Search Performance Analysis and Robust Search Algorithm in Unstructured Peer-to-Peer Networks*

Tsungnan Lin, Hsinping Wang, Jianming Wang
Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan
tsungnan@ntu.edu.tw

Abstract

Recently Peer-to-Peer networks (P2P) have gained great attention and popularity. One key challenging aspect in a P2P resource sharing environment is an efficient searching algorithm. This is especially important for Gnutella-like decentralized and unstructured networks due to the power-law degree distributions. We propose a hybrid search algorithm that decides the number of running walkers dynamically with respect to peers' topological information and search time state. It is able to control the extent of messages generating temporally by the simulated annealing mechanism, thus being a scalable search. Furthermore, we present a unified quantitative search performance metric, Search Efficiency, to objectively capture dynamic behavior of various search algorithms in terms of scalability, reliability and responsiveness. We quantitatively characterize, through simulations, the performance of various existing search algorithms. The proposed algorithm outperforms others in terms of Search Efficiency in both the local and global search spaces.

1. Introduction

Recently, a newly innovated architecture of Peer-to-Peer [5, 6, 7, 8, 9] networks has caught people's eyes and becomes popular, in which all participants are functionally equivalent. Unlike the centralized client-server model, the peers in the P2P network behave as clients and servers at the same time. P2P networks adopt a network-based computing style that neither excludes nor inherently depends on centralized control points. The unstructured designs are not based on any kind of network connection and much resilient to nodes entering and leaving the system. This feature is attractive to general users due to the capability to install, maintain, and share files easily [8, 9]. The rapid growth of Gnutella [1] is an example for the appealing architecture. Recent measurement data suggests that P2P applications have a significant impact on the Internet traffic [3, 4].

In this paper, we focus on Gnutella-like decentralized unstructured P2P environments since these systems are actively used by a large community of Internet users today [13, 14, 22]. These networks, while not centrally planned in structure, grow according to a simple self-organizing process, whose topologies have been shown to have power-law properties [1, 13, 15].

A resource discovery mechanism returns a set of resources that match the query descriptions. The name of a target file may be known, but due to the network's ad hoc nature, the node holding the file is not known until a real-time search is performed. In order to find files, peers pass messages along to the other peers that they know of. Devising an efficient searching algorithm is challenging because of the potentially large number of resources and users, and the heterogeneity in resource types as well as the complication that users may join and leave randomly. Therefore, the major focus in this paper is to explore an efficient search algorithm in a large-scale, dynamic, and heterogeneous networking environment.

Another focus of this paper is to provide an objective and overall evaluation metric for search performance. Hence, we propose a quantitative measure criterion, *Search Efficiency* (SE), to give a unified measure from both users' and networks' perspectives in terms of reliability, scalability, and responsiveness.

Current search algorithms in Gnutella-like networks tend to be inefficient, either generating too much load on the system [2, 16], or not meeting users' requirements [11]. Hierarchical structure like KaZaA [22] can reduce network traffic by using supernodes. However, the query operations among supernodes are the same as Gnutella networks. Specifically, flooding search, deployed by Gnutella, is known to have the scalability problem [2, 13]. It suffers from the exponentially growing number of search messages. On the other hand, it has been suggested random walk

* This work was supported in part by NSC grant NSC92-2213-E-002-087.

0-7803-8430-X/04/$20.00 ©2004 IEEE

algorithm [10, 12] can solve the scalability problem. We find that, in our simulations, *Search Efficiency* of random walk almost remains the same regardless of the search time because the number of the query messages (walkers) remains constant regardless of the network topology. However, random walk suffers from poor SE in the short term although it does have higher SE compared to that of flooding search in the long term. Besides, it is difficult to determine the optimal number of walkers in a dynamic environment in advance.

We therefore propose a hybrid search algorithm trying to solve the problem of scalability and how to determine the optimal number of running walkers. The proposed mechanism can decide the number of walkers dynamically with respect to the peer's linking status and the search time, making it possible to decide the optimal number of query messages. More specifically, the hybrid search fully explores the local search space to get responsive results as flooding, and control the impact on the network in the global space, as random walk, by a "simulated annealing" mechanism, making itself a scalable search. Simulation results demonstrate the proposed mechanism outperforms existing search algorithms in terms of SE in both long term and short term.

In addition, to mimic the P2P characteristics, especially the peers' file sharing behavior, we further introduce the idea of file-to-node distributions. It models this behavior as that peers with higher connection ability tend to share more files, and vice versa. This is true especially when there is some incentives for users to share files [22]. Some interesting results are shown in the analysis sub-section about the properties of algorithms, under this distribution.

