Targeted Search: Reducing the Time and Cost for Searching for Objects in Multiple-Server Networks

Graciela Perera and Ken Christensen
Dept. of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
{gpererao, christen}@csee.usf.edu

Allen Roginsky
National Institute of Standards and Technology
100 Bureau Drive, Stop 8930
Gaithersburg, MD 20899-8930
allen.roginsky@nist.gov

Abstract

In many applications – including peer-to-peer (P2P) file sharing, content distribution networks, and grid computing – a single object will be searched for in multiple servers. In this paper, we find the provably optimal search method for such applications and develop analytical models for search time and cost. A client node searching for objects maintains statistics on where (in which servers) it has previously found objects. Using these statistics to target future searches to popular servers is provably optimal. For object location and request distributions that are non-uniform, which has been shown to be the case in P2P file sharing networks, this method of targeted searching is found to be more cost effective (i.e., use less server resources) than broadcast-based searching. Our targeted search method is implemented in a prototype Gnutella servent called Ditella. Ditella can improve the scalability of file sharing in P2P networks and reduce the amount of traffic in the Internet by reducing file search query traffic.

1. Introduction

Efficiently finding a single object in multiple servers is fundamental to many emerging applications. For example, in peer-to-peer (P2P) file sharing a single file is searched for in multiple "servent" nodes that act as servers (a servent is a node in a P2P network with server and client capabilities). An efficient search returns with the location of the searched object quickly and with a low cost. Cost can be measured as the total server utilization per search. One means for an efficient search is to maintain a centralized directory of objects. A centralized directory is the approach Napster [6] used for P2P file sharing in the Internet. However, for reasons of robustness distributed solutions are typically sought. The simple approach is a fully distributed search based on a broadcast search query to all servers. Gnutella [8] uses this broadcast approach in an overlay network on the Internet. The time to find an object is small, however the cost is significant – all Gnutella servents are queried independently of the likelihood of (the servent) having the searched object. It has been shown that broadcast-based search does not scale well for systems with many servers [10]. As the number of servers, *N*, in a system increases linearly, the load on each server increases exponentially. Currently, at least 20% of the traffic in the Internet is P2P file query related [4]. Thus, to reduce load on P2P nodes and reduce traffic in the Internet new ideas in searching in P2P networks are needed.

A compromise between a centralized directory and broadcast search is to have a fraction of the nodes act as "super nodes" that contain partial or complete directories of where objects are located. Kazaa [3] uses this approach. Coordination between super nodes remains a problem. Another method of reducing the number of servers queried is to employ "random walk" [7] or other limited search methods [12, 13] where a subset of servers is queried in multiple iterations until the object is found. The challenge in such limited searching is to determine which subset of servers to query in each iteration. If objects can be stored in predetermined servers then searching becomes straightforward. Approaches such as Chord [11], which uses the hash of an object name to determine which server it is to be stored in, impose a structure on where objects are stored (and thus also found). These structured approaches depend on resilient servers, which are not typical in P2P networks.

Characterization of P2P networks has shown that connectivity of servents to other servents follows a power law where very few servents are high degree and the majority of servents are low degree [2]. File sharing also follows a power-law where few servents contain the majority of files shared (and many servents may share no

The use of Gnutella, Ditella, or any other product described herein does not represent its endorsement by the US Federal Government.

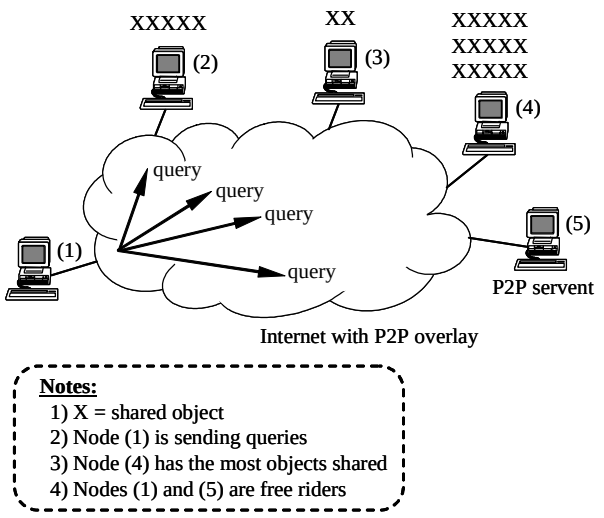

Figure 1. P2P network showing shared objects

files at all and are so called "free riders") [9]. These characterizations clearly show that connectivity and file distribution between servents are not uniformly distributed. Exploiting these probabilistic characteristics can result in greater search efficiency. Hybrid Periodical Flooding [13] and Directed BFS [12] direct their queries through neighbor nodes from where a query hit has been received. We seek to exploit the non-uniform distribution of objects in servers to improve multi-server search. Our hypothesis is that individual nodes can "learn" the distribution of objects in servers as a function of their request distribution and use this learned knowledge to improve searching.

