DiCAS: An Efficient Distributed Caching Mechanism for P2P Systems

Chen Wang, *Student Member*, *IEEE*, Li Xiao, *Member*, *IEEE*,
Yunhao Liu, *Member*, *IEEE*, and Pei Zheng, *Member*, *IEEE*

Abstract—Peer-to-peer networks are widely criticized for their inefficient flooding search mechanism. Distributed Hash Table (DHT) algorithms have been proposed to improve the search efficiency by mapping the index of a file to a unique peer based on predefined hash functions. However, the tight coupling between indices and hosting peers incurs high maintenance cost in a highly dynamic network. To properly balance the tradeoff between the costs of indexing and searching, we propose the distributed caching and adaptive search (DiCAS) algorithm, where indices are passively cached in a group of peers based on a predefined hash function. Guided by the same function, adaptive search selectively forwards queries to "matched" peers with a high probability of caching the desired indices. The search cost is reduced due to shrunk searching space. Different from the DHT solutions, distributed caching loosely maps the index of a file to a group of peers in a passive fashion, which saves the cost of updating indices. Our simulation study shows that the DiCAS protocol can significantly reduce the network search traffic with the help of small cache space contributed by each individual peer.

Index Terms—Peer-to-peer, query response, flooding, distributed caching and adaptive search, search efficiency.

1 Introduction

Compared with a structured P2P network [18], [23], [30], [33], an unstructured P2P network is less efficient due to its blind flooding search mechanism. However, unstructured P2P systems, such as Gnutella and KaZaA, still retain high popularity in today's Internet community because of their simplicity. In a Gnutella-like P2P system, a query is broadcast and rebroadcast until a certain criterion is satisfied. If a peer receiving the query can provide the requested object, a response message will be sent back to the source peer along the inverse of the query path.

The Breadth First Search behavior in a Gnutella system causes exponentially increased network traffic. Measurements in [19] show that even given that 95 percent of any two nodes are less than 7 hops away and the message time-to-live (TTL = 7) is preponderantly used, the flooding-based routing algorithm generates 330 TB/month in a Gnutella network with only 50,000 nodes, in which 91 percent of the traffic were query messages and 8 percent were PING messages. Studies in [27] and [25] show that P2P traffic contributes the largest portion of the Internet traffic based on their measurements on some popular P2P systems, such as FastTrack (including KaZaA and Grokster), Gnutella, and DirectConnect. The inefficient blind flooding search technique causes the unstructured P2P systems being far from scalable [20].

Many efforts have been made to avoid the large volume of unnecessary traffic incurred by the flooding-based search in unstructured P2P systems. Distributed Hash Table (DHT) algorithms try to improve the search efficiency by mapping the index of a file to a unique peer based on predefined hash functions. Following the routing table, a query can be directly forwarded to the mapped peer instead of being blindly flooded. However, the tight coupling between indices and hosting peers incurs high maintenance cost in a highly dynamic network. To balance the tradeoff between the costs of indexing and searching, we propose the distributed caching and adaptive search (DiCAS) algorithm, where indices are passively cached in a group of peers based on predefined hash functions. Guided by the same hash mapping functions, adaptive search selectively forwards queries only to matched peers with a high probability to provide the desired cache indices. In the DiCAS algorithm, each node randomly takes an initial value in a certain range [0..M-1] as a group ID when it participates into the P2P system. We define that a query matches a peer if and only if the following equation is satisfied: Peer Group ID = hash(query) Mod M.

Under the DiCAS protocol, a query response will only be cached in matched peers. The query forwarding will also be restricted to matched peers. The consequence is that the entire search space is virtually divided into multiple layers. Each layer consists of peers labeled with the same group ID. A Query is restricted in the matched layer where the targeted indices are cached. The query traffic is reduced due to the shrunk searching space. Fig. 1 shows an example when M equals 3. Different from the DHT solutions, distributed caching loosely maps the index of a file to a group of peers through passive caching. While a query still needs to be flooded to a group of peers instead of being

- *C. Wang and L. Xiao 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.*
- *Y. Liu is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: liu@cs.ust.hk.*
- *P. Zheng is with Microsoft, One Microsoft Way, Redmond, WA 98052. E-mail: peizheng@microsoft.com.*

Manuscript received 12 May 2004; revised 7 Mar. 2005; accepted 8 Sept. 2005; published online 24 Aug. 2006.
Recommended for acceptance by J. Fortes.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0122-0504.

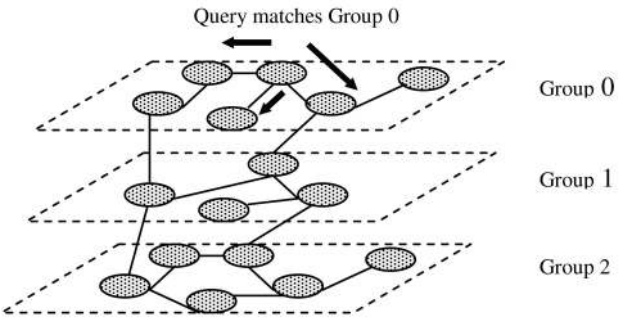

Fig. 1. Flooding in a multilayer P2P network.

routed directly to a specific one, the cost of indices build-up and maintenance can be reduced.

Our work can also be viewed as an enhancement to the uniform index cache mechanism (UIC), which utilizes the query locality to improve search efficiency in Gnutella-like P2P networks. In UIC, each peer caches passing by query responses with the hope that later queries can be answered by its nearby cached query responses instead of being forwarded further. The DiCAS algorithm outperforms UIC in two aspects. First, the DiCAS demands less caching space in each individual peer since it only caches matched query response instead of all passing by responses. Second, the DiCAS is more efficient in search traffic because a query is intentionally directed to peers with high probability of answering that query.

Our simulations have shown that the DiCAS protocol can significantly reduce the network traffic incurred by search in unstructured P2P systems with the help of small cache space contributed by each individual peer. The contributions of our study are as follows:

- An index cache-enabled peer was implemented and deployed in the real Gnutella network. Detailed behavior of the index caching in the Gnutella network has been measured and characterized.
- A distributed caching mechanism was put forward which is more storage efficient than the uniform index caching scheme proposed before.
- A concrete and approachable method based on modulus operation is proposed to divide the P2P network into multiple layers, such that each query is limited to a smaller searching space without degrading the search quality.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 presents our implementation of Index Cache-enabled Gnutella Clients, and experimental results on the index cache-enabled clients connecting to the Gnutella P2P network. Section 4 describes the Distributed Caching and Adaptive Search scheme. Section 5 describes our simulation methodology. Performance evaluation of the DiCAS is presented in Section 6. Our study is concluded in Section 7.

2 Related Work

Several caching mechanisms have been proposed to improve search efficiency in decentralized unstructured P2P systems. The DHT approaches [18], [23], [30], [33] were proposed to avoid query flooding by building distributed indices among participated peers based on predefined hash functions. However, the cost of indices maintenance makes it difficult to deploy DHT solutions to a highly dynamic P2P network such as Gnutella.

The authors in [28] analyze the characteristics of Gnutella queries and their popularity distribution, and propose that each peer caches query strings and results that flow through it. Similar to [28], the authors in [14] observed that submitted queries exhibit significant amounts of locality based on one hour of Gnutella traffic, and proposed a caching mechanism in which peers cache query responses according to the timestamp the query is responded to. The effectiveness of caching query results has been shown by simulations in both [28] and [14]. All the work above suggests a uniform index caching (UIC) mechanism. The UIC causes a large amount of duplicated and redundant cache results among neighboring peers, and the effect cache space can be increased by eliminating the duplication between neighbors, which is shown by our experiments.

