about client births and object births driving P2P file sharing workload, and showed significant locality in the Kazaa workload. Our work differs from theirs as we do not look at actual download of content but rather at the query traffic itself.

There have been recent proposals for P2P text search [24, 26, 20, 17] over DHTs. The most ambitious known evaluation of such a system [24] demonstrated good full-text keyword search performance with about 100,000 documents, which is only a fraction of the amount of data served on Gnutella. A feasibility study on DHT-based P2P Web search [22] focuses on the more demanding web corpus (3 billion documents) and a larger query rate (1000 queries per second).

Overnet [8] is a widely used file-sharing system that utilizes the Kademlia [23] DHT. To our knowledge, the Overnet query execution algorithm is not publicly available. While there has been no empirical study of how well Overnet performs in practice, there have been a study focusing on the availability of hosts on the Overnet file-sharing system to understand the impact of IP aliasing on measurement methodology [15].

Despite the recent interests in DHTs, there have been recent proposals [16] that an unstructured infrastructure like Gnutella is effective for searching for highly replicated content and would suffice for a file-sharing environment. The hybrid search infrastructure to improve the coverage of queries for rare files is complementary to their work.

6 Future Work

As part of our future work, we would like to explore the following:

- **Improved Hybrid Implementation.** Experimental results show skews in the bandwidth consumption. This is due to the Zipfian distribution in the keyword and filename distribution. To alleviate this problem, we will explore different load balancing strategies for both storage and computation. We will also explore other selective publishing strategies for rare files, such as those based on term frequencies, and historical statistics of the number of replicas per file. On top of issuing browse hosts to immediate neighbors, we will also explore the use of each PG ultrapeer to perform a crawl of nodes within its horizon and index the rare files.

- **Network Churn.** As nodes join and leave, the index will become stale. Currently, we have focused our attention on recall. Accuracy is also important. We intend to improve the publishing strategies to take into account the lifetimes of nodes in Gnutella. We will also study the effects of network churn on the overheads of maintaining the published index in the DHT.

- **Multiple Ultrapeers per Leaf.** The experiments on the PG ultrapeers did not take into account that each leaf node might have multiple ultrapeers. Sending the leaf query to multiple ultrapeers increases the coverage of the query at the expense of increased query load. In a DHT implementation, only one ultrapeer is required for coverage purposes, although this will increase the chances of a malicious node denying a leaf node query results. We will explore this tradeoffs as part of our future work.

7 Conclusion

In this paper, we performed a measurement study of Gnutella, focusing on its recall and search performance in view of recent improvements to the Gnutella protocol. Our analysis revealed that a moderate fraction of Gnutella queries would benefit from the use of DHTs. We described how keyword based searching can be implemented using DHTs, and provided an implementation using PIER, a P2P query processor over DHTs. We deployed 50 PIER nodes as ultrapeers on Planetlab, and used them reduce the number of immediate leaf node queries that did not locate files in Gnutella. Our preliminary evaluation showed that executing these queries over DHTs incur low bandwidth costs and low latency numbers.

8 Acknowledgments

We would like to acknowledge all members of the PIER project group for their useful feedback and insightful comments.

References

[1] Gnutella. `http://gnutella.wego.com`.

[2] Gnutella Proposals for Dynamic Querying. `http://groups.yahoo.com/group/the\_gdf/files/Proposals`.

[3] Gnutella Proposals for Ultrapeer Query Routing Tables. `http://groups.yahoo.com/group/the\_gdf/files/Proposals/`.

[4] Gnutella Ultrapeers. `http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm`.

[5] Gnutella Ultrapeers. `http://www.grouter.net/gnutella/search.htm`.

[6] Kazaa. `http://www.kazza.com`.

[7] Limewire.org. `http://www.limewire.org/`.

[8] Overnet. `http://www.overnet.com`.

[9] Planetlab. `http://www.planet-lab.org/`.

[10] Query routing for the gnutella network. `http://www.limewire.com/developer/query_routing/keyword/`.

[11] Why Gnutella Can't Scale. No, Really. `http://www.darkridge.com/~jpr5/doc/gnutella.html`.

[12] www.bearshare.com. `http://www.bearshare.com/`.

[13] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in p2p systems.

[14] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. *19th International Conference on Data Engineering (ICDE)*, 2003.

[15] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[16] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *In Proceedings of ACM SIGCOMM 2003*.

[17] O. D. Gnawali. A Keyword Set Search System for Peer-to-Peer Networks. Master's thesis, Massachusetts Institute of Technology, June 2002.

[18] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the 19th ACM Symposium of Operating Systems Principles (SOSP-19)*, Bolton Landing, New York, October 2003.

[19] P. K. Gummadi, S. Saroiu, and S. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems.

[20] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *1st*

International Workshop on Peer-to-Peer Systems (IPTPS'02), March 2002.

[21] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proceedings of 19th International Conference on Very Large Databases (VLDB)*, Sep 2003.

[22] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morrris. On the feasibility of peer-to-peer web indexing and search. In *IPTPS 2003*.

[23] P. Maymounkov and D. Mazieres. Kademlia: A peerto -peer information system based on the xor metric, 2002.

[24] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching.

[25] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[26] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *ACM HotNets-I*, October 2002.