The remainder of this paper is organized as follows. Section 2 describes optimal search and our targeted search method. Section 3 is a performance evaluation of targeted search. In Section 4 we describe the Gnutella-compatible Ditella P2P prototype servent we have developed. Section 5 is a summary and future work. An appendix contains proofs.

2. The targeted search method

In this section we show that an optimal search is possible. We develop a targeted search method based on optimal search. The targeted search method exploits a non-uniform distribution of objects in nodes to achieve an efficient search in terms of search time and cost. Fig. 1 shows a P2P network with multiple servents that share objects. The number of objects that a servent shares follows a distribution. This distribution is often peaked (or may even follow a power law relationship) [9]. A power law relationship between two quantities x and y is $y = ax^k$ where a and k are constants. A power law relationship appears as a linear fit on a log-log plot. Pareto and Zipf distributions follow a power law relationship. In fig. 1, servent (4) shares 15 objects, servent (2) 5 objects, servent (3) 2 objects, and servents (1) and (5) do not share any objects. Servents (1) and (5) are thus free riders. Servent (4) shares more files than all other servents combined. This attribute of few servents sharing the most files and most servents sharing the least files is characteristic of a power law.

2.1 Optimal search

We wish to determine the optimal search method for a given time constraint k. That is, our search must complete within no more than k steps. Each step can comprise the sending of a query (to one or more servers) and a wait for a response. The response will indicate whether a queried node contains the searched for object. Suppose that we have N servers and a single object that is located in exactly one server and the probability of it being located in any given server is $1/N$. At each of k steps we check $s(1), s(2), \ldots s(k)$ servers, so that

$$s(1)+s(2)+\cdots+s(k)=N. \qquad (1)$$

The condition in eq. (1) assures us that all servers are checked in the worst possible case (otherwise, the time constraint of k would not be satisfied.) The choice of the $s(i)$'s is our strategy. They can be any non-negative integers as long as eq. (1) holds. The cost C of finding our object is $s(1)+s(2)+\cdots+s(j)$, where j is the step where the object was discovered. The cost C is a random variable. We want to find a strategy, that is a set of $s(1), s(2), \ldots s(j)$ that minimizes the expected value of C, denoted here $E(C)$.

Lemma 1. The cost $E(C)$ of strategy $s(1), s(2), \ldots s(j)$ is

$$E(C)=N/2+\left(s(1)^2+s(2)^2+\cdots+s(k)^2\right)/2N. \qquad (2)$$

Lemma 2. The strategy $s(1), s(2), \ldots s(j)$ where $E(C)$ takes its minimum is such that

A. If N is a multiple of k, $s(1)=s(2)=\ldots=s(k)=N/k$ and $E(C)=N(k+1)/2k$.

B. If N is not a multiple of k then we set $s(1)=s(2)=\ldots=s(k-m)=(N-m)/k$ and we set $s(k-m+1)=s(k-m+2)=\ldots=s(k)=(N-m)/k+1$ where we define m as the smallest positive integer such that $m=N(\mathrm{mod}\,k)$.

Statement A is a special case of statement B. The optimal cost does not depend upon the order of the $s(j)$'s. The proofs of lemma 1 and lemma 2 are in the appendix.

If an object is found at step *j* then the cost of finding it is $a(j) = s(1) + s(2) + \cdots + s(j)$. For non-uniformly distributed objects where the above lemmas hold, we now weight the search as

$$E(C) = \sum_{i=1}^{k} a(i)P(C = a(i)) = \sum v(i)(p(v(i-1)+1) + \cdots + p(v(i))) \quad (3)$$

where

$$v(i) = \sum_{j=1}^{i} s(j) . \quad (4)$$

Here $p(i)$ is the probability of finding our file in bucket *i*, $p(1) \geq p(2) \geq ... \geq p(N)$ and

$$\sum_{i=1}^{N} p(i) = 1 . \quad (5)$$

2.2 Targeted search

The targeted search method uses a frequency list to direct queries to servers with a high probability of containing an object. The frequency list is a sorted list of tuplets <server_id, hit_count> which is sorted in descending order by hit_count. The value of hit_count is how many previous queries were successful (i.e., resulted in an object being found in this server). Thus, servers are ranked in the frequency list by order of previous search success.

