An Effective P2P Search Scheme to Exploit File Sharing Heterogeneity

Chen Wang, *Student Member*, *IEEE*, and Li Xiao, *Member*, *IEEE*

Abstract—Although the original intent of the peer-to-peer (P2P) concept is to treat each participant equally, heterogeneity widely exists in deployed P2P networks. Peers are different from each other in many aspects, such as bandwidth, CPU power, and storage capacity. Some approaches have been proposed to take advantage of the *query forwarding heterogeneity* such that the high bandwidth of powerful nodes can be fully utilized to maximize the system capacity. In this paper, we suggest using the *query answering heterogeneity* to directly improve the search efficiency of P2P networks. In our proposed Differentiated Search (DiffSearch) algorithm, the peers with high query answering capabilities will have higher priority to be queried. Because the query answering capabilities are extremely unbalanced among peers, a high query success rate can be achieved by querying only a small portion of a network. The search traffic is significantly reduced due to the shrunken search space. Our trace analysis and simulation show that the DiffSearch algorithm can save up to 60 percent of search traffic.

Index Terms—Peer-to-peer, search, efficiency, ultrapeers, heterogeneity.

1 Introduction

Peer-to-peer (P2P) networks are booming in today's network community because their fully distributed design makes them good candidates to build fault-tolerant file sharing systems which can balance the load of file storage and transfer. Nevertheless, the basic flooding search approach in unstructured P2P networks raises enormous concerns about the network scalability. Many approaches have been proposed to replace or optimize the flooding search mechanism.

The structured P2P networks (DHT) [17], [20], [24], [29] avoid the flooding search by tightly coupling data or indices of data with sharing peers such that a query can be directly routed to matched peers. However, the index maintenance overhead incurred by highly dynamic node activities and the partial keyword matching problem [4] still need to be addressed before the DHTs can be widely deployed.

Instead of replacing the flooding method, the hierarchical P2P networks such as KaZaA [2] try to reduce the flooding traffic by limiting the search scope within a small area of supernodes. Since the indices of leaf nodes are uploaded into supernodes, the search in KaZaA can achieve good coverage of an entire network by only querying a small number of supernodes.

The designs of both the DHTs and hierarchical P2P networks improve search efficiency at the cost of performing extra index operations such that a query can either be directed to a host previously registered with possible matched indices or flooded within a small searching space which aggregates the indices of an entire network since, in a spontaneously formed P2P network where both peers hosting indices and peers pointed to by indices are unreliable and may join and leave the network arbitrarily, the cost of frequent index updates cannot be ignored. Accordingly, how to balance the costs between searching and indexing becomes a critical design issue in such a dynamic environment. An intriguing question is: "Can we improve the search efficiency of P2P networks with zero or minimal indexing cost?"

In this paper, we propose the Differentiated Search (DiffSearch) algorithm to improve the search efficiency of unstructured P2P networks by giving higher querying priority to peers with high querying reply capabilities. Our experiment shows that the DiffSearch algorithm can improve the search efficiency without resorting to index operations.

Our proposal is based on the observation [3], [12], [21] that query reply capabilities are extremely unbalanced among peers: Seven percent of peers in the Gnutella network share more files than all of those other peers can offer and 47 percent of queries are responded to by the top 1 percent of peers. The remarkable heterogeneity of peers' reply capabilities intrigues us to revisit the nature of P2P networks. Rather than all peers actively participating in file sharing, only a small portion of volunteering peers provide the majority of the service in P2P networks. The observation on file sharing heterogeneity shows that some peers are more willing to share files than others. For example, some new musicians regard P2P networks as free platforms to distribute their productions and therefore build their reputations. P2P networks, compared with traditional client server architectures such as Web systems, provide ideal infrastructures for those content publishers because of several reasons: 1) It does not require publicized DNS names with fixed IP addresses and stable online time. 2) The bandwidth demand on a service provider in P2P networks is not as critical as that on a server since the load of file transfer can be balanced to other service providers

• *The authors are with the Department of Computer Science and Engineering, 3115 Engineering Building, Michigan State University, East Lansing, MI 48824. E-mail: {wangchen, lxiao}@cse.msu.edu.*

Manuscript received 2 June 2005; revised 31 Dec. 2005; accepted 28 Feb. 2006; published online 27 Dec. 2006.
Recommended for acceptance by C. Shahabi.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0274-0605.

or leaf nodes with replicated data. 3) Service providers in P2P networks can retain better anonymity than well-known servers. In summary, P2P networks create a low cost and easily accessible publishing environment. Our DiffSearch algorithm further improves this environment by promoting content-rich publishers to ultrapeers and making them more visible for other visitors.

The heterogeneity of file sharing motivates us to redraw the picture of P2P networks: A small portion of volunteers provides file sharing service to a large number of visiting peers. In such a model, the basic flooding approach is like looking for a needle in a bale of hay since queries are forwarded by a large number of visiting peers without any contributions to search results. Instead of flooding, we propose the DiffSearch algorithm, which gives service providers higher priorities to be queried. Under the DiffSearch algorithm, a small number of service providers (ultrapeers) with high query answering capability form a service provider overlay, and the rest of the visiting peers connect to the service provider overlay as leaf nodes. Each query will be sent to the service provider overlay first. If the query fails in the service provider overlay, it will resort to the entire P2P network.

In the DiffSearch algorithm, we make the basic assumption that peers are honest and cooperative. Malicious peers, however, do exist in deployed P2P networks. A malicious peer may corrupt a query by dropping messages or responding with false replies. It is possible that a malicious peer will pose as an ultrapeer in the DiffSearch algorithm. The negative effect is that the ultrapeer overlay is over-expanded, which incurs extra searching traffic. How to address "dishonest" peers has been intensively studied [9], [10] and is out of the scope of this paper.

This paper makes four contributions to improve the performance of unstructured P2P networks:

- We analyze the query/reply traces collected from the Gnutella network to investigate the phenomena of file sharing heterogeneity and propose utilizing the heterogeneity to improve the search efficiency.
- The DiffSearch algorithm proposed in this paper is a fully distributed approach. The two-tier hierarchical structure can be evolved from an ad hoc P2P network by the *localized* algorithm running on each peer without global knowledge.
- A caching and redirecting mechanism is proposed to balance the query load in ultrapeers according to their processing capacities.
- A buddy-monitoring solution is suggested to provide a stable indexing service in a highly dynamic environment.

The remainder of the paper is organized as follows: Section 2 presents an overview of the DiffSearch algorithm. Section 3 investigates the file sharing heterogeneity of P2P networks in detail. We illustrate how the two-tier hierarchical structure can be efficiently evolved by the DiffSearch algorithm in Section 4. The performance improvement of the algorithm is evaluated in Section 5. Section 6 describes the related work, and conclusions are drawn in Section 7.

2 Overview of the DiffSearch Algorithm

Current hierarchical designs select the ultrapeers by emphasizing their computing capabilities such as bandwidth, CPU power, and memory spaces. In this paper, we argue that the content capacity, i.e., the number of files shared in a node, is also an important factor to decide if a hosting peer is an ultrapeer. After aggregating peers sharing a large number of files into the ultrapeer overlay, good search performance can be achieved by only flooding a query within a small searching space.

Based on the ultrapeer overlay which contains the main portion of shared files, we propose the DiffSearch algorithm. In the DiffSearch algorithm, a query consists of two round searches. In the first round search, the query is only sent to the ultrapeer overlay. If the first round search fails in the ultrapeer overlay, the second round search will be evoked to query the entire network. Here, we assume the entire P2P network is connected, which is consistent with the observation in [21]. The prerequisite of the DiffSearch algorithm is that the ultrapeer overlay consisting of content-rich peers is well formed in a P2P network. We show that the DiffSearch algorithm can be self-utilized to shape an ad hoc P2P network into a two-tier hierarchical structure which clearly separates the content-rich ultrapeer overlay from other peers. We illustrate the ultrapeer overlay construction by an example as follows: If a DiffSearch query fails in the first round search, but succeeds in the second round, this implies two possible cases:

- The queried object is not shared by any ultrapeers such that it can only be found by querying all the network.
- The queried object is shared by ultrapeers and some of the replies in the second round search are responses from the ultrapeers, which implies that the failure of the first round search is caused by the incompleteness of the two-tier hierarchical structure: either because the leaf node is not connected to the ultrapeer overlay or the ultrapeer overlay is partitioned into multiple disjointed clusters. Otherwise, the query should have succeeded in the first round search in the ultrapeer overlay.