The rest of this paper is organized as follows. Section 2 explains the existing search algorithms for unstructured P2P networks. In Section 3, we describe in details the proposed search algorithm. The P2P environment modeling and simulations are described in Section 4. Before presenting our simulation results, we introduce the unified evaluation criterion, *Search Efficiency*, in Section 5. Performance analysis of various search algorithms is presented and discussed in Section 6. In Section 7, we introduce the file-to-distribution and discuss the performance natures of search algorithms. Finally, Section 8 concludes our work.

2. Related previous work

In this section, we briefly introduce some existing search algorithms in P2P systems that will be used for the performance comparison in the analysis sections.

Flooding is a search algorithm currently adopted in Gnutella [18]. It operates by sending query messages to all nodes within a neighborhood of size TTL (time-to-live: the number of times or hops a query will be forwarded). Every node passes on the query message to all of its neighbors and decrements the TTL by one. The process continues until TTL reaches zero.

Expanding ring [10, 11] is an extension of flooding search algorithm. It uses successive flooding searches with incremental TTLs. A node starts a flood with a small TTL and waits to see if a success is replied. If it is, the node stops searching; if not, the node continues to flood with larger TTLs. This procedure is repeated till the target object is found or exceeds the maximum TTL limit.

Random walk [10, 12, 17] has been proposed to avoid the scalability problem and generates reasonable search performance. The mechanism of random walk is to forward the query to a randomly chosen neighbor at each step until the object is found, where the query message is called a "walker". By doing so, the number of query messages at each time instance remains constant as the search time elapses (hop count increases). Intuitively, just one walker will hardly find the desired object and thus [10] modifies it to a k-walker algorithm in order to cut down the delay by a factor of k. In this paper, we use "random walk" to refer to multiple random walks.

3. A robust search: hybrid search algorithm

On one hand, flooding search algorithm sends query messages aggressively to all neighbors, producing rather high success probability thanks to its global coverage if the target object exists. However, the number of its query messages grows exponentially and unnecessary messages are generated during the search process. On the other hand, random walk search can be viewed as a local search method, which significantly reduces the number of messages generated. Its potential drawbacks are that the route of a query message may become a loop and messages never jump out as well as that it is hard to choose the optimal number of walkers in advance.

To devise a robust search algorithm, which possesses high success probability with low message overhead, is a challenging task.

We propose an algorithm, which takes approaches similar to the mechanism called simulated annealing [20], to explore the unknown performance space between random walk and flooding, from which we could hopefully gain performance advances.

The term "simulated annealing" derives from the roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure. It is a popular combinatorial optimization method, which accepts system changes depending on the cost function. If the energy of this new state is lower than that of the previous one, the change is accepted unconditionally and the system is updated. If the energy is greater, the new configuration is accepted probabilistically. This procedure allows the system to move consistently towards lower energy states, yet still "jump" out of local minima due to the probabilistic acceptance of some upward moves during the first few iterations.

To clearly illustrate how the idea of simulated annealing is applied in our proposed algorithm, we first bring in the mathematic model of forwarding mechanism. Basically, the

mechanism of forwarding a query message to the ith neighboring peer can be described by a probability function $p_i(t,o_i)$, where t denotes the current searching time (hop count) and o_i represents the statistic information of the ith peer. Note that we define the time a peer initiates a search as $t = 0$. When $p_i(t,o_i) = 1$, the query message is forwarded to the ith peer. If $p_i(t,o_i) = 0$, the query is not. Assuming the number of links (neighbors) of certain peer is l, we obtain the number of messages the peer will forward at some search time t as

$$\sum_{i=1}^{l} p_i(t) = k, \qquad (1)$$

where k is the number of forwarding walkers of the peer statistically. For pure flooding search algorithm, the probability model, which every node adopts, can be described as $\forall i, p_i(t,o_i) = 1$.

If the termination parameter TTL (a fixed number) is used, then the probability function should be modified as $(1-u(t-TTL))\cdot p_i(t,o_i)$, where $u(t)$ is the unit step function ($u(t) = 1$, when $t \geqq 0$, $u(t)=0$, otherwise). In the case of random walk search algorithm, the function $p_i(t)$ does not depend on the knowledge of its peers since the algorithm randomly selects its peers to relay the query. Moreover, there is one additional constrain, $\sum_i p_i(t) = 1$, to let each node only forwards the received message to one neighbor.