The targeted search method, which executes in a searching node, is shown in fig. 2. In step #1 each query sent is followed by a time-out period during which a response is waited for. A response indicates that the searched for object has been found and comes directly from the server that contains the object. In step #2 a time-out is also used to wait for a response. The frequency list is updated at the end of step #3. In the update, the tuplet with the matching server_id has its number of hits incremented. If the server_id is not in the list, then a new tuplet is created for the new server_id with hit_count set to 1 and this tuplet added to the end of the frequency list. Sorting of the frequency list is of low complexity. For a non-uniform distribution of file location, most sorting occurs in the N_{top} entries and involves at most one change in location to update the list.

Step 1: Send a direct query iteratively to the servers in the frequency list starting with the first listed server. This step terminates when N_{top} servers have been queried or the object has been found.

Step 2: If Step 1 did not find the object then broadcast a query to all *N* servers.

Step 3: If Step 1 or Step 2 found the object in a server then download the object and update the frequency list.

Figure 2. The targeted search method

3. Performance of targeted search

In this section we develop an analytical model of targeted search for a peaked distribution of objects. Using this analytical model, numerical results are generated to show mean time and cost to find a object. A simulation model is used to study performance for cases where the analytical model cannot be used. We assume *N* servers with all objects uniquely stored in these servers (i.e., there are no duplicated objects between servers). A requesting node will send queries directly (and one at a time) to up to N_{top} number of servers ($N_{top} \leq N$) and will then, if the object has not yet been found, broadcast a query to all *N* servers. This is the basic function of targeted search as described in the previous section.

3.1 Analytical model and numerical results

We first develop a distribution to be used to vary object distribution from uniform in all servers to peaked in one server. For *N* servers, a uniform distribution has $\Pr[\text{file in server } i] = 1/N$ for $i = 1,2,\ldots N$. We introduce a parameter *K* to adjust a uniform distribution so that $\Pr[\text{file in server } i = 1]$ is *K* times greater than $\Pr[\text{file in server } i > 1]$. That is,

$$\Pr[\text{file in server } i] = \begin{cases} \dfrac{K}{N+K-1} & i = 1 \\ \dfrac{1}{N+K-1} & i = 2,3,\ldots N \end{cases} \quad (6)$$

For $K = 1$, eq. (6) describes a uniform distribution. For *K* increasing, $\Pr[\text{file in server } i = 1]$ approaches 1 (and $\Pr[\text{file in server } i > 1]$ approaches 0). This models a peaked object distribution similar to that observed in operational P2P networks used for file sharing [9]. We assume that the frequency list in the targeted search

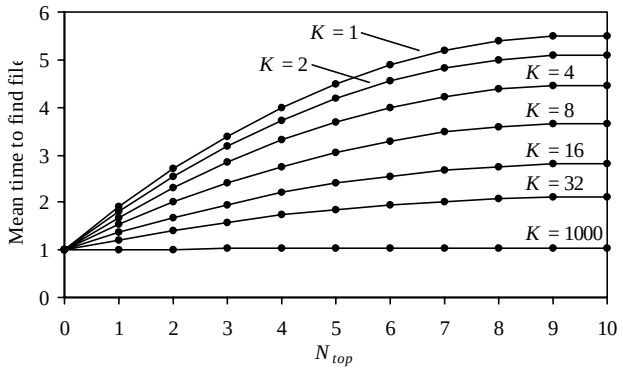

Figure 3. Mean time results for N = 10

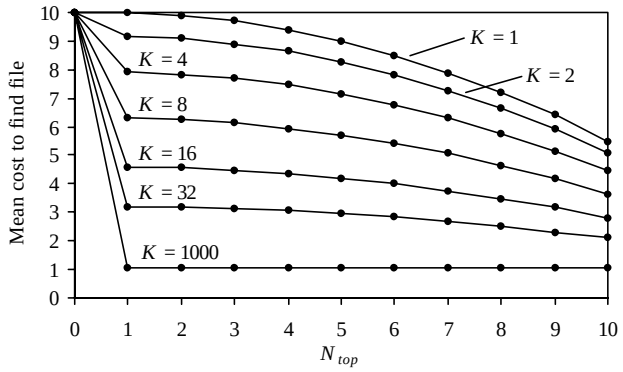

Figure 4. Mean cost results for N = 10

requesting node empirically matches the distribution of files in servers. This will occur after "many" searches. Later in this section we explore this assumption and evaluate how many searches are needed for the frequency list to approach the actual object location distribution.