Based on an observation of query locality in peers behind a gateway of an organization, transparent query caching [17] is proposed to cache query responses at the gateway. In contrast to the work in [17], our approach can fully take advantage of the internal nodes' resources and avoid the bottleneck and single point of failure in the centralized cache.

Caching file content has also been studied. An ideal cache (infinite capacity and no expiration) simulator is built [25] to investigate the performance of content caching for KaZaA P2P network. It has been shown that caching would have a great effect on a large-scale P2P system on reducing wide-area bandwidth demands. Compared with the index caching, the content caching is less storage efficient.

The K-walker [13] proposes a random walk search mechanism and evaluates three different strategies to replicate data (file content or query responses) on multiple peers. Uniform strategy creates a fixed number of copies when the item first enters the system. Proportional strategy creates a fixed number of copies every time the item is queried. In a square-root replication strategy, the ratio of allocations is the square root of the ratio of query rates. Our work is different from K-walker in that the query is more tightly connected to the cache. In DiCAS, the query is forwarded intentionally to the peers with a high probability to provide the desired cache results.

The superiority of the cluster-based P2P network has been mathematically proved in [7], while the mechanism of how to break the P2P network into multiple clusters has not been mentioned yet.

3 Experiments of Index Caching in Gnutella Network

3.1 Overview of Experimental Setup

To investigate the performance impact of index caching in a real, large-scale peer-to-peer network, we modified the LimeWire Gnutella servant [3] with support of Gnutella protocol v0.6 [2], and developed an index Cache-enabled Gnutella Client (CGC). Compared with the normal Gnutella client, our CGC peer is able to create and maintain a local index cache by overhearing traversing query response results in an existing Gnutella network. As a result, other

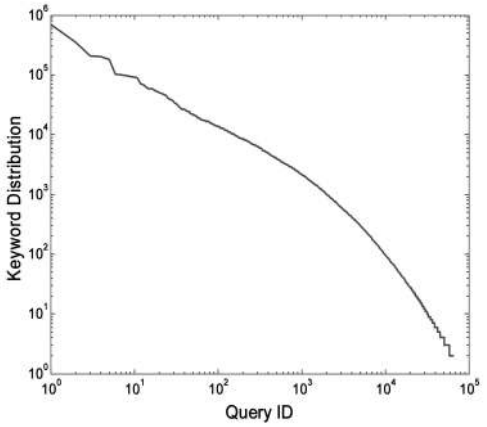

Fig. 2. Keyword distribution of the CGC query trace. Query IDs are ordered by the frequency.

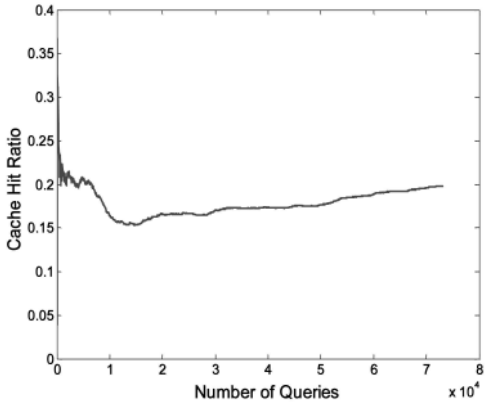

Fig. 3. Cache hit ratio on a single CGC peer with trace-driven query generator. Cache size is 1,024 KB.

peers neighboring to the CGC peer will have the opportunity to utilize the index cache for future searches. We conduct a number of experiments with the CGC peer in Gnutella network to explore the performance improvement by the index caching.

We have also built a traffic monitoring tool that works in conjunction with the CGC peer to trace incoming and outgoing queries and responses, as well as cache hits and misses on the index cache. Two aspects of the dumped P2P network traffic through the CGC peer could be used. First, by analyzing the query patterns and locality in both space and time we could gain more insight into some fundamental issues of index caching, such as how to determine cache size and cache expiration time. Second, the trace data can be used as a traffic source to flexibly test our CGC peer implementation in a variety of scenarios with comparable results. In this sense, we have actually built a cache-aware P2P network testbed with the CGC experimental setup and the traffic monitoring and trace-driven tool. In the following section, we present four test scenarios that employ either single or multiple CGC peers to examine the benefit and overhead of index caching in a Gnutella network.

The CGC peer is a PC with a 2.4GHz Pentium IV processor, 1 Gigabyte memory, and Ethernet connection to the campus network. The CGC software is running on Linux. We use LRU as the index cache replacement policy. Other cache replacement policies can also be incorporated into CGC. Clearly, different cache replacement policies will have different effects on the hit ratio of the index cache. To examine the impact of cache size on overall performance, we vary the cache size from 2 Kbytes to 64 Kbytes.

3.2 Trace-Driven Single CGC Peer Experiment

Our Gnutella network query trace was collected on one CGC peer on 11 March 2003. Some nonmeaningful words such as articles and propositions were removed from the trace to improve the accuracy of our analysis. The total number of queries was 13,705,339, while 129,293 unique keywords existed in the trace. As shown in Fig. 2, the frequency of query keyword in the trace roughly follows a Zipf distribution, which substantially suggests that the index cache in a Gnutella network could make use of the keyword and query response result locality to improve searching performance. Fig. 3 shows the index cache hit ratio on the single CGC peer in a trace-driven query experiment. The cache size is 1,024 Kbytes in this case. It shows that about 21 percent of total traversing queries will be replied by the single CGC index cache.

3.3 Single CGC Experiment

As shown in Fig. 4a, the single CGC is connected to a Gnutella network, which works as a regular Gnutella client

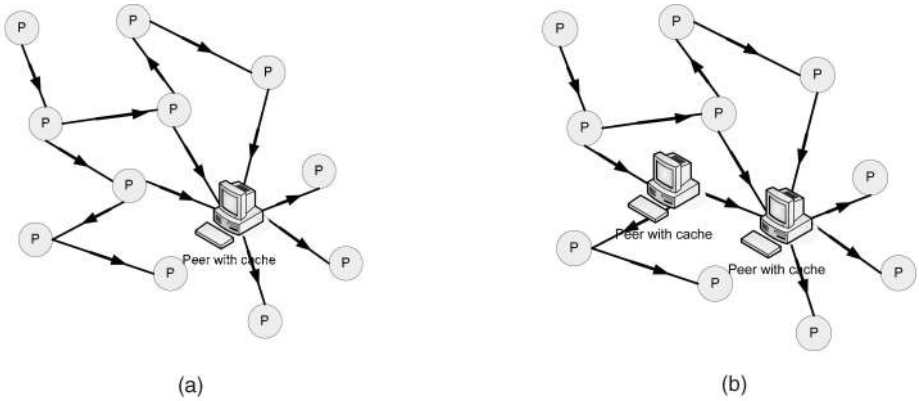

Fig. 4. CGC peers in Gnutella Network. In (a), a single CGC peer connected into Gnutella network, whereas in (b), two CGC neighboring peers connected to Gnutella network.