One potential problem of local search algorithms is that the relay path of the query message may form a loop (The situation is similar to the local minimum problem in gradient-based searching process). Therefore we take a similar approach to simulated annealing that the searching mechanism should explore the search space globally when the temperature is high, which means the search just begins. Specifically, the proposed algorithm floods network when the search time is low. We use the parameter n to specify that, within initial n search steps (hops), the search space should be global, relaying the query messages to all the peers. When the search time is larger than n, the algorithm then searches the network locally. In this case, the probability model that a node will relay a query message to the ith peer becomes

$$(1-u(t-n))+u(t-n)\cdot p_i(t,o_i). \qquad (2)$$

In our design of the hybrid search, we limit nodes to relay queries only to one of its neighbors when the search radius is out of n. That is, if the number of links (neighbors) of a node applying the hybrid search is l, we define our forwarding probability model as

$$(1-u(t-n))+u(t-n)\cdot 1/l \qquad (3)$$

We let $p_i(t,o_i)$ in (2) to be $1 / l$ in order to make the search behaves as random walk since summation of (3) yields 1 when $t \geqq n$. For example, we set n to be 3. If a node receives the query message at $t = 2$, then it forwards to all its peers. If a node receives the query message at $t = 5$, it will randomly select one of its peers to relay the query. (On average, it behaves this way, or we can use a post-limit to make it behaves as random walk. Our implementation is the latter.) Specifically, the search mechanism shows a hybrid phenomenon between flooding and random walk algorithm. In this example, it first floods the local network globally (when the search time $t < 3$), and then the queries are relayed in the global network locally.

In the pure random walk search, it is difficult to determine the optimal number of walkers in advance. The hybrid approach, however, provides the flexibility to decide the number of walkers depending on peers' topology and search time state as well as the choice of n. If a user wants to search results aggressively, he or she can set a lager number of n. After firing off a search, the number of walkers is determined topologically and based on local connections of the searching node within the radius of n.

4. P2P network modeling and simulation

We use simulations to evaluate the search performance of various search algorithms. To get convincing results, the simulation environments are based on real characteristics of Gnutella network measured by [17, 21].

Although the methodology used in this work is static and the real peer-to-peer network is rather dynamic, the statistics of network sizes, node link degrees, and object replication and distribution are pseudo-static in the long-term view. Therefore, our static simulations would still catch the features of various searching algorithms within the dynamic peer-to-peer networks.

4.1. P2P environment setup

In order for solid performance analyses of P2P systems, we have built a simulator, modeling a static P2P file-sharing system, embedded with different search algorithms to perform comprehensive analyses. Modeling aspects in this simulator include the modeling of Gnutella topology, peer file-sharing behavior, peer query behavior, all of which are essential ones for critical performance comparison and breakdown. Topology modeling strictly follows the measured results in [17], which has suggested a topology of two-stage power-law distributions (Gnutella), other than the expanded network size. Note that we have tried network sizes from 1,000 to 200,000, all of which give similar results as long as the two-stage power-law distributions are followed. The resulting link degree distribution of 10,000 nodes is plotted in Figure 1 and the statistics of mean link degree are 6.05 with maximum degree of 199 and standard deviation of 13.09. The links between nodes (peers) are randomly connected and we reassure that every node is connected (has at least one link).

To model the peer file-sharing behavior, we assume there are 100 distinct objects with 100 replicas each, thus totally 10,000 objects in the network. These objects are randomly assigned into the 10,000 nodes, without duplicated sharing. The resulting cumulative density function (CDF) of the number of objects is plotted in

Figure 2, in which the characteristics resemble the ones measured in [21]. 25% of nodes, in this graph, share nothing (this illustrates the facts that Gnutella has an inherently large percentage of free-riders in spite of claims of that every peer is both a server and a client), 35% of nodes share only one object, and only 1% of nodes share more than six objects. Finally, for simplicity reasons, the query behavior of peers is uniformly modeled; that is, each object is of the same popularity and thus randomly queried.

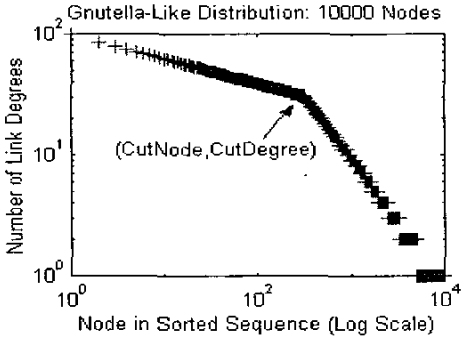

Figure 1. Link degree distribution of Gnutella topology embedded in our simulator

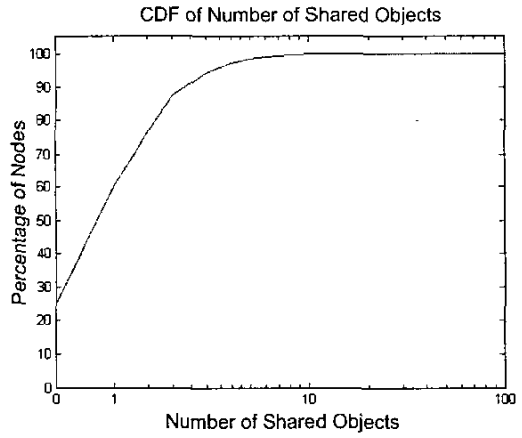

Figure 2. CDF of the number of shared objects across nodes in our simulations.