In the second case, the incompleteness of the two-tier structure is detected and can be repaired by adding new connections between the querying peer and responding ultrapeers. Since any first round search failure due to the incompleteness of the hierarchical structure can be self-repaired by the DiffSearch algorithm, the two-tier structure can be incrementally evolved from a random status through frequent search behaviors initiated from a large number of querying peers. Our experiment shows that the algorithm converges very fast and the two-tier hierarchical structure can be evolved after each peer sends out about 20 queries on average.

A concern of the DiffSearch algorithm is that the load of file transfer will be unbalanced between ultrapeers and leaf nodes. Since ultrapeers have higher priorities to be queried, leaf nodes may have no chance to contribute their uploading bandwidth if they share the same files as ultrapeers. To solve the problem, we enhance the DiffSearch algorithm by uploading indices of leaf nodes to ultrapeers

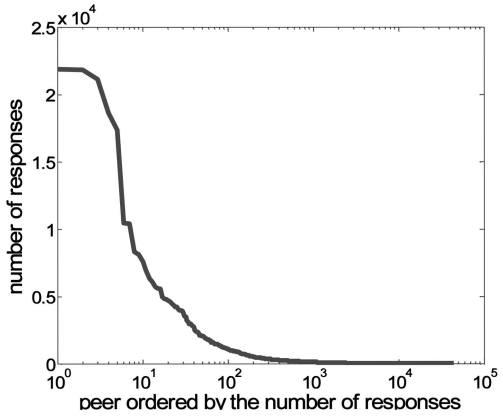


Fig. 1. Response distribution.

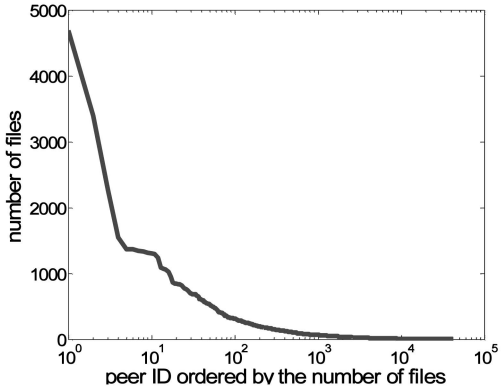


Fig. 2. File distribution.

such that the indices of all the shared files are searchable in the first round search. Since the ultrapeer overlay shares the majority of files, the cost of index operations is small, which is shown in our experiment.

We start our investigation on the file sharing heterogeneity in the next section, which is followed by a detailed discussion on the DiffSearch algorithm.

3 Heterogeneity of File Sharing

While the primary intent of the P2P concept is to blur the border between service providers and consumers and treat each participant equally, the heterogeneity is an inherent characteristic of P2P systems. Peers vary in many aspects, such as network bandwidth, CPU power, and storage capacity. It is a natural choice that this heterogeneity is taken into account when designing P2P systems, such as Gia [4] and the flow control and topology adaptation algorithm in [14]. Those algorithms try to balance the querying forwarding load among peers according to their bandwidth resources to maximize the system capacity. In contrast to the heterogeneity of the query process capability, the heterogeneity of query reply capability, observed in several studies [3], [21], has not been emphasized in P2P system design yet.

To investigate the heterogeneity of peers' querying answering capabilities in detail, we implemented a Gnutella trace collector based on the Limewire servant. By hooking the log functionality to the Limewire servant, all of the queries and responses passing by the trace collector are recorded. Aiming at collecting enough data within a short period of time, we intentionally set the collector as an ultrapeer with 100 neighbors. After one day of online time on 28 December 2003, 20 million queries were collected in a query trace file of 1.6 Gbytes and 5 million responses were collected in a response trace file of 670 Mbytes. Because each query and its responses share the same GUID (Global unique ID), the query can be matched with its corresponding responses, even if they are logged in two separated files. According to the RFC 1918 [18], the responses from IP addresses prefixed by 10/8, 172.16/12, and 192.168/16 are filtered from the response trace file because those IP addresses are allocated to private networks and widely used in Network Address Translation (NAT). The responses coming from the IP addresses within those ranges cannot be guaranteed to be from the same host even if they have the same private IP address. Based on the analysis of the refined traces, we investigate the response distribution among peers and metrics of selecting ultrapeers.

3.1 Response Distribution

The response distribution based on the response trace is illustrated in Fig. 1. The peers are ranked by the descending order of the number of responses. Fig. 1 shows a similar result as in [3], i.e., the top 1 percent of peers answers the main portion of queries. Such an observation supports the conjecture that, if we could route all the queries to those top peers first, close to 90 percent of query traffic would have been saved.

3.2 Number of Shared Files

Peers vary in their query reply capabilities because of their file sharing heterogeneity. The trace analysis demonstrated in Fig. 2 shows that very few peers share a large number of files. Because query answering is the process of matching keywords with all the shared file names, we conjecture that, as the number of shared files increases, the probability of successful matching should become higher. However, the investigation in [3] shows that there is a weak correlation between query responses and the number of shared files. Our explanation is that some useless files make no contribution to the query answering, i.e., some files are never used to answer the queries. If we only count the files which have been used to answer the queries, the number of responses does show a strong correlation with the number of shared files. To distinguish those files from useless files, we define the files which have been used to answer the queries as ***Effective Files***. The correlation between the number of responses and the number of effective files is illustrated in Fig. 3. As we can see, the peers sharing more effective files have a greater tendency to answer queries. This correlation makes the number of effective files shared by a peer a good criterion to determine if the peer should be selected as an ultrapeer.

The trace analysis in Fig. 2 also shows that a steep declining curve exists in the file distribution, which makes it simple to decide the threshold of selecting ultrapeers. By setting a threshold of 100 effective files, the top 2 percent of peers are selected from 10,000 peers to form the ultrapeer overlay. Fig. 4 shows that more than 80 percent of queries can be replied to from the ultrapeer overlay, which is contrasted by the fact that less than 20 percent of queries can be answered by a cluster consisting of the same number of peers picked randomly. Fig. 5 shows that the success

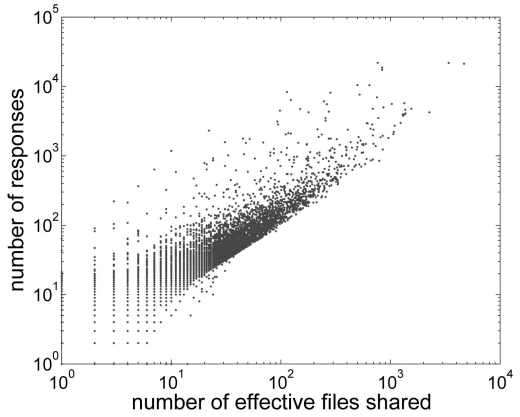


Fig. 3. Correlation between query response and number of files.

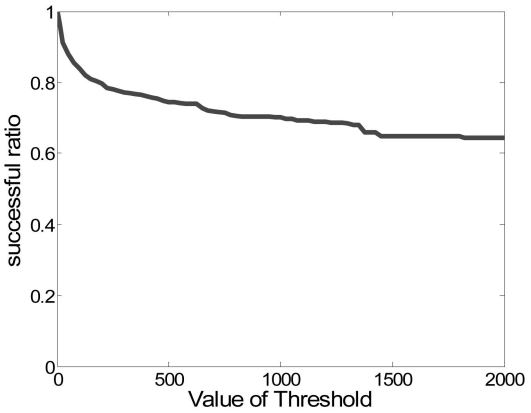


Fig. 5. Success ratio under different threshold values.

ratio in the ultrapeer overlay decreases from 100 percent to 60 percent when varying the threshold from 0 to 2,000. The trade-off here is between the overlay success ratio and the size of searching space of the ultrapeer overlay.