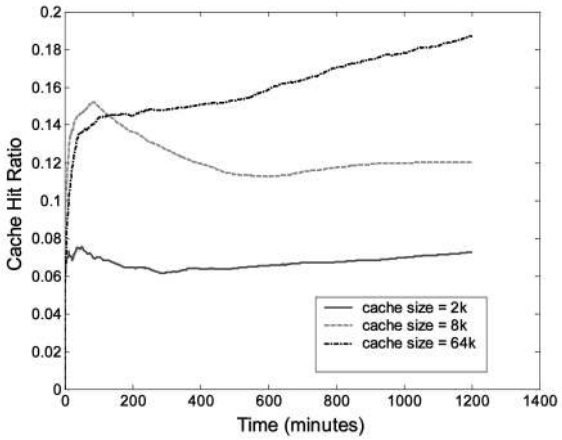

Fig. 5. Cache hit ratio with different cache sizes in single CGC experiment.

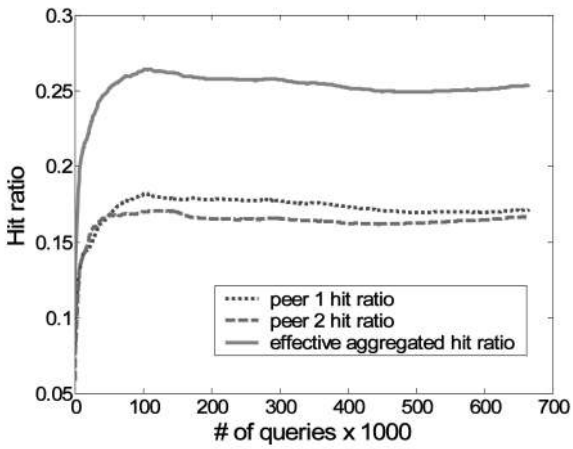

Fig. 6. Aggregate cache hit ratio in the twin CGCs experiment. The cache size on each CGC is 64K.

except that it maintains an index cache. Intuitively, the benefit of index caching will be more evident if the CGC peer could populate a large number of cached items. Hence, the CGC has been configured to be an ultrapeer which has a higher probability to establish connections with others than regular peers, according to Gnutella protocol. We have observed that the number of neighbors of the CGC peer ranges from 10 to 120.

Fig. 5 illustrates the index cache hit ratio as a function of time with different cache sizes in one day. Clearly, when the cache size increases, the hit ratio will increase as well. We identified that the major factors that limit the hit ratio of an index cache are transit search locality of neighboring peers to the CGC peer, and the number of neighboring connections the CGC peer could reach over the time we conducted the experiment. We expect that with a longer warm-up time and allowing more connections to the CGC peer, the cache hit ratio will be further improved.

We also expect if there are more CGC peers participating in the P2P network, the overall searching performance can be improved due to cooperation between the neighboring peers. However, our following experiments show that a large amount of duplicated items are cached among neighbors. Here, we define the duplicated cache items as follows: Suppose two neighboring peers A and B have the same caching item. When peer A receives a query for this item, it will respond to the query and not forward the query to peer B any more. Therefore, the cache item in peer B is not utilized and is regarded as redundant. We define the same content cached in neighboring peers as duplicate cache.

3.4 Twin CGC Peer Experiment

For the purpose of investigating the performance of distributed index caching in a Gnutella network, we connected our two CGC ultrapeers into the network (see Fig. 4b). The two CGC ultrapeers should be neighbors in the overlay such that we could measure their overall contributions to traversing queries with two separated index caches. Due to the inherent overlay nature of P2P systems and Gnutella's topology optimization scheme, the two CGC ultrapeers, even if they are close to each other in the physical network, can hardly maintain a persistent neighboring relation after some up time. To enforce a fixed neighboring relation between the two CGC ultrapeers (the Twin CGCs), a "dummy" regular Gnutella Client was added to the test environment. The "dummy" GC can only have two neighbors (the twin CGCs) in the overlay. It is "dummy" because when forwarding queries between the twin CGCs, it will not decrease TTL of the query such that the two peers connected by the dummy Gnutella Client become virtual neighbors. In order to estimate the cache redundancy between neighboring peers, we intentionally modified the Gnutella clients, such that a peer will continue to forward a query to its neighbors even if it can answer that query. By counting the number of cache hits answered by the same cache items in neighboring peers, we can roughly estimate the duplication between neighboring peers.

In the Twin CGCs experiment, all the cache hits in each CGC were recorded in log files. The comparison between the records of both peers shows a significant cache hit overlap between the neighboring peers. Among the 8,500 cache hits recorded in each peer within two hours, there were 2,741 duplicated cache hits. The overlapped cache hits between two neighboring peers exceeded 32 percent of all the cache hits in one peer. The comparison between the effective aggregated cache hit ratio and the cache hit ratio in a single peer is shown in Fig. 6. We expect more overlap among neighboring peers when multiple peers are fully connected with each other. Those duplicated cache hits are unnecessary since only one of the duplicated cache hit is sufficient to satisfy the correspondent query. The observation above suggests that it is possible to improve the search efficiency by distributing index cache among neighboring peers. Based on the distributed index caching, the search efficiency can be further improved by our adaptive search method which forwards the query only to peers with the matched Group ID.

4 Distributed Caching and Adaptive Search

4.1 Gnutella Protocol

We first briefly introduce a related part of the Gnutella protocol before presenting our proposed DiCAS. Topology maintenance and search operations of the Gnutella network are specified in [2]. Each Gnutella peer connects to several overlay neighbors using point-to-point connections. A peer

sends *ping* messages periodically to check all connections with its direct neighbors, and expects the *pong* messages from them. Typical Gnutella peers will try to maintain a prespecified number (three to five for a normal node, and much more for an ultrapeer) of connections. Gnutella peers overhear all the *pong* and Query Response messages passing by and cache IP addresses of other peers currently alive. If a peer detects that one of its neighbors is offline, it will look up its host cache or connect to a well-known Gnutella host cache server, then randomly create another connection.

In order to locate a file, a source peer floods a query to all its direct neighbors. When a peer receives a query, it checks its local index to see whether it has the queried content. If so, a query response will be returned along the reverse of the query path to the source peer. Otherwise, the query will continue to be broadcasted. In the current Gnutella protocol, query responses are not cached by any peers in the returning path.

4.2 Distributed Caching

In addition to a *local index* that keeps indices of local files, each peer maintains a *response index* which caches the query results that flow through the peer. Each item cached in the *response index* includes the queried file name, and the IP address of the responding peer where the file is located. When a peer receives a query from its neighbor, it will look up the *response index* as well as the *local index*. A query match with either of them will generate a response. Instead of caching query responses in all peers along the returning path, *distributed caching* attempts to cache the responses in some selected peers. The key of *distributed caching* is to determine whether an incoming query response should be cached or not so that the duplicated query responses among neighboring peers can be minimized. In DiCAS, when a peer joins the P2P system, it will randomly take an initial value in a certain range [0..M-1] as its group ID so that all the peers are separated into M groups. A uniform hash algorithm is employed to translate the queried file name string to a hash value. We define that a query matches a peer if the following equation can be satisfied: Peer Group ID = hash(filename) Mod M.

For a passing query response, each peer overhearing the response independently performs a computation on the response using the hash function, and caches this response only when this hash value matches the peer's group ID. For example, when M = 2, all peers are separated into two different groups. Suppose the modulus operation result of a file name's hash value equals 1. Only the peers in group1 will cache this response, as illustrated in Fig. 7.