The more complicated modeling of peer behavior of various query/replication distributions presented in [10] has also been explored in our private experiments, but due to space limit, we only show and analyze the results of uniform/uniform distributions (random/random), which actually capture the general behavior and give representative results.

4.2. Simulation performance metrics

In simulations, the simulator collects several statistics to evaluate the performance of various algorithms. Specifically, *MsgPerNode*, *Coverage*, *SuccessRate*, *HopNum*, and *QueryHits* are the metrics that are used to judge the search performance.

MsgPerNode is the total number of incurred query messages normalized by the *NetworkSize* during the search process, where *NetworkSize* is the number of nodes in the network. *Coverage* is the number of nodes that receive the query messages for the search. *SuccessRate* is the chance that each search is able to return results successfully. *HopNum* abstractly represents the response time in which a search first finds the desired object (one-way count). *QueryHits* is the count of query messages, which find the target object.

Basically, these metrics address the performance issues from both network's and users' perspectives. From the network's perspective, *MsgPerNode* should be as low as possible in order to have the search scalable. From users' point of view, *Coverage* and *SuccessRate* should be as high as possible. In addition, users expect the search time to be short. A small *HopNum* implies the quick response time. As for *QueryHits*, one may think this factor is inappropriate since finding one target is functionally equivalent to finding many. However, returning multiple locations of the objects is essential since the multi-source download has become a standard feature of many successful file-sharing systems [8, 9]. In addition, current file searches will report a hit just if the file names match. Whenever a file is downloaded, there needs confidence that the contents of the file are what is expected and advertised. If only one file is downloaded, a user has no information to verify the content. Downloading multiple targets allows users to use the matched algorithm [19] to select the target with correct content.

4.3. Simulation methodology

We perform 5 different runs for four algorithms: flooding, random walk, expanding ring, and the proposed hybrid search. Since the search performance is relevant to the starting nodes, we randomly choose 1,000 different nodes to perform the search for each run. Totally, 5,000 times of query searches are performed for each algorithm. Every simulation variable is randomly regenerated for each experiment. For random walk algorithm, we choose the number of walkers k (32) to be the same as [10]. The parameter of n used in the proposed hybrid algorithm is chosen empirically to be 2. The TTLs of expanding-ring algorithm starts from one in the increment of one. The termination of flooding, random walk, and hybrid search is also via TTL. For each algorithm, we perform with termination TTLs from 2 to 10. Finally, the query distribution of the distinct 100 objects is random (other distributions give similar results in our simulations). Experimental results are shown in Table 1.

5. Unified analysis criterion: search efficiency

A good search algorithm must be scalable, efficient, and responsive. Looking at various metrics shown in Table 1 can easily make one confused of how to choose the better search algorithm. Therefore, a unified cost function, which takes all factors into consideration, is important and helpful. In this section, we will propose a unified search analysis criterion to evaluate the quality of search algorithms in terms of scalability, efficiency, and responsiveness.

A scalable and efficient search should not generate a huge number of redundant messages and waste the network bandwidth unnecessarily. In addition, efficiency of search means that the query messages generated during the search process should have a high hit rate (finding the target objects). Therefore, we define *Query Efficiency* (QE) as the ratio of *QueryHits* to *MsgPerNode*:

$$Query\ Efficiency = \frac{QueryHits}{\frac{QueryMsg}{NetworkSize}} = \frac{QueryHits}{MsgPerNode}$$

Another important factor of performance is *Search Responsiveness*, evaluating responsiveness and reliability. Responsiveness is the ability of a search to respond quickly to meet the needs of a user. Additionally, reliability is essential to a healthy responsive algorithm, demonstrating the ability to meet the commitments made to users. Therefore, *Search Responsiveness* (SR) measuring the responsiveness and reliability of a search can be defined as:

$$Search\ Responsiveness = \frac{SuccessRate}{HopNum}$$

To capture the characteristics of efficiency, reliability, and responsiveness at the same time, we propose the unified criteria *Search Efficiency* as:

Search Efficiency (SE)

$$= Query\ Efficiency \times Search\ Responsiveness$$

$$= \frac{QueryHits \times SuccessRate}{MsgPerNode \times HopsNum}$$

Note SE describes objectively a relative performance measurement. We treat both the users' satisfaction and network's load equally important. When a search has two times higher *SuccessRate*, its SE will be two times higher. If an algorithm generates more *MsgPerNode*, its SE will be lower.