The time to find an object is the number of queries sent until the object is found. If the object is found in the first query, the time is 1. For direct queries, the maximum time is N (i.e., all servers are queried and the object is found in the last server). For a broadcast query the time to find an object is 1. The cost to find a object is equal to the number of servers queried. A broadcast query has cost N (and time 1). Thus, the tradeoff in the targeted search method is time versus cost. The value of N_{top} can be used to control this tradeoff.

The mean time to find a object in targeted search is:

$$E[\text{time}] = \sum_{j=1}^{N_{top}} j \cdot f(j) + (N_{top} + 1)\left(1 - \sum_{j=1}^{N_{top}} f(j)\right) \qquad (7)$$

where $f(i) = \Pr[\text{file in server } i]$. The first term in eq. (7) is the mean time for direct queries, the second term captures the probability of not finding the object in the N_{top} direct queries (and thus incurring an additional time of 1 to the already expended time of N_{top}). For $f(i)$ as eq. (6), eq. (7) simplifies to the closed form:

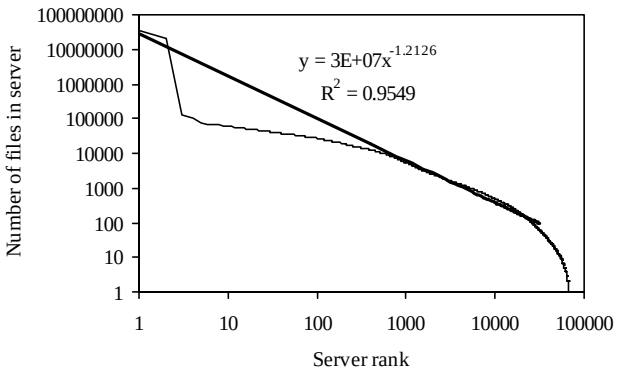

Figure 5. Rank versus number of files for trace

$$E[\text{time}] = \frac{2N_{top}N - N_{top}^2 - N_{top} + 2 \cdot N + 2K - 2}{2(N + K - 1)} \qquad (8)$$

The mean cost to find an object is:

$$E[\text{cost}] = \sum_{j=1}^{N_{top}} j \cdot f(j) + (N_{top} + N)\left(1 - \sum_{j=1}^{N_{top}} f(j)\right) \qquad (9)$$

For direct queries cost and time are the same. For a broadcast query, the cost is N (since all N servers are queried). Thus, in the second term in eq. (9) an additional cost of N is incurred to the already expended cost of N_{top} with the probability that the object was not found in the first N_{top} direct queries. For $f(i)$ as in eq. (6), eq. (9) simplifies to:

$$E[\text{cost}] = \frac{2N^2 - N_{top}^2 + N_{top} + 2K - 2}{2(N + K - 1)} \qquad (10)$$

These equations (eqs. (7) to (10)) were compared with results from a simulation model and found to match.

Using the expressions for $E[\text{time}]$ and $E[\text{cost}]$ we can study the effect of N_{top} and K (peakness) on the performance of targeted search compared to a fully broadcast search. A broadcast search will always have $\text{time} = 1$ and $\text{cost} = N$. For $N = 10$, K increasing, and $N_{top} = 1,2,\ldots N$ fig. 3 shows $E[\text{time}]$ and fig. 4 shows $E[\text{cost}]$. It can be seen that as N_{top} increases, the mean time increases and the mean cost decreases. As K increases, both the time and cost decrease. For K very large the object is always found in the first server queried and thus $E[\text{time}] = 1$ and $E[\text{cost}] = 1$ for all $N_{top} > 0$.