4.3 Adaptive Search

By exchanging data with neighboring peers, a peer can learn group IDs and other information such as connectivity of its neighbors. Based on this information, a query is selectively forwarded to only neighbors with a group ID that matches the hash value of the desired file name in the query. For example, when a node receives a query which matches the group ID of 1, the query will only be forwarded to neighbors with the group ID of 1. We claim that the group ID is uniformly distributed in the P2P network due to its value being randomly chosen. Benefiting from the group ID's uniform distribution, a query can be forwarded to matched neighbors in most cases. However, it is still possible that query forwarding can be blocked if none of a peer's neighbors have a matched group ID. To avoid the early death of the query, the peer will select a neighbor with the highest connectivity degree to forward the query to in this case. Based on the *adaptive search* algorithm above, the query forwarding will be restricted to peers with the matched group ID. Those peers form a virtual layer which has much smaller searching space than the original P2P network. Based on the modulus operation, the whole network is logically divided into multiple layers and each query will be forwarded within the correspondent layer with matched group ID.

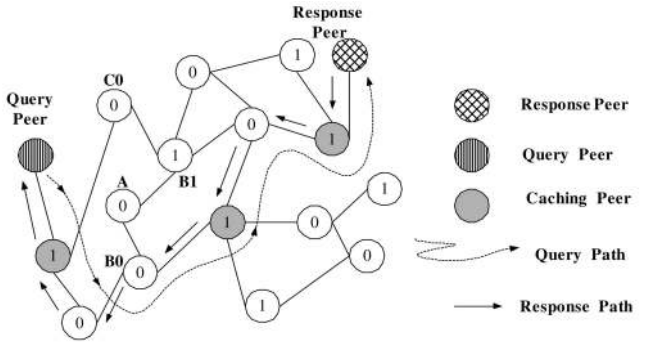

Fig. 7. Cache strategy of DiCAS.

In Adaptive Search, it is possible a query may miss some documents because those documents are shared by unmatched peers. To avoid such a situation, we proposed two solutions: *push-DiCAS* and *select-DiCAS*. In push-DiCAS, each peer will push the indices of unmatched documents to its neighbors whose peer ID is matched with those document IDs. In Select-DiCAS, instead of only forwarding queries to matched neighbors, one of unmatched neighbors will be selected. These two solutions are discussed in more detail in Section 6.3.

4.4 Cache Update

Cached items may become expired for two possible reasons: the files are not shared by the hosting peers any more; or the hosting peers leave the network. There are two intuitive solutions to address the expiration issue. In the first one, the *checking* approach, a peer with cached items periodically checks with the hosting peers on the freshness of the corresponding contents, and clears obsolete indices. The limitation of this approach is that the overhead incurred by the periodic checking messages cannot be ignored. Meanwhile, it is not easy to determine the frequency of checking messages to well balance the accuracy and the cost. The second approach, *notifying*, is that the hosting peer notifies the caching peers each time when it changes the sharing list. However, this approach needs each peer to keep a list of its caching peers. Otherwise, the flooding has be to used to send out the notification message, which will incur tremendous overhead. Besides, the status change of ungraceful leaving peers cannot be sent out to the caching peers.

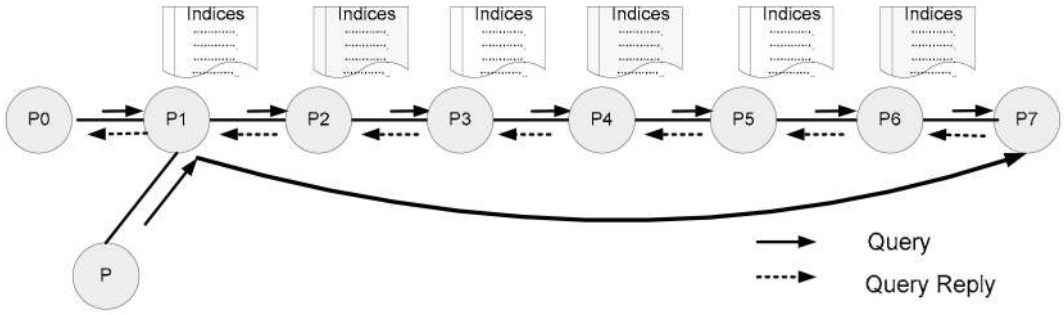

Fig. 8. One more step forwarding.

We propose *one more step forwarding* to address the cached item expiration issue. In this approach, when a query hits a cached item, instead of being responded back directly, the query will be forwarded to the peer which actually shares the file. If the file is not available or the hosting peer is not reachable, the caching peer will clear the expired cache item and continue the query forwarding. Otherwise, the query reply will be sent back. For example, in Fig. 8, whenever a cache hit occurs in a peer P1, the query will be forwarded to the peer P7 who hosts the queried file to verify the consistence between the cache index in P1 and the pointed file in P7. The overhead of network traffic incurred by the extra step forwarding is trivial because the caching peer only forwards the query to the hosting peer instead of all its neighbors.

However, the query response time will be increased by one hop on average because each successful query response generated from cached indices also spends the extra step to verify the cache consistence, which is not necessary. To reduce the unnecessary verification overhead, we attach a timestamp T_c to each cached item c to record the last validation time of c. For example, if a cached item c is created at time t_1, the value of T_c is set to t_1. Next time, if c is verified by *one more step forwarding* at time t_2, T_c will be updated to t_2 accordingly. We define a time interval τ. For a query hit of a cached item c at time t, if $T_c < t \leq T_c + \tau$, a query response generated from the cached item will be sent back directly. Otherwise, if $t > T_c + \tau$, the query will be forwarded to the hosting peer to verify the content of the cached item c. If the shared file pointed by the cached item c is still available, the query response will be sent back and the time stamp T_c will be updated to the latest validation time. Otherwise, the cache item c will be cleared. The rationale behind the timestamp approach is that there is a high probability that the cache items will not expire during a short time interval. The timestamp solution can effectively improve the response delay incurred by the *one more step forwarding*, especially for popular cache items, which may be hit frequently within time interval τ.

The second approach we propose to address cached item expiration is *backtrack clearing*, in which the caching peer's IP address along with the reply, will be sent back to the querying peer along the inverse path of querying. If the querying peer fails to retrieve the file based on the reply generated from the caching peer, it will send a *clearing* message to the caching peer through a temporary connection. When the caching peer receives the *clearing* message, it will clear the expired cache item and continue to forward the originally terminated query. An example of the *backtrack clearing* approach is shown in Fig. 9. In this approach, if the cached item is valid, it will incur no overhead and achieve the same performance as the caching algorithm without any extra verification procedure. However, if the cached item is invalid, the extra overhead involves message forwarding between the querying peer P, caching peer P_4 and the hosting peer P_7, i.e., 4 hops, which is worse than the *one more step forwarding*, in which the verification messages are only forwarded between caching peer P_4 and the hosting peer P_7, i.e., 2 hops. Therefore, *one more step forwarding* favors the false cache items, while *backtrack clearing* favors the positive cache items. The natural choice is to combine them together.