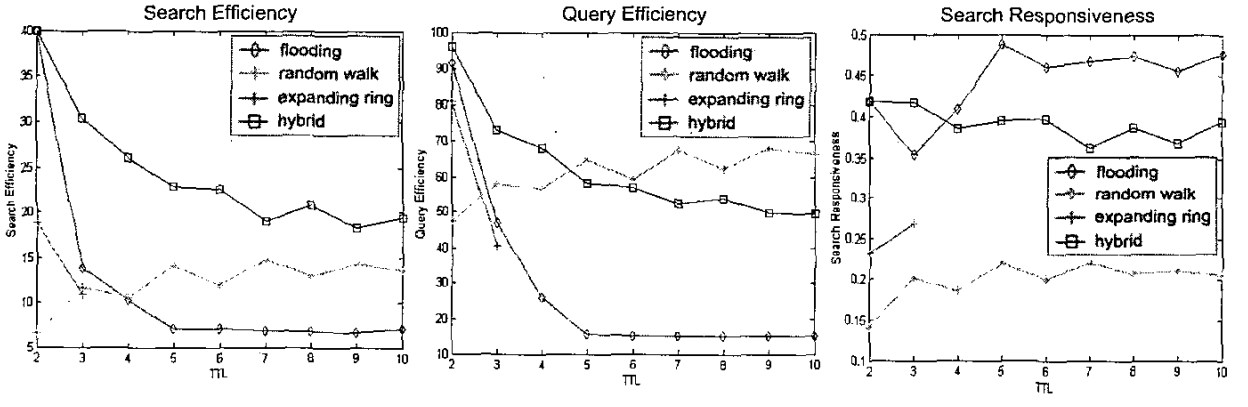

Figure 3. Search Efficiency (a), Query Efficiency (b), and Search Responsiveness (c) comparisons of the four algorithms simulated in Gnutella network in terms different termination (initial) TTLs

6. Performance analysis

Search Efficiency, *Query Efficiency*, and *Search Responsiveness* of various search algorithms are shown in Figure 3, whose results are calculated from the data in Table 1. The x-axes of the figure are the termination TTLs, i.e. the initial TTLs when queries start. The y-axes are the performance results (SE, QE, and SR) when a search finishes, that is, when the TTLs drop to zero. Hence, one can view termination (initial) TTLs in the x-axes as the hop counts or search time of queries since they are the same when a search terminates. From Table 1 and Figure 3(b), we can see QE of flooding algorithm decays dramatically with respect to the search time since *MsgPerNode* grows exponentially and increases at a much higher rate than that of *QueryHits*. Additionally, QE of expanding ring falls below that of flooding. Since expanding ring algorithm stops searching whenever a target is found, *QueryHits* is low and the volume of redundant messages in the local area is high. However, QE of random walk remains almost constant when the search time increases. Although *MsgPerNode* grows linearly as the search time increases, *QueryHits* also increases. Therefore, QE of random walk is consistent regardless of the search time. From the figure, it is noted that QE of random walk is four times better than that of flooding. In the case of the proposed hybrid search,

although QE will decay a little when the search time increases, it remains constant since its long-term dynamic behavior is similar to that of random walk algorithm.

From Figure 3(c) of SR, we can easily see flooding generates the fastest response since it aggressively sends the query messages. It is not surprising to see SR of random walk gives the slowest result. From this figure, it is interesting to know that the speed of flooding is about 2.4 times faster than that of random walk. The proposed hybrid search returns the query results promptly since it searches the network globally when the search time is less than n.

The overall performance (SE), as displayed in Figure 3(a), can be obtained from the multiplication of QE and SR. Although random walk has the best QE performance, the performance of SE is not satisfactory because of the low SR. SE of flooding is high in the short term. It, however, decays dramatically due to the huge number of redundant messages when the search time increases. The proposed hybrid search generates robust SE since it inherits the advantages from both flooding and random walk and performs relatively consistent performance in both QE and SR aspects. By these observations, we conclude that flooding is a responsive algorithm and random walk is an efficient one. Hybrid search, on the other hand, incorporating the strengths of both, is the most robust among the discussed algorithms.

7. File-to-node distribution

In the previous experiments, files (objects) are assigned to nodes randomly regardless of a node's capability. However, in real Gnutella network, it is reasonable to assume the number of shared files in a node depends on the node's capability. For example, the nodes with higher bandwidth have better capability to share more files. One of the most popular P2P systems, KaZaA [22], uses "Participation Level" to encourage users to share more files. With higher values of Participation Level, the user has the privilege to get more and better search results. Therefore, the nodes with higher bandwidth will attempt to share more files in order to get better search results. In order to explore this property and get more convincing results, we introduce the idea of "file-to-node" distribution and perform simulations, whose settings are the same as Section 4 expect this proposed distribution, to obtain the data for analysis.