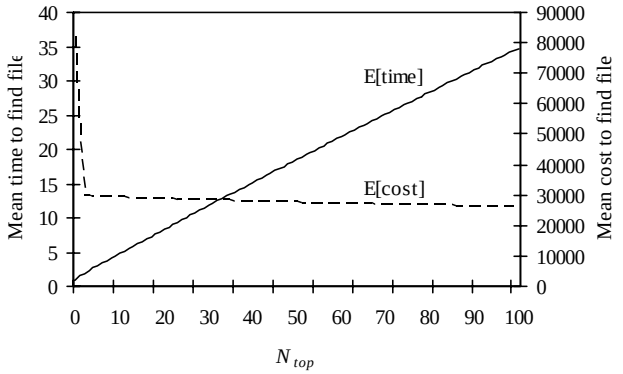

Figure 6. Mean time and cost results for trace

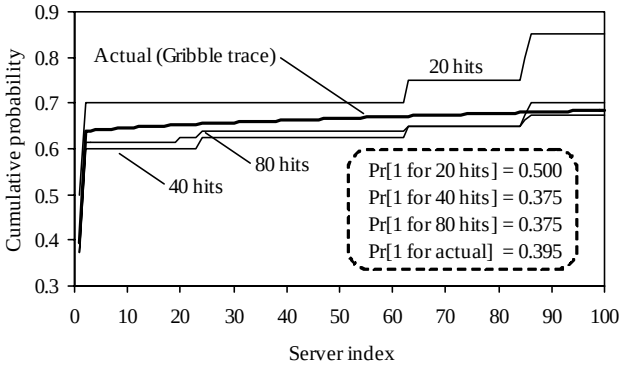

Figure 7. Convergence to actual for trace data

3.2 Numerical results for representative case

The evaluation of Section 3.1 used a theoretical distribution. To evaluate targeted search for a representative case, a trace file collected from a real Gnutella network was used to empirically form a file location distribution. The trace file used comes from an eight day trace collected by Saroiu and Gribble from the University of Washington [9]. The trace was collected from May 6 to May 14, 2001 and contained the number of files shared by each of 82281 Gnutella servents. The total number of files shared was almost 85 million. The average number of files shared was 1032. It was found that 93.5% of the hosts shared less than 1000 files while the maximum number of files shared was 33.5 million and the minimum was 0. We do not know how many files were unique (i.e., not duplicated between multiple servents). Fig. 5 shows the rank versus number of files shared on a log-log plot. The linear fit shown on the graph indicates that file distribution in servents has the power law property. For our performance evaluation, we assumed that all files were unique. Duplication of files between nodes can be expected. However, some of our own trace results show that duplicates are rare. A trace of 987 Gnutella Servents was collected for 3 days in July 2004. Of the 331096 files discovered available for sharing, 97.8% of them were unique.

Numerical results for targeted search for the Saroiu and Gribble trace data are shown in fig. 6. The solid line shows $E[\text{time}]$ and the dashed line shows $E[\text{cost}]$ for $N_{top} = 1,2,\ldots 100$. For a broadcast query the time is 1 and cost is N (N = 82281 for this evaluation). The results show that targeted search in a real P2P network can significantly reduce search time and cost when compared to broadcast search. For example, for $N_{top} = 2$ the search time is roughly doubled, but the cost is reduced by 63% (on average 29769 nodes are queried and not the full 82281 nodes). This is seen in fig. 6 as a sharp drop in the $E[\text{cost}]$ as N_{top} increases.

3.3 Comparison to the random walk method

In random walk search [7] one or more "walking queries" are routed through the P2P network. A walking query randomly chooses nodes. Previously queried nodes in a given search are not re-queried. Thus, random walk is random sampling without replacement where there are M samplers (walkers). The mean time and cost for a single random walker is $(N+1)/2$ and is independent of the distribution of files. As the number of walkers, M $(1 \le M \le N)$, is increased the time decreases proportionally but the cost remains the same. For M much less than N we have $E[\text{time}] = (N+1)/(2M)$ and $E[\text{cost}] = (N+1)/2$. For the trace data, random walk has significantly greater $E[\text{time}]$ for all values of M than does targeted search for all values of N_{top}. $E[\text{cost}]$ is greater for random walk for most values of N_{top}.