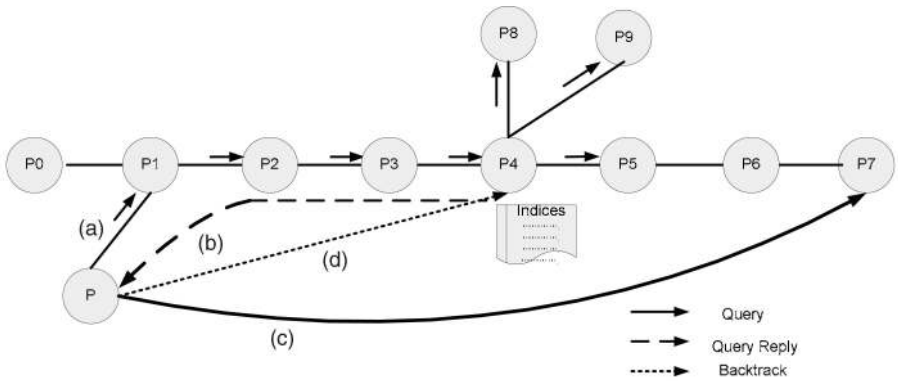

Fig. 9. Backtrack Clearing: (a) Node P sends out a query. (b) Node P4 responses back the query reply generated from cache index. (c) Based on the query reply, node P requests file from node P7. (d) If the file is not available, a backtracking clearing message is sent back from node P to node P4.

TABLE 1
Hash Value Distribution of Gnutella Queries

%	M=2	M=3	M=4
Group ID 0	52%	36%	28%
Group ID 1	48%	32%	24%
Group ID 2	0	32%	24%
Group ID 3	0	0	24%

In the combined approach, if the cache item is hit within the time period of $[T_c, T_c + \tau]$, which implies the cache item is valid with high probability, *backtrack clearing* will be used to verify the consistence; otherwise, *one more step forwarding* is applied.

4.5 Further Discussion

We continue some further discussions below when DiCAS is applied to the P2P network.

4.5.1 Uniform Distribution of Index Cache

It is possible that some peers with a certain group ID will cache much more indices than others if the queries cannot be uniformly mapped to each group ID. The hot spot caused by the uneven mapping may seriously limit system performance. The analysis of query trace shows that the queries in the Gnutella network can be evenly mapped to each Group ID based on different modulus operations. See Table 1 for the detailed results.

4.5.2 Partial Match

The hash function is used in our DiCAS protocol. However, its weak support for partial matches prevents it from being widely applied in the P2P search. The partial match occurs if only partial keywords instead of the whole filename are used in a query, which is a prevalent search behavior in Gnutella-like file sharing systems. For example, suppose a user is looking for the file with the name $F = key_1 + key_2 + \ldots + key_n$. Typically, the user will only use the partial keyword $K = key_l + key_m + key_n$ to search the file F and hope the system can return a list of filenames that contains the combined searching keywords. In DiCAS algorithm, in order to match a query reply with its caching peers, we use the function $Peer\ Group\ ID = hash(F) \mod M$ to calculate its peer group ID, where the file name F is retrieved from the query reply. However, in the query forwarding, the partial keywords K is used to calculate the peer group ID by $Peer\ Group\ ID' = hash(K) \mod M$. Obviously $Peer\ Group\ ID' \neq Peer\ Group\ ID$ if $K \neq F$. Therefore, if partial keywords K are used in a query, the DiCAS algorithm may forward the query to unmatched peers which contains no cache items pointed to the desired file F. To address the partial match problem, we propose to calculate the peer group ID in query reply based on query keywords K instead of the file name F. In order to do so, an extra field is appended to the query reply message to record the query keywords K. When the query reply is forwarded back to the initial peer, the keywords K instead of file name F is retrieved to decide the matched peers to cache the response. Due to the consistent computation of peer group IDs between query and query reply, the query can be forwarded to the matched peers with desired cached items. It is possible that multiple cached items K_i are mapped to the same shared file F. Although users repeat similar query patterns to search the same file, there still exist multiple combinations of partial keywords K_i which are mapped to the same file F. The multiple mapping between cached item K_i and shared file F requires more caching space for each sharing object.

5 Simulation Methodology

5.1 Considerations of P2P Simulation

It is not realistic for us to make a considerable number of peers in Gnutella network configured with the support of DiCAS for the purpose of evaluating performance improvement. We decided to develop a DiCAS simulator for a large-scale cache-aware P2P network. We chose to simulate each peer's message-level behaviors as an effort to investigate searching and index caching on all peers across the entire network. Each simulated peer is able to send queries, modify local and response index caches, and generate responses based on both caches. Our previous experiences on network simulations and experiments show that simulation configurations and parameters strongly influence the validity of simulation results. In this section, we summarize a list of network parameters used in the simulations of previous studies.

The parameters that determine the simulation scenarios fall into three categories: network and topology parameters, workload parameters, and initial content/keyword distributions over the network. Content popularity at a publisher follows a Zipf-like distribution (also known as Power Law) [4], [5], where the relative probability of a request for the ith most popular page is proportional to $1/i^{\alpha}$, with α typically taking on some value less than unity. The observed value of the exponent varies from trace to trace. The request distribution does not follow the strict Zipf's law (for which $\alpha = 1$), but instead follows a more general Zipf-like distribution. The query word frequency does not follow a Zipf distribution [10], [31]. The user's query lexicon size does not follow a Zipf distribution [31]. Instead, it exhibits a heavy tail distribution.

Both the overall traffic and the traffic from the 10 percent most popular nodes are heavy-tailed in terms of host connectivity, traffic volume, and average bandwidth of the hosts [27]. Schlosser and Kamvar [26] suggest a log-quadratic distribution ($10^{-\alpha^2}$) for stored file locality and transfer file locality. The length of time that nodes remain available follows a log-quadratic curve [26], which could be approximated by two Zipf distributions.

Research on content searching in P2P networks generally uses simulation to illustrate the effectiveness of the underlying approach. Thus, the problem of choosing a decent abstraction level becomes a critical issue, which in turn determines what simulation configuration is needed for such a scenario. For specific simulation, one should carefully choose related parameters and distributions such that the simulation results and observations are reasonable.

TABLE 2
Configuration Parameters for P2P Network Simulation

Parameter Description	How to choose?	Example
Network and Topology Configuration		
Number of Nodes in the network	Depends the size of the network you want to simulation. Gnutella has a rough number of 16000 nodes that accept incoming connections, 80000 nodes online, as of March 2003 [1].	[8] chooses 60000 as double of the maximum Gnutella network size in 2000. 8000-40000 in [29].
Network Topology	Can be tree with extra links, or power-law with outdegree exponent. [16] has details about generating power-law topology.	[8] uses both methods. Its outdegree exponent is –2.2088 adopted from [9].
Node Connectivity	Randomly chosen.	Simulation based with average of 3.4 in [29], Regular graph based of 4 in [15], randomly 3 in [11].
Network Dynamics	Median session duration for a node is one hour [24].	Randomly chosen node failure [30].
TTL	Mostly 7 to match current Guntella network	[29] uses 3.
Workload Configuration		
Query Generation	Determines which nodes to issue a query and query interarrival distribution.	Poisson in [12, 30]. Randomly selected nodes in [11, 21]. Poisson/Pareto in [22]. Zipf for node distribution in [21].
Traces	Used by trace-driven simulation.	[29] uses one-day traces of KaZaA and Gnutella from CMU, and two web request traces.
Collecting traces	For Gnutella network, use crawler. Web traffic can be obtained from system administrator.	Gnutellasim has a open-source crawler.
Content Distribution		
Number of documents requested for each query	Determine when to terminate a query when enough number of hits are found	[8] uses 10
Distribution of documents over all nodes in the network	Determine which node has documents, and how many documents.	80/20 in [8]. [29] assumes that in the web/Gnutella traces, if a host requests a file/URL, the host will share the content. Each node has 3 randomly chosen documents from a pool of 2000 in [11]. [32] states that average number of files shared by a peer is 340.
Keyword in a document	Could be meta-data.	Each document has a set randomly chosen keyword, but the number of keywords chosen is a linear function with the document number in [12], randomly chosen 3 keywords from a pool of 1000, or follows a Zipf distribution in [11].
Keyword in a query	How to choose keywords for a query?	Only 1 keyword, Zipf-like (α=1.2, 0.8, 2.4, similar results) in [12, 21].