7.1. Formulation of file-to-node distribution

In the following experiments, we model the bandwidth of a node to be proportional to its link degree. Hence, the number of files v shared by a node i with the link degree of l can be described by the following equation:

$$v_i \propto l_i^{\frac{1}{s}} \qquad s = 1, 2, 3 \ldots\ldots$$

$$= \frac{TotalNumberOfFiles}{\sum_{j=1}^{N} l_j^{\frac{1}{s}}} \times l_i^{\frac{1}{s}} \qquad s = 1, 2, 3\ldots\ldots$$

where N is the number of nodes in the network and the parameter s controls the extent of how the number of shared files v is proportional to the link degree l. Based on this formula, there are 55.7% of nodes to be free riders for s =1, 33.7% free riders for s =2, and 24.9% for s =10. Total number of files is raised to 14,000, compared with the setting in Section 4. Figure 4 plots the CDFs of shared files for s = 1, 2, and 10.

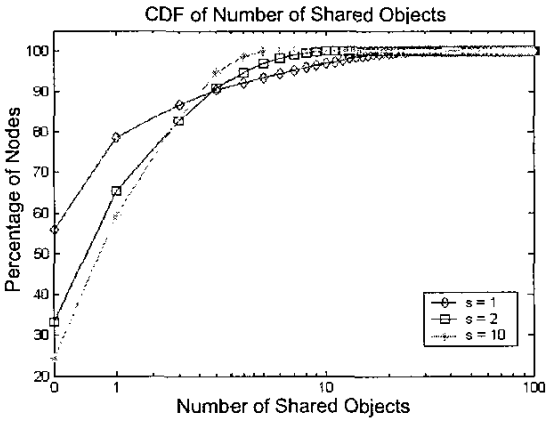

Figure 4. CDF of the number of shared objects with respect to the node percentage for s = 1, 2, and 10

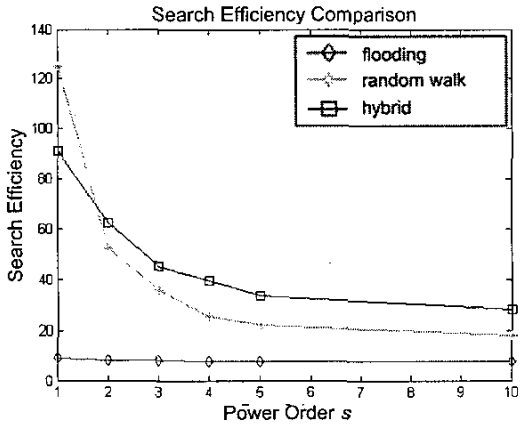

Figure 5. Search Efficiency comparison in different file-to-node distributions (s = 1 to 10) with TTL = 5

7.2. Search performance analysis

When we modify the simulator according to file-to-node distributions, the files are not randomly distributed over the network anymore, and *QueryHits* is no longer linearly proportional to *Coverage* as shown in Table 1. Using only *Coverage* to evaluate the performance of a search algorithm will easily jump to the biased conclusion that a search algorithm should try its best to visit every possible

node in order to generate satisfactory results. This is not true if we look at the simulation results of Figure 5, which shows the simulation results of different file-to-node distributions (s = 1~10) for the three search algorithms. From Figure 5, it is not surprising to note that the search performance of flooding is independent of how the files are distributed. However, it is interesting to learn that random walk shows outstanding search performance when the cluster effect (the number of files distributed based on the link degrees of a node) is evident. To further analyze the effect, we plot the performance metrics in Figure 6 for the case of s = 2. We notice SR of random walk shows about 50% improvement when compared to results of Figure 3(c) in the case of random distribution. Besides, it is noticeably clear that random walk shows excellent QE. This phenomenon suggests that random walk algorithm is likely to visit the high-degree nodes despite the existence of a huge number of free riders. The reason why random walk has great QE may be suggested by Figure 7, in which we analyze the link degree distributions of the visited nodes for the three algorithms when termination TTL is set as 5. Besides, it shows the distributions, normalized by the numbers of visited nodes, not *NetworkSize*. This figure demonstrates the effect that random walk tends to visit nodes with higher degrees, thus having better QE than flooding. Since the proposed hybrid algorithm inherits the dynamic characteristics of random walk in the long term, its QE is shown to be satisfactory although not as good as random walk. Due to the lack of studies about the choice of *s*, we can not conclude which one of hybrid or random walk is better in terms of SE. However, it is true that random walk performs well when file distributions are clustered in high connection peers. This may be a hint for our future work.