Fig. 6 shows that the size of the ultrapeer overlay does not change significantly when the threshold is decreased from 300 to 50. This is because only a small portion of peers share a large number of files and the rest of peers share a few or zero files. As a result, it will not include too many peers into the ultrapeer overlay when lowering the threshold from 300 to 50. This observation illustrates that the size of the ultrapeer overlay is not sensitive to the threshold value when the value varies within a certain range. Consequently, we can choose the threshold value from a wide range to achieve the same optimal performance of the DiffSearch algorithm in terms of the number of forwarding hops incurred by each query.

4 DiffSearch Algorithm

Using the metric of *effective files* discussed above, the ultrapeers are self-aware by simply counting the number of shared files which have been visited before. As long as the number of shared effective files reaches the threshold, a peer can promote itself as an ultrapeer. In our DiffSearch algorithm, those peers should form an ultrapeer overlay and have higher priority to be queried. The challenge is how to find the ultrapeers and connect them into an overlay efficiently.

The basic objectives of our topology creation algorithm are: 1) Each leaf node has at least one ultrapeer as a neighbor and 2) the ultrapeer overlay is a connected graph.

4.1 Finding Ultrapeers

For the first objective, the peer isolated from ultrapeers needs to find an ultrapeer as a neighbor. There are two basic approaches to finding an ultrapeer. The first one is a *passive approach* in which the isolated peer sends out an "ultrapeer search" message in the way of flooding and the ultrapeers respond back with their IP addresses. The second one is an *active approach* in which ultrapeers periodically publicize themselves to the P2P network and the isolated peers can find ultrapeers by overhearing the publicizing messages.

In the first approach, both the "ultrapeer search" message and response messages will cause tremendous overhead if all the isolated peers repeat the same process. Furthermore, the isolated peers will be overwhelmed by the response messages sent back from thousands of ultrapeers. In the second approach, the periodically publicized messages will also cause remarkable overhead, and the peer being publicized may be overwhelmed by the connection requests from a huge number of isolated peers, which will result in some heavily loaded ultrapeers.

To design an efficient topology creation algorithm, we suggest the following principles: 1) To minimize the

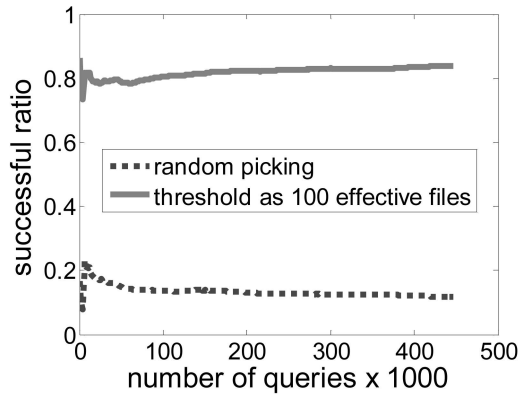


Fig. 4. Success ratio in ultrapeer overlay.

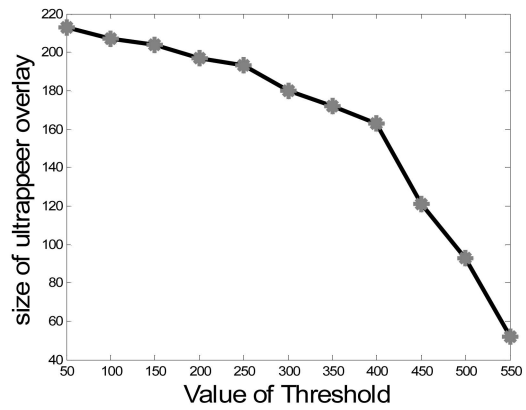


Fig. 6. Size of the ultrapeer overlay under different threshold values.

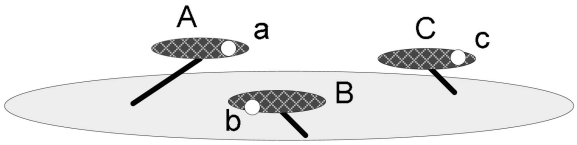


Fig. 7. A hierarchical P2P network.

overhead, the topology creation message should be hitchhiked on other query/response messages, 2) the extreme asymmetric scenario, i.e., thousands of peers responding to or connecting to a single peer should be avoided, and 3) the newly created connections should be balanced among ultrapeers. Based on these principles, we incorporate topology adjustment operations into the query/reply protocols of a current P2P network. One bit of data is appended to the reply message to indicate if the respondent is an ultrapeer. All of the replies received by isolated peers will be checked and the IP addresses will be extracted from the message sent from ultrapeers. Because ultrapeers have high query answering capabilities, the isolated peers can find ultrapeers with high probability after sending out several queries. In addition, no extra messages are required for the isolated peers to find the ultrapeers and the overhead is minimized by using only one bit of data in query replies. Furthermore, the isolated peer will not be overwhelmed since the number of replies is limited.

4.2 Evolve an Ultrapeer Overlay

To guarantee that each peer in the ultrapeer overlay can be reached by the first round search in DiffSearch, all the peers in the ultrapeer overlay should form a connected topology. The basic approach is to detect all the separated clusters consisting of ultrapeers and connect them with each other. Again, in a fully decentralized environment, a careful design is necessary to avoid too much overhead caused by the cluster detection.

All of the separated clusters can be found if we flood the cluster detection message to the entire P2P network. However, this flooding may be initiated by multiple peers due to the lack of a synchronization mechanism. Since each peer is uncertain if other peers will start the cluster detection, all the peers must repeat the same operations. Furthermore, the cluster detection messages must be periodically flooded since the connected overlay will be disjointed by the dynamic movement of the P2P network.

The P2P network cannot tolerate the tremendous overhead caused by the detection messages periodically flooded from all the peers. A minor improvement is that we can designate the bootstrapping server to be responsible for the cluster detection and reduce possible redundant operations. This centralized solution, however, may overload the bootstrapping server.

To minimize the overhead of cluster detection, we follow the same design principles as finding ultrapeers, i.e., the peers' search operations will be reused by cluster detection. In our design, the cluster detection data hitchhikes to DiffSearch query messages. We use an example in Fig. 7 to illustrate the cluster detection, where the ultrapeer overlay is divided into three clusters. Suppose peer a fails to search keyword k in cluster A during the first round search of DiffSearch, which is caused by two possible reasons: 1) The keyword k is not shared by ultrapeers and 2) the file k is shared by ultrapeers, but they are located in separated clusters B or C. For any case, peer a will initiate the second round search to the whole network. If the second round search is received by ultrapeers b and c in possession of file k, both b and c can make the conclusion that they are disconnected from the cluster A. Otherwise, the first round search would have received a successful response from peers b and c, and it is unnecessary for peer a to initiate the second round query. As long as peers b and c detect the cluster disconnection, the notification messages will be carried back in the query replies made by peers b and c. Based on the notification and the IP addresses in the query replies, peer a will set up new connections with peers b and c such that all the three disjointed clusters will be connected together. If the source peer a itself is not an ultrapeer, an ultrapeer from its neighbors will be picked up to set connections with peers b and c. If peer a cannot find any ultrapeer in its neighbor list, peer a itself will be connected to peers b and c. In this case, peers b and c cannot be reached in the first round search just because peer a is isolated from the ultrapeer overlay.

The approach proposed above works in an incremental way. As the number of queries increases, the ultrapeer overlay will gradually evolve into a connected graph. Since the ultrapeers have a high probability of answering queries, the disjointed clusters can be detected with high probability. The simulation shows that the algorithm to connect disjointed clusters converges very fast.

One problem of the above algorithm is that clusters B and C may be connected with each other. In this case, one connection from a to either b or c is enough to connect all the clusters. We hope the number of newly added connections can be minimized so that the duplicated query messages in the ultrapeers overlay can be reduced. To eliminate the redundant connections, peer a has to judge if peers b and c are in a connected graph. Again, the testing approach for peer b to flood a message to peer c is not encouraged. An alternative method is that peer a only makes one connection with one of the two responding peers. This conservative approach will slow down the convergence speed but reduce redundant connections.