Table 2 shows the list of network parameters that might be used in P2P network simulation.

5.2 Our Simulation Configuration

In our simulation configuration, We use BRITE [6] to generate the underlying network topology with 50,000 nodes. Above this underlying network, we generate the Power-law [16] overlay topology of a P2P network with 10,000 nodes and average connectivity degree of 3. Node dynamic behavior is an important factor to impact the search performance of P2P networks. To simulate the behavior of frequent node joining and leaving, we model the node joining process as a Poisson distribution with the rate $\lambda = 8$. The online time of a node is modeled as a exponential distribution with the average online time of $1/\mu = 1,000$.

We examine the impact of index caching on searching efficiency in terms of keyword matching. Hence, in our simulation, we only look at single keyword matching rather than document matching and semantic layer searching. Blind flooding in the Gnutella network is simulated by conducting a Breath First Search algorithm from a specific node. A search operation, bounded by TTL of 7, is simulated by randomly choosing a peer as the sender, and a keyword according to Zipf distribution. In each simulation session, a large number of search operations are simulated sequentially. While receiving a query, a peer will consult its local index and its query response index cache using the searching keyword for possible matches. The trace we collected (described in Section 3.2) is used in our simulation.

6 Performance Evaluation

A well-designed search mechanism should seek to optimize both efficiency and user satisfaction. Efficiency focuses on better utilizing resources, such as bandwidth and processing power, while user satisfaction focuses on user-perceived qualities, such as if they can find the desired files and how long it will take to find the files. To evaluate the effectiveness of DiCAS, we will use three performance metrics defined below: query success rate, query response time, and traffic overhead incurred by queries. **Network**

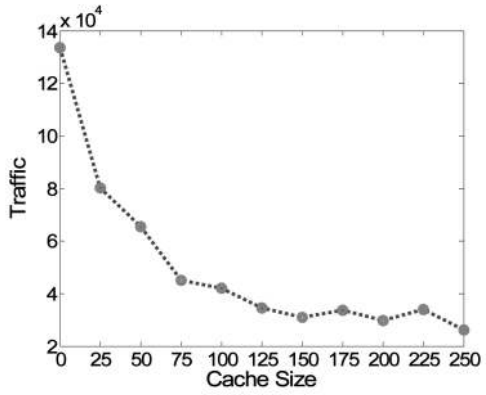

Fig. 10. Traffic comparison for different cache size.

Traffic incurred by each query is defined as the total number of forwarded hops in the underlying network. **Success Rate** is the percentage of queries which can be responded by at least one query reply. **Response Time** is the minimum number of hops of the underlying network for the query reply to be forwarded back to the source peer.

6.1 Effectiveness of Uniform Index Caching

In the first simulation, we examine the effectiveness of uniform index caching (UIC) scheme in which all peers in a query response path will cache the query response. Blind flooding is still used in UIC to forward queries. Fig. 10 and Fig. 11 show the average traffic and the average query response times for different cache sizes. The results show that the UIC approach with a moderate cache size of 50 can significantly reduce network traffic by 59 percent, and reduce query response time by 32 percent. However, further increasing cache size in each peer would not improve performance proportionally. One reason we have mentioned is that there exists a large amount of overlapped query responses among neighboring peers in UIC, which can limit the performance improvement of caching query responses.

6.2 Effectiveness of DiCAS

Aiming at further improving search efficiency, we propose DiCAS to cache query responses in selected peers and forward the query to peers with matched group ID. DiCAS is evaluated in this section using M = 3, which logically divides the search space into three layers.

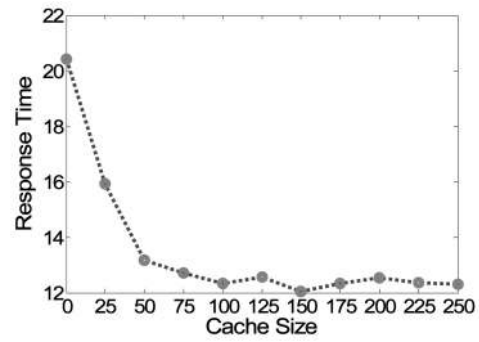

Fig. 11. Response time comparison for different cache size.

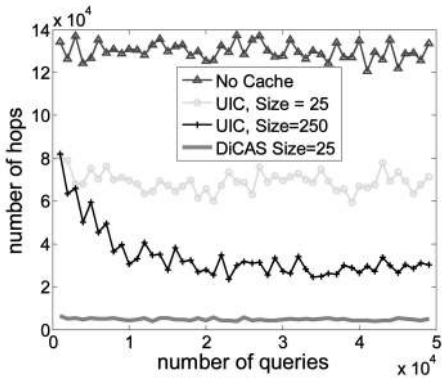

Fig. 12. Traffic comparison of UIC versus DiCAS.

Fig. 12 compares the average traffic of UIC and DiCAS. We can see that DiCAS outperforms UIC by 80 percent in terms of average traffic reduction. When we measure the query success rate of UIC and DiCAS, we find that UIC can keep the same query success rate as original flooding without caching (see Fig. 13). However, Fig. 13 also shows that query success rate of DiCAS is decreased by 80 percent compared with UIC. Because the DiCAS protocol only forwards a query to some selected neighboring peers instead of all neighboring peers, it is likely that the query will miss some peers who have queried results. There are two reasons for a query to miss matched peers.

First, some matched peers may be missed. In DiCAS, a source peer forwards its query to those neighboring peers whose group ID matches the query. Some other neighbors are nonmatched neighbors. However, the nonmatched neighbors' neighbors may have matched group ID with this query, but may be never reached by the query. See Fig. 7 again for an example, Peer A has two neighbors $B1$ and $B0$. Peer $C0$ is $B1$'s neighbor, but two hops away from A. Assume that peer $B0$ and $C0$ have the same group ID (e.g., $GID = 0$), and peer $B1$ has another group ID (e.g., $GID = 1$). If peer A initiates a query that matches $GID = 0$, the query will only be forwarded to peer $B0$. Peer $B1$ will not receive the query, so the query may not reach C0, but C0 is indeed a matched peer that should be queried.

Second, some matched objects may be missed. When a peer joins, it selects a group ID, but this cannot guarantee that all its local objects will match the group ID. In this case, there are some objects that do not match the owner's group

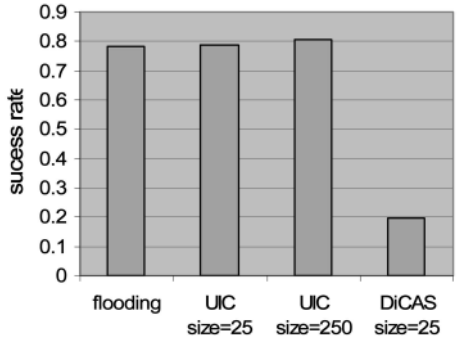

Fig. 13. Success rate comparison: UIC versus DiCAS.