3.4 Convergence of the learned frequency list

The targeted search nodes build their frequency lists by learning from previous searches (see step 3 in the targeted search method in fig. 2). We evaluate how fast the learned frequency list converges to the actual frequency list (the distribution of files by location). Using simulation on the Saroiu and Gribble trace data, we uniformly randomly choose 20, 40, and 80 files, located them in the simulated servers, and then plotted the resulting cumulative probability of the learned frequency list. Fig. 7 shows this plot for the first 100 servers (of 82281 servers in the trace data). The heavy line is the actual. It can be seen that after to 40 to 80 updates, the learned frequency list is converging very closely to the actual frequency list. This is expected for the heavy tailed case where the first two servers contain almost 2/3 of all files shared. In fig. 7 we list the probability for a file being found in the first server ("1") and note how this also converges quickly to the actual.

4. Ditella – targeted search in Gnutella

We have implemented the targeted search method in a servent software release named Ditella [1]. The name Ditella comes from the prefix *di* and suffix *tella* meaning that which is woven or the webbed communication between peers. The Ditella servent is compatible with Gnutella servents that use the Gnutella protocol v0.4. Ditella is written in C (and is about 800 lines of code) for the Microsoft WindowsXP platform. The Ditella specification, source code, and executable can be found on the project web site [1].

Ditella directly queries servents where a file has been previously found (i.e., using the targeted search frequency list) before broadcasting a query to all servents. The connection to the P2P network is accomplished by a three-way handshake and a bootstrapping procedure. Once connected to the network, a Ditella servent can issue and respond to messages in a Gnutella-like fashion. Ditella will first use directed queries before a broadcast is issued. The directly queried servents are selected using the statistics of successful searches in a frequency list. The purpose of the prototype is to test the feasibility of directly asking a servent for a file before flooding.

Before a file search is issued the Ditella prototype must first connect to the Gnutella network by asking the user for the IP address of a known Gnutella servent. The servent is then able to accept connections from other servents, send Pings, send and forward Pongs, send a Query, identify a QueryHit, and download a file using HTTP. Each connection to a servent is handled by a separate process so it can have parallel TCP connections open and a separate process is created to accept user input so performance can be optimized for response time. The implementation maintains statistics on the nodes responding to Pings as well as those responding with QueryHits. The statistics are kept in two files. The `statistics.txt` and the `pong.txt` files. The first file maintains the IP address, the port number, the file size and the file names received from QueryHits. This file is used to create and update the frequency list for the targeted search method. The second file stores the responses to ping requests created or forwarded by the servent and are used to verify connectivity to the network. We are currently implementing targeted search in LimeWire, a popular open source Gnutella client.

5. Summary and future work

The distribution of files to servers in P2P networks is well known to follow a power law. Few servents have most files and most servents have few files. This known distribution of files can be exploited in order to reduce search cost (e.g., the number of query packets in the network) at only very small expense in increase of search time. Our targeted search method outperforms broadcast and random search by "remembering" in which servents files have been found in and building from this a frequency list. This frequency list is then used to send direct queries to the N_{top} highest ranked servents. If the file is not found in these N_{top} servers (and in the far majority of cases the file is found due to the power law distribution of files in servers), only then is a broadcast search used. With targeted search, the reduction in traffic (in Query packets) is directly proportional to the degree of peakedness of this underlying distribution of files.

The value of N_{top} affects the trade-off of time and cost. Future work will address automatic setting of this value. Future work will also further evaluate the implementation of targeted search in a Ditella servent (and in our new LimeWire implementation) and study the influence of both network and overlay topology on P2P system performance.

Acknowledgment

The authors thank Stefan Saroiu and Steven Gribble at the University of Washington for sharing their trace data used in the evaluation of targeted search. The authors also thank Sasha Dos Santos at the University of South Florida for her assistance in developing the Ditella prototype.

References

[1] Ditella, 2004. URL: http://www.csee.usf.edu/~gpererao/Ghybrid1001.htm.

[2] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power Law Relationships of the Internet Topology," *Proceedings of SIGCOMM*, pp 251-262, 1999.

[3] Kazaa Media Desktop, 2004. URL: http://www.kazaa.com/us/index.htm.

[4] T. Karagiannis, K. Claffy, and M. Faloutsos, "File-Sharing in the Internet: A Characterization of P2P Traffic in the Backbone," Technical Report, UC Riverside, November 2003.

[5] LimeWire.org, 2004. URL: http://www.limewire.org.