4.3 Maintaining the Hierarchical Structure

From the above analysis, we show that all of the overlay constructing operations can be finished by only using the search messages. If peer a is disconnected from the ultrapeer overlay, it will initiate the second round search, which may be replied to by some ultrapeers with their IP addresses carried back in the query reply messages. In such a case, the disjointed clusters are detected and the notification will be sent back to querying peer a. To hitchhike the overlay construction to the search messages, three bits need to be appended to the original query and reply messages. One bit is used in the query message to show whether the query is in the first round or second round. Two bits are used in the reply message to show whether the reply is from an ultrapeer and to which round search the reply is responding.

Due to the highly dynamic nature of P2P networks, the ultrapeer overlay may be broken by peers' ungraceful departures. However, as illustrated above, the incompleteness of the two-tier structure can be detected and repaired by ongoing queries of the DiffSearch. We show in our experiment that the DiffSearch approach is a self-maintained

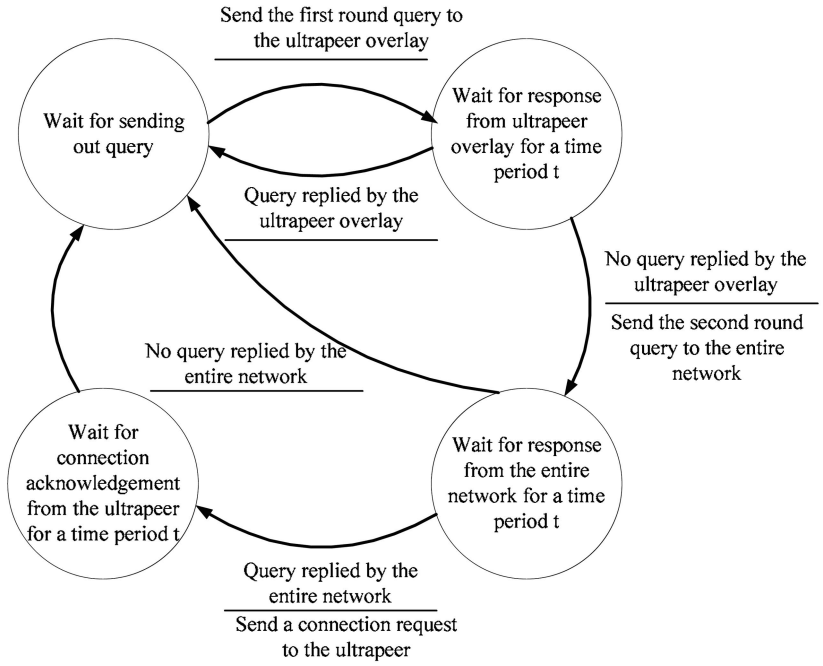


Fig. 8. Two round query operation of an individual peer.

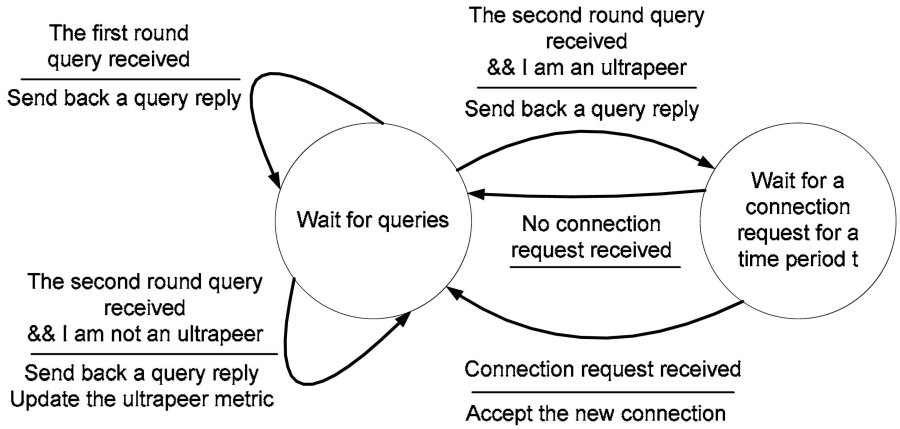


Fig. 9. Query reply operation of an individual peer.

algorithm which can keep the network in a good shape even in an unstable environment.

4.4 Fully Distributed Operations

In this section, we show that all of the operations above, including the discovery of ultrapeers, the formation of ultrapeer overlay, and the maintenance of the two-tier hierarchy structure, can be implemented in a fully distributed fashion. In our design, each peer behaves as a normal Gnutella client, which only has the basic functions of query and query response, and the operations of evolving and maintaining the two-tier hierarchy structure are hitchhiked to the query/response messages.

The state transition of a peer's query operation is illustrated in Fig. 8. When a peer initiates a query, it will send the first round query to the ultrapeer overlay. If the first round search fails, it will send the second round search to the entire network. After it receives query replies to the second round search, the peer will pick up an ultrapeer from the reply messages and request a new connection to that ultrapeer. The entire operations above are completed locally by the querying peer without global information. The two-tier hierarchical structure can be shaped and maintained well by the search operations when all of the participating peers perform the two round query searches.

The state transition of a peer's query reply operation is illustrated in Fig. 9. If the search message is the first round, the responding peer will simply send back the query reply. Otherwise, it will check its own ultrapeer's status. If the responding peer is not an ultrapeer, it will send back a query reply and update the ultrapeer metric, i.e., the number of effective files. If the responding peer is an ultrapeer, it will send back a query reply and wait for a period for a new incoming connection.

5 Performance Evaluation

We implemented a trace-driven P2P network simulator in the message level to evaluate the DiffSearch algorithm. The metrics we evaluated are listed below:

Average Network Traffic: For each query, the network traffic is the total number of hops that the query is forwarded. The Average Network Traffic is defined as the sum of the network traffic of all the queries divided by the number of queries.

Query Success Rate: Query success rate is the percentage of queries which are successfully responded to by at least one query reply.

Overlay Query Success Rate: The Overlay Query Success Rate is the percentage of DiffSearch queries which can succeed in the first round search.

Response Time: We define Response Time as the minimum number of hops for the query reply to be forwarded back to the querying peer.

There are two possible approaches to implementing the P2P simulator, i.e., the probabilistic mode and the file sharing mode. In the probabilistic mode, each peer responds to a query according to a binary random distribution, i.e., a peer uses a random process with a fixed probability to decide if a query can be answered or not. Since this paper focuses on utilizing the heterogeneity of query answering capabilities to improve the search efficiency, this highly abstracted mode is not suitable for measuring the performance. Otherwise, the whole simulation will be suspected to "stacking the deck" since the query answering capabilities—the probabilities of successful query replies—are artificially assigned to peers.

To precisely reflect the impact of file sharing on the search behavior, the actual files are shared by each peer in our simulation. When a peer receives a query, the query keywords will be matched against the shared file list. Obviously, only the filename is useful in the keyword matching process and we only keep the filenames to represent the whole files. From this point, a shared file list can be described by an array of text strings.

The file sharing mode simulator is more similar to real P2P applications than the probabilistic mode. However, the file distribution and the sequence of query keywords will have a significant impact on the file sharing mode. To achieve a convincing simulation result, all of the simulation configuration parameters are based on the traces collected from the Gnutella network.

5.1 Configuration of the Simulation

Network Topology: A modified crawler is used to take a snapshot of the Gnutella network. By repetitively sending out the Ping messages with $\mathrm{TTL} = 2$, the crawler can collect the topology information together with the number of files shared by each peer. The 10,000 peers are picked up from the snapshot to form a connected network topology of our simulation.

File Distribution: The files are assigned to each peer according to the trace of query replies. All of the files responding from the same IP addresses in the trace of query replies are assigned to the same peer in the simulation. By using the threshold of 100 effective files, 2 percent of the peers are selected out from the 10,000 peers as ultrapeers.

Query Distribution: The sequence of query keywords is extracted from the query trace file. The queries which have no correspondent replies in the query reply trace are filtered such that any query is always solvable if all the peers in simulation are queried. Through this method, we set up a baseline to measure the query success rate. The filtered sequence of query keywords will be used by the querying peers randomly picked up from the simulated P2P network. We use $\mathrm{TTL} = 7$ when broadcasting a query in the simulated network.