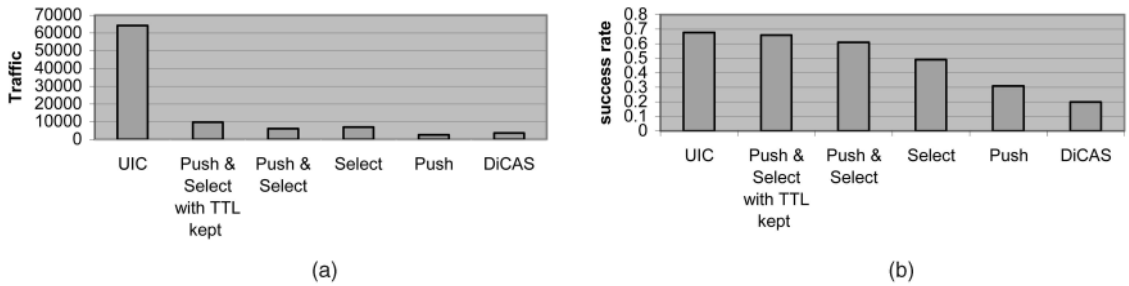

Fig. 14. Solutions to improve success rate of DiCAS.

ID and will never be queried, forming some dead corners. Thus, some of the objects, even though they are available, may not be found by many queries.

Motivated by the above two reasons, we proposed two solutions to address the problem of query success rate degradation in DiCAS, which are described and evaluated next.

6.3 Solutions to Improve the Query Success Rate

The analysis above shows that the DiCAS approach reduces the search traffic at the price of degrading the success rate. To reconcile the contradiction, we propose several solutions to improve the search success rate of DiCAS algorithm. In the following performance comparison, we use the cache size of 50 units in each individual peer.

The first solution is called *push-DiCAS* that attempts to avoid missing matched objects. When a peer is joining a P2P network and randomly taking a group ID, it computes hash values of the file names for all its sharing objects. If some objects do not match this peer's group ID, the peer will push the indices of these objects to one of its neighboring peers with matched group ID. These neighboring peers will cache the indices of pushed objects with a similar format of a query response indicating whereabouts of the objects. If none of the neighboring peers with matched group ID exists, a peer with the highest connectivity degree will be selected. The whole process is repeated until a certain number of peers with matched group ID are found.

We called the second solution *select-DiCAS* that attempts to avoid missing matched peers. When a query cannot be forwarded any more because all the neighbors of current peer cannot match with the query, instead of ignoring all nonmatched neighboring peers, the peer forwards its query to some nonmatched neighboring peers with high degrees. As a result, matched peers will be reached to continue the search. An improvement to *select* operation is that the TTL value of a query will not be decreased when forwarded by nonmatched peers because those nonmatched peers are temporarily used as forwarding nodes that will not actually match the query. We call this improvement *TTL-Kept* select operation.

Since the *push* operation and *select* operation are orthogonal to each other, we can combine them together to further improve the success rate of DiCAS algorithm. Fig. 14 shows the performance comparison of different solutions, including *base DiCAS*, *push*, *select*, *push & select*, *push & select with TTL-Kept*, and *UIC*. The hollow bar shows the comparison of network traffic incurred by the query message. The traffic is normalized by the traffic of UIC. The comparison shows that the *push* and *select* operations are effective solutions to improve the success rate of DiCAS algorithm. With the combination of *push & select with TTL-Kept*, the DiCAS approach can achieve almost the same success rate as UIC, while reduce the traffic by about 83 percent compared with UIC.

6.4 The Number of Layers

Another question is: What is the proper value for the number of layers that the search space should be divided into by DiCAS? To investigate the relationship between the search performance and the number of layers, we repeat the simulation of DiCAS approach enhanced by the *push & select with TTL-Kept*, varying the number of layers from 2 to 10. The results are shown in Fig. 15, where the number of layers equals to 1 corresponds to UIC approach. We have two observations from Fig. 15.

The success rate will be proportionally decreased when the number of layers increases. The success rate is inverse proportional to the number of layers because with the number of layers increases, the peers with the same group IDs will scatter far away from each other. The sparse distribution lowers the probability for the query of DiCAS algorithm to find the peers within the same group.

The network traffic drops fast at the beginning, then the increasing layer gradually converges to a certain value. The concave curve indicates that it will not reduce the network traffic effectively by further increasing the number layers after some extent. The network traffic converges because there is a trade-off between the searching traffic within

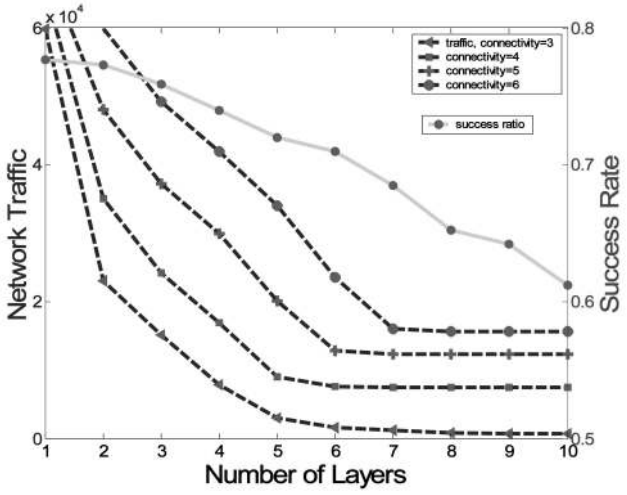

Fig. 15. Comparison for different number of layers.

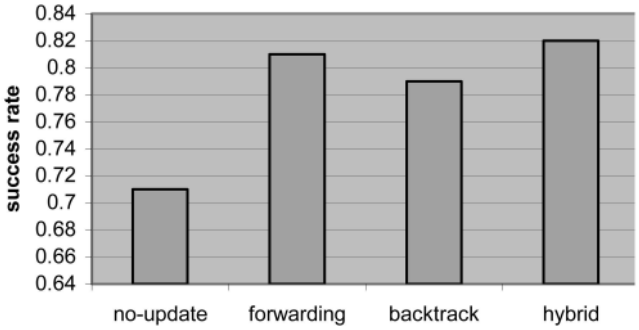

Fig. 16. Success rate comparison of cache update.

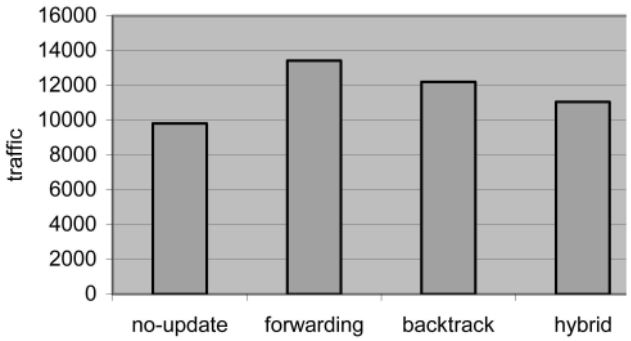

Fig. 17. Traffic comparison of cache update.

layers and the traffic between layers. Increasing the number of layers will divide the whole search space into smaller groups so that the search traffic in each group can be reduced. However, it will increase the probability of partition, i.e., peers within the same group have more chance to be separated far away from each other, such that the search traffic to locate match peers is increased.