[6] A. Loo, "The Future of Peer-to-Peer Computing," *Communications of the ACM*, Vol. 46, No. 9, pp. 56-61, September 2003.

[7] L. Qin, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proceedings of the 16th International Conference on Supercomputing*, pp. 84-95, 2002

[8] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design," *IEEE Internet Computing*, Vol. 6, No. 1, pp. 50-57, January-February 2002.

[9] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proceedings of SPIE*, pp. 156-170, 2001.
[10] R. Schollmeier and G. Schollmeier, "Why Peer-to-Peer (P2P) Does Scale: An Analysis of P2P Traffic Patterns," *Proceedings of the 2nd International Conference on Peer-to-Peer Computing*, pp. 112-119, 2002.
[11] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger., M. Kaashoek, F. Dabek, and H. Balakrishnan,. "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking*, Vol. 11, No. 1, pp. 17-32, February 2003
[12] B. Yang, P. Cao, and H. Molina, "Efficient Search in Peer-to-Peer Networks," *Proceedings of the International. Conference on Distributed Computing Systems*, pp. 5-14, 2002.
[13] Z. Zhuang, Y. Liu, L. Xiao, and L. Ni, "Hybrid Periodical Flooding Search in Unstructured P2P Systems," *Proceedings of the International Conference on Parallel Processing*, 2003.

Appendix – Proofs for the lemmas

Proof of Lemma 1: If the object is found at step *j*, then the cost of finding it is $a(j)=s(1)+s(2)+\cdots+s(j)$. Let us find the probability of finding our object at step *j*.

$$\Pr[\text{the file is not found at steps } 1,2,\ldots j-1 \text{ and the file is found at step } j]$$

$$=\Pr[\text{the file is not found at steps } 1,2,\ldots j-1 \text{ and (the file is found at step } j \text{ given that the file is not found at steps } 1,2,\ldots j-1)]$$

$$=[1-(s(1)+s(2)+\cdots+s(j-1))/N][s(j)/(s(j)+\cdots+s(k))]$$

$$=(1/N)[s(j)+\cdots+s(k)][s(j)/(s(j)+\cdots+s(k))]$$

$$=s(j)/N.$$

Therefore,

$$E(C)=\sum_{i=1}^{k}a(i)\Pr[C=a(i)]=\sum_{i=1}^{k}a(i)\frac{s(i)}{N}$$

$$=\frac{1}{N}\sum_{i=1}^{k}\sum_{j=1}^{i}s(j)s(i). \qquad \text{(A1)}$$

If $E(C)$ is doubled, $2E(C)=[2s(1)^2+2s(2)(s(1)+s(2))+ +2s(3)(s(1)+s(2)+s(3))+\cdots+2s(k)(s(1)+\cdots+s(k))]/N = [s(1)^2+s(2)^2+\cdots+s(k)^2+N^2]/N$ and the statement of Lemma 1 now immediately follows from this. End of proof of Lemma 1.

Proof of Lemma 2: Let $s(1), s(2),\ldots s(k)$ be the optimal strategy in terms of minimizing $E(C)$ under the constraint that the search time does not exceed *k*. Such an optimal strategy exists since there are a finite number of strategies. If more than one strategy leads to the same $E(C)$ we can choose any one of them. We will show that for any *i*, *j*, it is true that

$$\text{abs}(s(i)-s(u))\le 1 \qquad \text{(A2)}$$

Indeed, if eq. (A2) were not true then we find some *i*, *j*, such that $s(i)-s(j)$ is greater than or equal to 2. Then we can find another strategy with $s(i)-1$ instead of $s(i)$ and $s(j)+1$ instead of $s(j)$ and the value of $E(C)$ will be reduced by $[s(1)^2+s(j)^2-(s(i)-1)^2-(s(j)+1)^2]/2N = [s(i)-s(j)-1]/N>0$, so the initial strategy was not optimal. Hence all $s(i)$'s are different by no more than one and they also have to satisfy $s(1)+s(2)+\cdots+s(k)=N$. This defines them uniquely. Let us prove this and find them. We suppose that $k-m$ of the $s(i)$'s are equal to some number *n* and the remaining *m* of them are equal to $n+1$. Then we have $(k-m)m+m(n+1)=N$, hence $kn+m=N$, so $n=N(\text{mod}\,k)$ and $n=(N-m)/k$. This gives us the values of the $s(i)$'s. End of proof of Lemma 2.