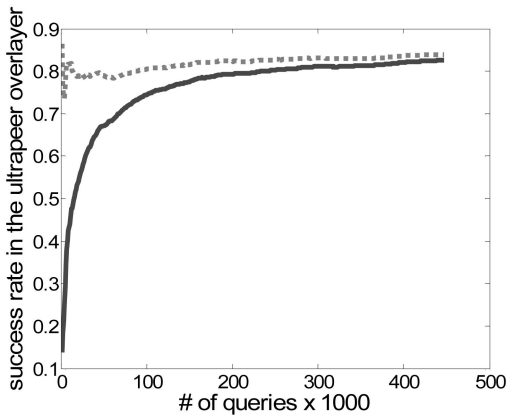


Fig. 10. Overlay success rate.

Peer Capacity: We use the maximum number of connections that a peer can setup to model the peer's capacity. Two configurations are used in our simulation. In the first configuration, each peer can have a maximum of 100 connections. In the second configuration, we use the power law distribution to assign a maximum number of connections to each peer.

5.2 Performance Evaluation of DiffSearch

5.2.1 Convergence Speed

Since the DiffSearch algorithm builds up the two-tier hierarchical structure in an incremental way, the primary concern is how long it will take to create the structure. We use the query success rate in the ultrapeer overlay to measure the convergence speed of the DiffSearch algorithm. First, we intentionally send the queries to a set which consists of all the ultrapeers. The testing result showed that about 80 percent of queries can be answered by the set of ultrapeers, which means the DiffSearch algorithm can achieve 80 percent query success rate in the ultrapeer overlay if the two-tier hierarchical structure evolved into a perfect shape. In the following experiment, we construct the network topology based on the snapshot taken by our Gnutella crawler. Since the Pong messages collected by the crawler also contain the number of files shared by each peer, we approximately assign each node the same number of files as that collected by the crawler. The filenames are extracted from query reply traces recorded by our trace collector and all the files shared by the same peer are assigned to the same node in our simulation. The queries are sent out from randomly selected nodes in the same sequence as that collected in the query trace. The simulation result is shown in Fig. 10. As we can see, with the increase in the number of queries, the query success rate rises to 80 percent, which is close to the rate under the ideal condition that the ultrapeer overlay is perfectly formed. The figure also shows that, after each peer sends out about 20 queries, the ultrapeer overlay can be evolved to an ideal situation. The simulation above demonstrates that DiffSearch is an effective algorithm which can efficiently construct the two-tier hierarchical structure from a completely random starting point. The algorithm converges very fast because of the file sharing

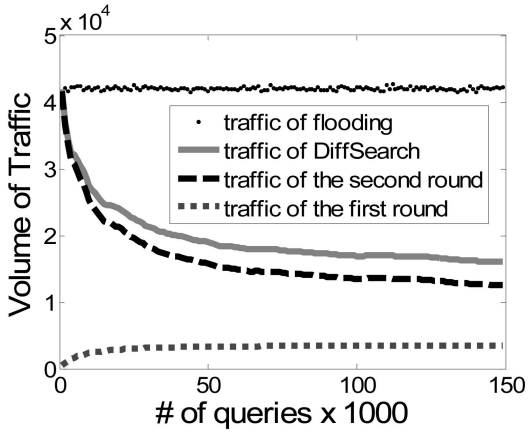


Fig. 11. Average traffic of DiffSearch.

heterogeneity in P2P networks. As we have shown in the previous trace analysis, the query reply can be made by ultrapeers with high probability such that the ultrapeers can be easily found by the DiffSearch algorithm and the ultrapeer overlay can be quickly formed.

5.2.2 Performance Improvement

To measure the search efficiency improvement of the DiffSearch algorithm, we use the metrics of the average network traffic, average query response time, and query success rate. The average network traffic per query is used to measure the query cost. The average query response time and the query success rate are used to measure the user perceived query quality, i.e., how long a user has to wait before a query result can be sent back to a querying node and how likely it is that a query can be solved.

We compare the search performance of the DiffSearch algorithm with the flooding-based approach. The simulation results are shown in Fig. 11, Fig. 12, and Fig. 13. The comparison in Fig. 11 shows that, with the help of the DiffSearch algorithm, the average volume of traffic per query decreases from 42,000 hops to 16,000 hops. More than 60 percent traffic can be saved by utilizing the ultrapeer overlay. The DiffSearch algorithm can significantly reduce the query cost because 80 percent of the queries can be solved in the ultrapeer overlay, which has a much smaller searching space than the whole network. Fig. 11 also shows that the traffic incurred by the second round search consists

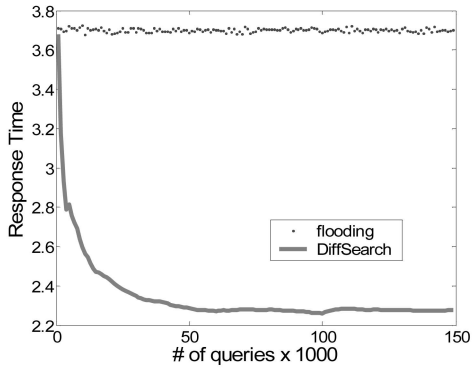


Fig. 12. Average response time of DiffSearch.

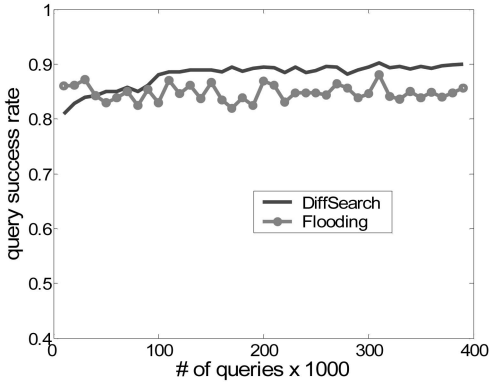


Fig. 13. Query success rate comparison.

of more than 80 percent of the cost, even though only 20 percent of the queries need to resort to the second round search. Since the second round search contributes the main portion of the overall cost, it is extremely important to improve the success rate in the ultrapeer overlay such that fewer queries need to resort to the costly second round search. We address this issue in Section 5.2.3.

The average response time comparison is shown in Fig. 12. The simulation result shows that the average response time decreases from 3.7 hops to 2.3 hops. The overall user perceived response time can be reduced by 40 percent. The DiffSearch algorithm can reduce the query response time because 80 percent of the queries can be solved in the ultrapeer overlay, which has smaller diameter than the entire P2P network, such that most of the queries can be responded to within a shorter distance.

Fig. 13 shows that the DiffSearch algorithm can also improve the query success rate, but to some limited extent. The DiffSearch algorithm can improve the query success rate because the ultrapeer overlay creates shortcuts between ultrapeers and leaf nodes such that more ultrapeers are within searching distance. However, the improvement is not so significant due to the small world property of P2P networks. The original flooding-based search approach with a high time to live (TTL) value already has good coverage in the searching space.

5.2.3 Load Balance

The DiffSearch algorithm causes serious load balance problems, which are demonstrated in two aspects:

1. the query loads are unbalanced and some ultrapeers become hotspots and
2. the file transfer loads are unbalanced since most of queries are solved by ultrapeers who become the main content providers and the leaf nodes make few contributions to file transfer.

We use the number of received messages per query to measure the query load of each peer. In a P2P network with tree-like topology, a query will be forwarded to each node at most once. Nevertheless, the existing P2P networks are randomly formed by unsupervised peers, which creates link circles in the overlay topologies. Due to a large number of link circles existing in a P2P network overlay, a query may be forwarded to the same peer multiple times along different incoming connections. In such a case, the greater the number of neighbors a peer is connected to, the greater

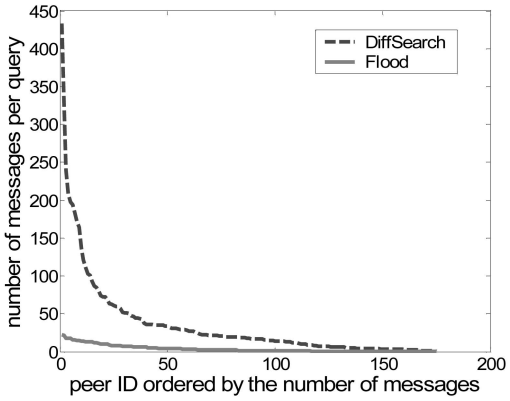


Fig. 14. Unbalanced load of ultrapeers.