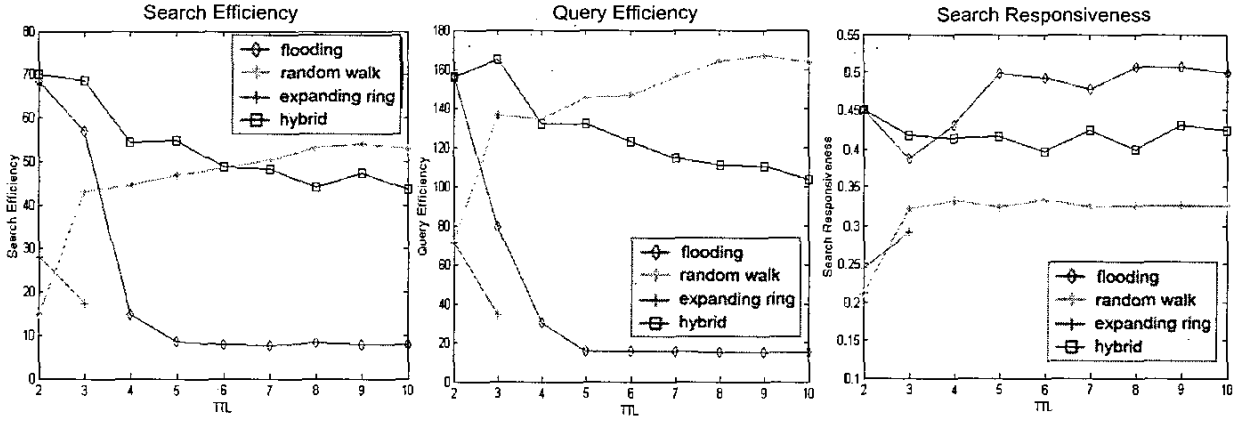

Figure 6. Search Efficiency, Query Efficiency, and Search Responsiveness comparisons of the four algorithms simulated in Gnutella network with the file-to-node distribution of $s = 2$

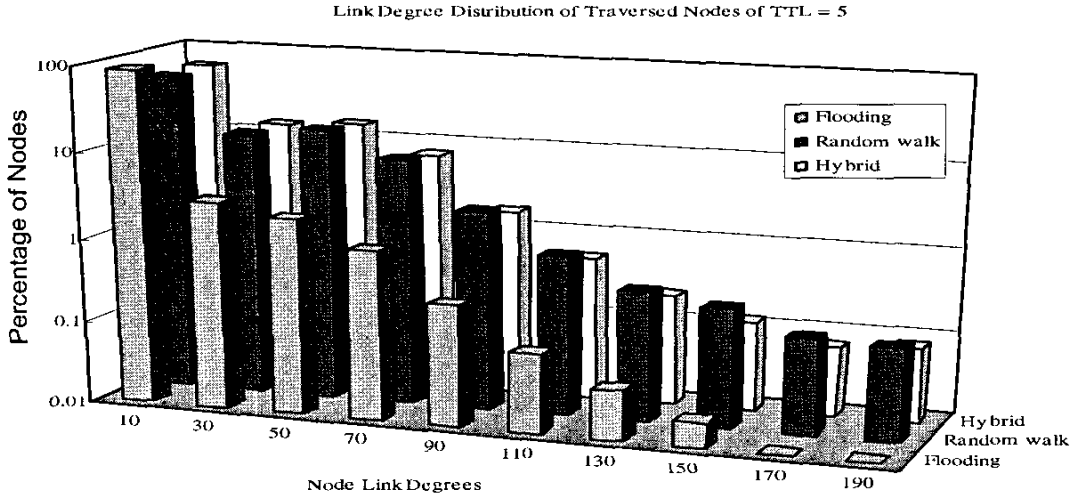

Figure 7. Link degree distribution of traversed nodes for the three algorithms with termination TTL = 5

8. Conclusion

In this paper, we propose a hybrid search algorithm combining advantages of existing search algorithms. Based on the concept of simulated annealing, hybrid search traverses every possible peer in the local as soon as possible, and keep a limited traffic on the global. Therefore, high users' satisfaction can be guaranteed without impacting the whole network, therefore providing a robust and scalable solution.

We also propose a performance evaluation criterion, *Search Efficiency*, to measure scalability, reliability, and responsiveness of search algorithms on a unified basis. From simulation results, our hybrid search outperforms other search algorithms. It inherits the high performance of flooding in the local and keeps the consistent performance of random walk search in the long term. Furthermore, the proposed algorithm is simple in design and implementation, so that it can be easily incorporated into existing unstructured P2P systems.