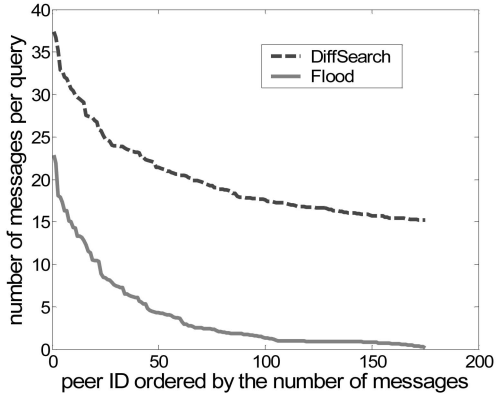


Fig. 15. Load balance of ultrapeers.

the number of messages the peer may receive for the same query. In two-tier hierarchical P2P networks, ultrapeers become central hubs connected by a large number of leaf nodes. It is likely that ultrapeers may be overloaded by incoming query messages. To measure the query load of ultrapeers, we recorded the total number of messages received by each ultrapeer. The average number of received messages per query equals the total number of messages received divided by the total number of queries. To investigate the impact of the hierarchical topology on the query load of ultrapeers, we first use the flooding-based search in the "flat" topology and calculate the average number of received messages per query in each ultrapeer. After that, the same experiment is repeated using the DiffSearch algorithm in the hierarchical topology.

Fig. 14 shows the ultrapeers' load comparison. Two results can be observed from the comparison: 1) The average query load of ultrapeers in a hierarchical structure is much heavier than that in a flooding-based P2P network. 2) The load among ultrapeers is extremely unbalanced, varying from a maximum of 400 messages per query to a minimum of three messages per query.

The ultrapeers are heavily loaded by query messages because of the asymmetry between ultrapeers and leaf nodes. The ratio between ultrapeers and leaf nodes is 2 to 100, which means one ultrapeer is connected by 50 leaf nodes on average. The large number of connections between ultrapeers and leaf nodes causes a serious problem in the second round search of the DiffSearch algorithm. Since the second round search floods queries to the whole network, the large number of connections between ultrapeers and leaf nodes results in duplicated queries being forwarded to the same ultrapeer multiple times. To alleviate the impact of connections between ultrapeers and leaf nodes on the query load, we distinguish the connections by marking the newly added connections which are created by the DiffSearch algorithm to form the ultrapeer overlay. The new connections are only used in the first round search and ignored by the second round search. The consequence is that the ultrapeers work as ordinary nodes in the second round search since the newly added connections are virtually removed. Fig. 15 shows that, after we apply the connection distinguishing solution to the DiffSearch algorithm, the query load of ultrapeers is significantly reduced.

The other unsolved problem is that the query load is extremely unbalanced among ultrapeers, which can be addressed by the ***Caching and Redirecting*** solution we propose below. In a P2P network enhanced by caching and redirecting, a peer's current load status is carried by query reply messages and distributed to the network. Each ultrapeer overhears query reply messages and caches the IP addresses of other ultrapeers which are less loaded than itself. When a fully loaded ultrapeer cannot accommodate more incoming connection requests, it will redirect the requests to other ultrapeers in the caching list. The same operation will be recursively applied until a capable candidate is found. The caching and redirecting algorithm is described in Algorithm 1. To represent the query process capability of each peer in our simulation, we manually set a threshold as the maximum number of incoming connections for each peer. Fig. 15 also shows that the query load is balanced among ultrapeers after the caching and redirecting solution is applied.

Algorithm 1: Caching and redirecting load balance algorithm

While true
  Wait for message m;
  If m is a query reply
    If $load\_of\_responder < local\_load$
      Cache the responder's IP address in the list L;
    End;
  Else if m is a connection request
    If $local\_load < max\_load$
      Accept the new connection;
      $local\_load = local\_load + 1$;
    Else
      Forward the request connection to other ultrapeers in the cache list L;
    End;
  End;
End;

The second issue of load balance is that ultrapeers may be overloaded by file downloading. Since leaf nodes are queried only when the DiffSearch fails in the ultrapeer overlay, they have no chances to contribute their upload bandwidth if they share the same files as the ultrapeers. We

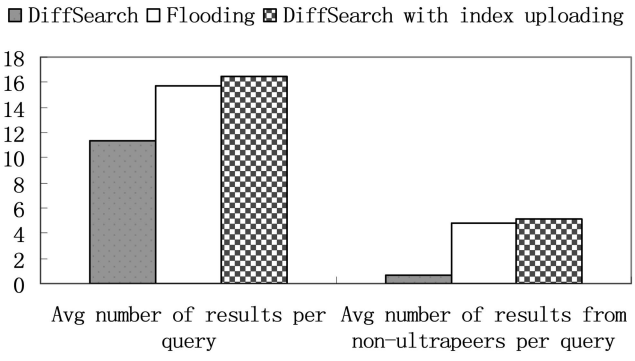


Fig. 16. Comparison of the number of results.

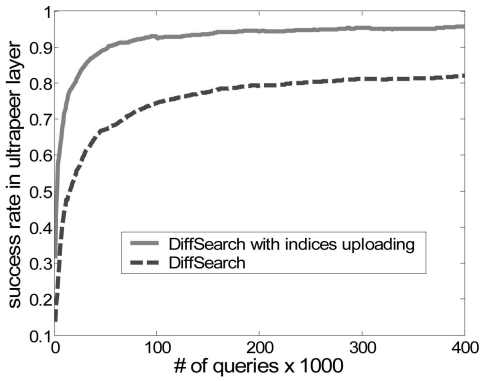


Fig. 17. Comparison of the success rate of the ultrapeer overlay.

use the average number of responses per query and the average number of responses from nonultrapeers per query to measure the file transfer load balance. More results returned to a querying peer means more possible candidates for a user to select for downloading. The multiple choices of possible file providers can build a good basis to further develop a selection algorithm which can balance the file transfer load among several sources. Fig. 16 shows the comparison of the average number of results per query. As we can see, the DiffSearch algorithm returns fewer results per query than the flooding-based search approach because some of the leaf nodes are eliminated from the search scope of DiffSearch.

To improve the number of results per query, we have leaf nodes upload their indices of the shared files to their connected ultrapeers such that both the local indices of ultrapeers and the uploaded indices from leaf nodes are queried during DiffSearch's first round search in the ultrapeer overlay. With the help of index uploading, leaf nodes can have the same opportunity to be queried. Because leaf nodes share a small number of files, they have small indices size, which leads to the small size of index uploading messages. We regard the size of the index uploading messages to be the same as that of query messages in our following evaluation. Fig. 16 shows that the DiffSearch algorithm, enhanced by uploading leaf node indices to ultrapeers, can find a greater number of results per query than that of the flooding-based search approach. The flooding-based search misses files shared by some peers because the TTL limits the search scope and some peers are out of flooding search range. On the other hand, the index uploading of the DiffSearch algorithm aggregates file indices of the entire network into the small size ultrapeer overlay such that all of the file indices are reachable by the first round search. Some files, however, may still not be found in the first round by the DiffSearch algorithm in a dynamic environment due to the partition of the ultrapeer overlay.

The extra benefit of uploading leaf node indices to ultrapeers is that the success rate of the first round search is improved, as shown in Fig. 17. It is because the indices uploaded from leaf nodes can help the first round search to find the indices of files only shared in leaf nodes. With the improvement of the query success rate in the ultrapeer overlay, fewer queries need to resort to the costly second round search such that the overall search cost is reduced, as shown in Fig. 18.

5.2.4 Resilience to Transient Peers

The performance measurement above does not consider the dynamic nature of P2P networks, which is a broadly existing phenomena in current P2P networks. In such spontaneously formed systems, a peer may freely join and leave the overlay network at any time. To simulate peers' dynamic behavior, we assign each peer a lifetime value which follows the Poisson distribution. By varying the mean value of the Poisson distribution, we can investigate the effect of peers' dynamic behavior on the performance of the DiffSearch algorithm. Peers randomly join the overlay network within a specific time range and "die" if their online time equal to the specified lifetime. Dead peers are randomly selected to rejoin the overlay network such that the size of the entire network is stabilized.

Fig. 19 and Fig. 20 show that the overlay success rate decreases and the query traffic increases when peers' average lifetimes become shorter. The peers' dynamic behavior can deteriorate the search performance because of two reasons: 1) The crash of ultrapeers may cause the ultrapeer overlay to be separated into disjointed clusters such that the first round search in the ultrapeer overlay fails due to the limited search scope. 2) The crash of ultrapeers results in the loss of leaf node indices such that the searchable indices in the ultrapeer overlay are reduced. Either of the two reasons above will cause the success rate in the ultrapeer overlay to decrease, which increases the possibility of a costly second round search. The consequence is that the overall query traffic is increased. As mentioned before, the DiffSearch algorithm can detect the

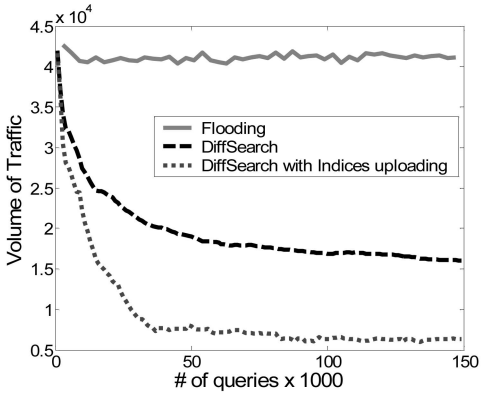


Fig. 18. Traffic comparison of the DiffSearch algorithm.

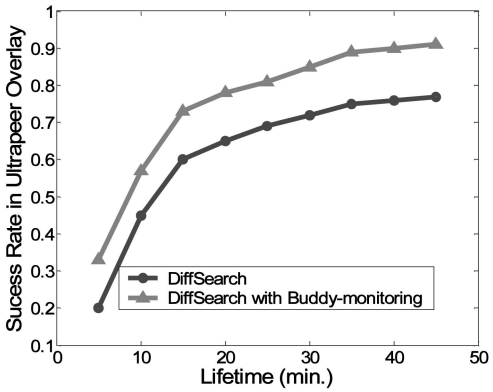


Fig. 19. Impact of dynamic behavior on the success rate of the ultrapeer overlay.

partitioned clusters and eventually reconnect disjointed parts. To prevent the loss of leaf node indices, an intuitive solution would be that a leaf node monitors its connected ultrapeer by continually sending out Ping messages. The extra overhead incurred by Ping messages will increase the burden of ultrapeers.

We propose the solution of ***Buddy-Monitoring*** to improve the robustness of indices with a minimum overhead. The solution of buddy-monitoring makes use of the query/reply messages passing through the ultrapeer overlay to monitor the online status of ultrapeers. In a large scale P2P network such as Gnutella, query/reply messages pass through an ultrapeer in a high frequency. The measurement in [15] shows that a peer receives query requests ranging from 46 to 52 requests per second. Due to such a high density of querying messages, an ultrapeer can always hear from its neighboring ultrapeers within a very short period of time. In other words, if an ultrapeer does not receive any message from its neighbor for a period of time, it is likely that the neighbor is not online any more. To take advantage of this property, buddy-monitoring requires each leaf node to upload its indices to two neighboring ultrapeers, which can monitor each other through the exchanged messages. If one of the ultrapeers in the buddy-monitoring system crashes, the other one selects one of its neighbors and recreates buddy-monitoring by mirroring its indices to the selected neighbor. With the help of buddy-monitoring, the robustness of leaf node indices is improved since the indices

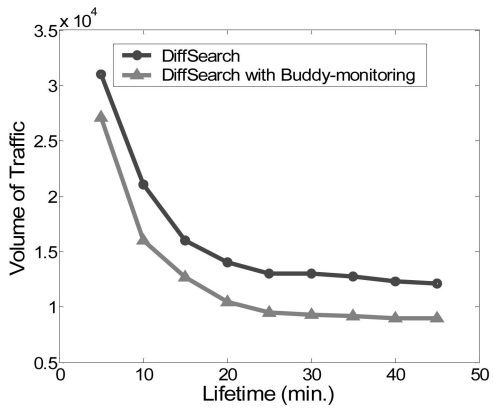


Fig. 20. Impact of dynamic behavior on query traffic.

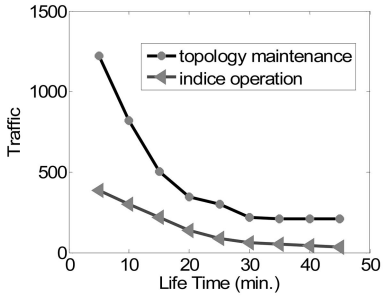


Fig. 21. Overhead of the DiffSearch algorithm.

are lost only when the two ultrapeers in the buddy-monitoring system crash almost at the same time. Fig. 19 and Fig. 20 show that the overlay success rate is improved and the query traffic is reduced when applying the buddy-monitoring solution.

5.2.5 Overhead of the DiffSearch Algorithm

To construct and maintain the hierarchical structure, extra costs are required for the DiffSearch algorithm to set up needed connections. In the enhanced DiffSearch algorithm, the index uploading operations also incur some overheads. In the simulation, we record the total number of messages spent in topology and index maintenance operations and use the metric of average number of messages per query to measure the overhead of DiffSearch algorithm. We vary the mean value of peers' life time to measure the overhead incurred in different dynamic status. As shown in Fig. 21, the overhead of the basic DiffSearch algorithm varies from 1,200 to 260, which is relatively low when compared with the search traffic that varies from 2.7×10^4 to 0.8×10^4, as shown in Fig. 20. Fig. 21 also shows that the index uploading adds a small cost to the overhead since the majority of files are already shared in the ultrapeer overlay.

5.2.6 Improve Performance with Response Caching

Our proposed DiffSearch algorithm is orthogonal to many other P2P search improvement solutions and, therefore, they can be combined together. In this section, we show how to incorporate a *response caching* mechanism into the DiffSearch algorithm to cooperatively improve the search efficiency. Each peer overhears query responses and caches file names together with IP addresses of peers who share the queried files. When a peer receives a query, it will look up its local indices and cached indices as well. A response will be sent back if the query matches one of the two indices. The relationship between the average network traffic and the cache size is shown in Fig. 22, which shows that the search

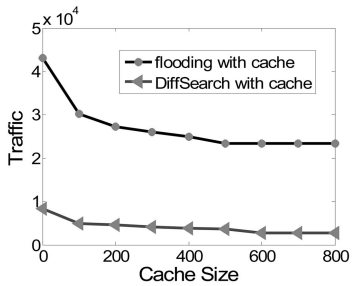


Fig. 22. Performance improvement with response caching.

traffic is reduced from 8,311 to 2,642 in DiffSearch when the cache size is increased from 0 to 800. The cache can help reduce the query traffic because the cache indices increase the possibility for a query of being answered by nearby peers. As a result, the broadcast messages, either in the first round search or the second round search, are limited to smaller scopes and, therefore, incur less flooding traffic. Fig. 22 also shows that the DiffSearch with response caching performs better than the search technique which simply applies the response caching to the flooding algorithm.

6 Related Work

Many approaches have been proposed to improve the search efficiency of P2P networks. Besides the alternative DHT [17], [20], [24], [29] solutions, other approaches try to optimize the unstructured P2P networks by utilizing some of their characteristics.