9. References

[1] Matei Ripeanu, Adriana Iamnitchi and Ian Foster. Mapping the Gnutella Network. IEEE Internet Computing Vol. 6 Issue 1, Jan/Feb 2002, 50-56

[2] K.Sripanidkulchai, The popularity of Gnutella Queries and its Implications on Scalability, white paper, Carnegie Mellon Univ. Pittsburgh, Feb. 2001.

[3] D. Gallagher and R.Wilkerson. Network performance statistics for university of South Carolina. http://eddie.csd.sc.edu, Oct. 2001.

[4] D. Plonka. Uw-madion napster traffic measurement. http://net.doit.wisc.edu/data/Napster, Mar. 2000.

[5] FreeNet website. http://freenet.sourceforge.net.

[6] Gnutella website. http://gnutella.wego.com.

[7] Napster website. http://napster.com.

[8] eDonkey website. http://www.edonkey2000.com.

[9] KaZaA website. http://www.kazaa.com.

[10] Q. Lv, P. Cao, E. Cohen, E.Felten, X. Li and S. Shenker. Search and replication in unstructured peer-to-peer networks. In Proc. 2002 ACM SIGMETRICS, 2002.

[11] Beverly Yang and Hector Garcia-Molina. Improving Search in Peer-to-Peer Networks. ICDCS, 2002.

[12] Lada Adamic, R. Lukose, and B. Huberman. Local Search in Unstructured Networks. Handbook of Graphs and Networks: From the Genome to the Internet, S. Bornholdt and H.G. Schuster (eds.), Wiley-VCH, Berlin, 2002.

[13] Clip2. Gnutella: To the bandwidth barrier and beyond. http://www.clip2.com/gnutella.html, 2000.

[14] Kelly Truelove. Gnutella: Alive, well, and changing fast. http://www.openp2p.com/pub/a/p2p/2001/01/25/truelove0101.html.

[15] Theodore Hong. Performance. In Andy Oram, editor, Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology, chap 14, p 203-241, O'Reilley, 2001.

[16] M. Jovanovic, F.Annexstein, and K. Berman. Scalability Issues in Large Peer-to-Peer Networks: A Case Study of Gnutella. Tech. Report. Univ. of Cincinnati, Lab. For Networks and Applied Graph Theory, 2001.

[17] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in Power-Law Networks. Phys. Rev. E, Vol. 64, pages 46135-46143, 2001.

[18] The gnutella protocol specification v0.6. http://rfc-gnutella.sourceforge.net/draft.txt

[19] Leslie Lamport, Robert Shostak and Marshall Pease, The Byzantine General Problem. ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, Pages 382-401, July 1982.

[20] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. Science, V. 220, No. 4598, page 671~680, 1983.

[21] 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.

Table 1. Performance data for the four discussed algorithms simulated in the Gnutella network

	Initial TTL	2	3	4	5	6	7	8	9	10
Flooding	*Coverage*	277	1896	8606	9969	9975	9875	9975	9975	9975
	MsgPerNode	0.0279	0.3986	3.527	6.4356	6.6298	6.5634	6.6299	6.6298	6.6298
	HopNum	1.95	2.82	2.33	2.2	2.13	2.18	2.23	2.25	2.13
	QueryHits	2.33	15.46	84.56	99.88	99.99	98.99	99.99	100	100
	Success(%)	0.82	1	1	1	1	1	1	1	1
Random Walk	*Coverage*	31	52	79	100	127	149	176	197	225
	MsgPerNode	0.0064	0.0096	0.0128	0.0159	0.0191	0.0222	0.0253	0.0285	0.0316
	HopNum	1.93	2.13	2.73	3	3.46	3.75	4.07	4.42	4.67
	QueryHits	0.24	0.338	0.53	0.656	0.85	1.034	1.17	1.362	1.566
	Success(%)	0.224	0.27	0.404	0.51	0.592	0.65	0.686	0.74	0.822
Expanding Ring	*Coverage*	27.	1946	N/A						
	MsgPerNode	0.0293	0.423	N/A						
	HopNum	2.92	3.69	N/A						
	QueryHits	2.25	16.93	N/A						
	Success(%)	0.82	1	N/A						
Hybrid	*Coverage*	248	412	516	635	748	862	892	996	1098
	MsgPerNode	0.0265	0.0562	0.1185	0.1841	0.2442	0.3127	0.3336	0.4021	0.4548
	HopNum	1.98	2.22	2.29	2.39	2.40	2.52	2.53	2.5	2.68
	QueryHits	2.08	3.042	3.624	4.544	5.486	6.204	6.432	7.212	8.218
	Success(%)	0.794	0.902	0.892	0.932	0.928	0.946	0.954	0.948	0.948