To take advantage of the query locality, the authors in [23] try to improve the search efficiency by setting up shortcuts among peers which share the same query interests. Query reply caching is also proposed in [16], [22], [25] to reuse query replies cached in nearby peers. The CUP [19] algorithm tries to control the flooding query/reply messages by registering a peer's interest to its neighboring peers. As we discussed before, our proposed DiffSearch algorithm can be combined together with the cache mechanism to cooperatively increase the search efficiency.

The random walk [11], [13] is suggested to avoid the exponentially increasing flooding traffic by selecting only several searching paths among peers. As indicated in the same papers [11], [13], the random walk can achieve a high search success rate when it is combined with the replication technique, i.e., files need to be replicated in the network to increase the possibility of being found by the random walk search. DiffSearch is different from the random walk in that we group the peers with high query answering capabilities together and query those peers before searching the entire work.

The heterogeneity of P2P networks is observed in [3], [21]. Nevertheless, how to exploit the heterogeneity to improve search efficiency is not suggested in these papers. It is explicitly advocated in [14] to make Gnutella scalable by taking advantage of its heterogeneity. Gia [4] proposes controlling the query flow based on the peers' query processing capabilities. By assigning tokens to its neighbors, each peer will not be overloaded since the peer only accepts the queries "traded with" tokens. All the work focuses on how to balance the query load according to their CPU power and network bandwidth and aims to decrease the query dropping rate in the flooding. The DiffSearch algorithm is different from the previous work [4], [14] in that we propose improving the search efficiency directly by utilizing file sharing heterogeneity. The DiffSearch algorithm selects ultrapeers based on the number of shared files. The criterion of the number of shared effective files can be combined with other metrics, including CPU power and network bandwidth suggested in [14], to achieve both load balance and search efficiency at the same time.

An optimal P2P structure is proposed in [6], [7], in which the number of links of each node should be proportional to the square root of the popularity of its content. Such a topology can increase the possibility of a query being forwarded to a peer with popular content. The paper further proposed selectively forwarding a query to neighbors with more content. DiffSearch shares the same idea with this work in that content-rich peers should be queried with higher priorities. The difference between these two approaches is that the DiffSearch algorithm groups all of the content-rich peers together and queries them without searching the entire network.

The superiority of cluster-based P2P networks and super node-based hierarchical P2P networks is theoretically analyzed in [5], [8], [27], [28]. Our approach is different from this work in two aspects: 1) The DiffSearch uses the number of shared effective files as the criterion to select ultrapeers and forms the content-rich ultrapeer overlay. Based on this ultrapeer overlay, a query can be answered by a small portion of peers with high probability before it is broadcast to the entire network. 2) The ultrapeer overlay can be self-maintained by the DiffSearch algorithm without depending on any global operations.

Different from the self-maintained ultrapeers in DiffSearch, the supernodes in KaZaA [2] depend on a proprietary fixed infrastructure to construct and maintain the ultrapeer overlay. The ultrapeers in Gnutella [1] are selected based on the metrics of bandwidth other than query answering capability and there is no guarantee that the ultrapeers in the Gnutella network are formed as a connected overlay. Some preliminary results have been presented in [26]. This paper further develops the DiffSearch algorithm by addressing the load balance and transient peer issues on top of the previous work.

7 Conclusion and Future Work

In this paper, we propose the DiffSearch algorithm, a fully distributed approach which can evolve a two-tier hierarchical structure from a spontaneously formed ad hoc P2P network. By hitchhiking the topology operations to query/reply messages and prompting content-rich peers to the ultrapeer overlay, the DiffSearch algorithm can achieve significant performance improvement with a little overhead on topology maintenance and index operations. It is notable that the DiffSearch algorithm is orthogonal to many existing P2P optimization algorithms. For example, the random walk or DHT can still be applied in the ultrapeer overlay to further reduce the search traffic, which will be investigated in our future work.

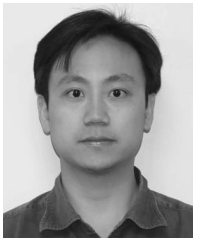
Acknowledgments

This work was supported in part by the US National Science Foundation under grants CCF-0325760, CCF-0514078, and CNS-0549006.

References

[1] "The Gnutella Protocol Specification 0.6," http://rfc-gnutella.sourceforge.net/developer/testing/index.html, 2003.
[2] "KaZaA," http://www.kazaa.com, 2003.
[3] E. Adar and B.A. Huberman, "Free Riding on Gnutella," http://www.firstmonday.dk/issues/issue5_10/adar/, 2003.
[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-Like P2P Systems Scalable," *Proc. SIGCOMM,* 2003.

[5] B. Cooper and H. Garcia-Molina, "SIL: Modeling and Measuring Scalable Peer-to-Peer Search Networks," http://dbpubs.stanford.edu:8090/pub/2003-12, 2003.
[6] B.F. Cooper, "An Optimal Overlay Topology for Routing Peer-to-Peer Searches," *Proc. IFIP/ACM Middleware Conf.,* 2005.
[7] B.F. Cooper, "Quickly Routing Searches without Having to Move Content," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2005.
[8] A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.
[9] E. Damiani, S. Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks," *Proc. ACM Conf. Computer and Comm. Security (CCS),* 2002.
[10] R. Dingledine, N. Mathewson, and P. Syverson, "Reputation in P2P Anonymity Systems," *Proc. Workshop Economics of P2P Systems,* 2003.
[11] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," *Proc. INFOCOM,* 2004.
[12] L. Guo, S. Jiang, L. Xiao, and X. Zhang, "Fast and Low Cost P2P Searching by Exploiting Localities in Peer Community and Individual Peers," *J. Parallel and Distributed Computing,* vol. 65, no. 6, pp. 729-742, 2005.
[13] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. 16th ACM Int'l Conf. Supercomputing,* 2002.
[14] Q. Lv, S. Ratnasamy, and S. Shenker, "Can Heterogeneity Make Gnutella Scalable?" *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2002.
[15] E.P. Markatos, "Tracing a Large-Scale Peer to Peer System: An Hour in the Life of Gnutella," *Proc. Second IEEE/ACM Int'l Symp. Cluster Computing and the Grid,* 2002.
[16] S. Patro and Y.C. Hu, "Transparent Query Caching in Peer-to-Peer Overlay Networks," *Proc. 17th Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2003.
[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM,* 2001.
[18] Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J.d. Groot, and E. Lear, "RFC 1918 Address Allocation for Private Internets," http://www.ietf.org/rfc/rfc1918.txt?number=1918, 1996.
[19] M. Roussopoulos and M. Baker, "Controlled Update Propagation in Peer-to-Peer Networks," *Proc. USENIX Ann. Technical Conf.,* 2003.
[20] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Middleware Conf.,* 2001.
[21] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking Conf. (MMCN),* 2002.
[22] K. Sripanidkulchai, "The Popularity of Gnutella Queries and Its Implications on Scalability," http://www2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html, 2002.
[23] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proc. INFOCOM,* 2003.
[24] R.M. Stoica, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM,* 2001.
[25] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed Caching and Adaptive Search in Multilayer P2P Networks," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS),* 2004.
[26] C. Wang, L. Xiao, and P. Zheng, "Differentiated Search in Hierarchical Peer to Peer Networks," *Proc. Int'l Conf. Parallel Processing (ICPP),* 2005.
[27] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network," *Proc. Int'l Conf. Data Eng. (ICDE),* 2003.
[28] B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-Peer Networks," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.
[29] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *J. Selected Areas in Comm.,* 2004.



Chen Wang received the BS and MS degrees from Northeastern University, China, in 1996 and 1999, respectively. He is currently a PhD student in computer science and engineering at Michigan State University. His research interests are in the areas of distributed systems and computer networking, including peer-to-peer systems and sensor networks. He is a student member of the IEEE and the IEEE Computer Society.

Li Xiao received the BS and MS degrees in computer science from Northwestern Polytechnic University, China, and the PhD degree in computer science from the College of William and Mary in 2002. She is an assistant professor of computer science and engineering at Michigan State University. Her research interests are in the areas of distributed and Internet systems, overlay systems and applications, and sensor networks. She is a member of the ACM, the IEEE, the IEEE Computer Society, and IEEE Women in Engineering.