We repeated the simulations under different network configurations with various average connectivity degree. We can see from Fig. 15 that for all the connectivity degrees dividing searching space into multiple layers can help to reduce the network traffic at the beginning, but further increasing the number of layers will not reduce the traffic obviously. Since the P2P network is highly dynamic, the network connectivity degree may change from time to time. The optimal value of the number of layers is different for different connectivities, but we do not suggest to adaptively changing the number of layers dynamically and prefer to keep a relatively small value for three reasons. First, a relatively small value is preferred in order to balance the trade-off between the query traffic and success rate. Second, it is costly and time consuming to measure the connectivity change in a large-scale and dynamic P2P network. Third, it incurs large overhead to frequently update peers with the latest optimal number of layers and cached contents.

6.5 Cache Update

We evaluate the performance of cache update schemes: one more step forwarding and backtrack clearing described in Section 4.4. We compare these two proposed cache update schemes with the original DiCAS which has no cache update. We also compare them with a hybrid scheme of one more step forwarding and backtrack clearing. As shown in Fig. 16, the cache update can improve success rate by about 10 percent, while the overhead of network traffic incurred by extra operations is trivial, shown in Fig. 17. Fig. 17 also shows that the hybrid scheme outperforms the two aforementioned schemes in terms of the costs of network traffic.

6.6 The Final Comparison

The final performance comparison between the flooding (No Cache), UIC, and Enhanced DiCAS (with *push & select with TTL-Kept*) is shown in Fig. 18. We repeat the queries starting from random chosen peers for 50,000 times, and calculate the average of each metric every 1,000 queries. Compared with the flooding approach, the enhanced DiCAS algorithm can reduce the network traffic by an order of magnitude, decrease the response time by 25 percent, while keeping almost the same success rate (slightly lower than flooding). The comparison also shows that the success rate and response time of the enhanced DiCAS gradually approach to those of UIC as the number of queries increases. The noticeable warming up period of the enhanced DiCAS algorithm results from the partition of peers belonging to the same group. When the peers are kept being queried, the cache results will be carried across peers by the query responses, such that the popular cached indices will be uniformly distributed among peers.

7 Conclusions

In this paper, we propose the DiCAS algorithm in the Gnutella-like peer-to-peer network, which can significantly

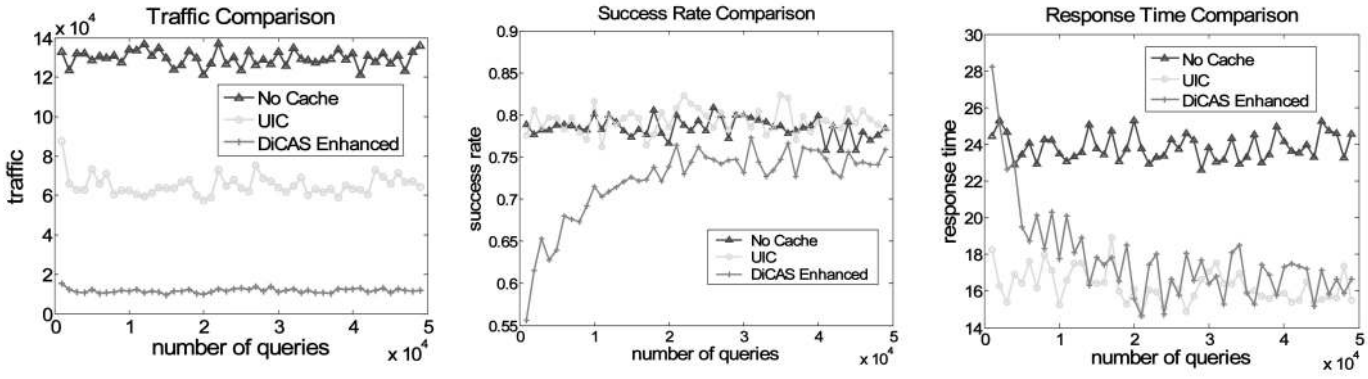

Fig. 18. Comparison between Flooding and DiCAS.

reduce the searching traffic by distributing the index cache among neighboring peers and dividing the searching space into multiple layers. Compared with previously uniform cache mechanism, Distributed Cache can reduce the cache redundancy between neighboring peers and improve the utilization of cache space. Therefore, the performance improvement can be achieved even in the case where each individual peer contributes a small size cache space. This is common in the P2P network community since it is formed by spontaneous users who are not willing to share large size memory as public cache space. Based on the Distributed Cache, the Adaptive Search only forwards a query to a group of peers such that the query traffic is reduced due to shrunk searching space.

It is notable that all the performance improvements above can be achieved at low cost, which contrasts to the high maintenance cost of DHT-based solutions. In a DHT P2P network, the index of a file is stored in a peer whose ID is mapped to the file ID by a predefined Hash function. Therefore, the query for that file can be directly forwarded to the desired peer based on the same Hash function. However, in a highly dynamic P2P network, the cost of frequently index updates may compromise the performance improvement brought by DHT approaches. Different from DHT approaches, DiCAS builds the indices of a file to a group of peers through passive caching, which does not require massive communication between peers hosting indices and peers hosting files to maintain the correct mapping in dynamic cases. The extra cost of the DiCAS algorithm is the memory caching space contributed by each individual peer. As a conclusion, the DiCAS algorithm is a feasible solution to improve P2P network search efficiency at low cost.

Our simulation results demonstrate its strong effectiveness under different conditions. We have also shown that deploying such a caching scheme in an existing P2P network, such as Gnutella, is feasible with an immediate favorable impact on P2P search performance, thus making unstructured P2P systems more scalable. We are refining a prototype version of the Gnutella-based DiCAS for public release in the P2P community.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation under grants CCF-0325760, CCF-0514078, and CNS-0549006, and by Hong Kong RGC Grant HKUST 6152/06E. Some preliminary results of this work were presented in the Proceedings of ICDCS 2004.

REFERENCES

[1] Gnutella Network Size, http://www.limewire.com/index.jsp/size, 2003.
[2] The Gnutella Protocol Specification 0.6, 2002, http://rfc-gnutella.sourceforge.net.
[3] Limewire, http://www.limewire.com, 2003.
[4] V. Almeida, A. Bestavros, M. Crovella, and A.d. Olivera, "Characterizing Reference Locality in the WWW," *Proc. IEEE Conf. Parallel and Distributed Information Systems (PDIS),* 1996.
[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. INFOCOM,* 1999.
[6] BRITE, http://www.cs.bu.edu/brite/, 2003.
[7] B.F. Cooper and H. Garcia-Molina, "Studying Search Networks with SIL," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2003.
[8] A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems," *Proc. 28th Conf. Distributed Computing Systems,* 2002.
[9] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. SIGCOMM,* 1999.
[10] B.J. Jansen, A. Spink, J. Bateman, and T. Saracevic, "Real Life Information Retrieval: A Study of User Queries on the Web," *Proc. SIGIR Forum,* vol. 32, no. 1, pp. 5-17, 1998.
[11] S. Joseph, "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks," *Proc. Int'l Workshop Peer-to-Peer Computing (colocated with Networking 2002),* 2002.
[12] C. Lindemann and O.P. Waldhorst, "A Distributed Search Service for Peer-to-Peer File Sharing in Mobile Applications," *Proc. Int'l Workshop Peer-to-Peer Computing,* 2002.
[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] 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.
[15] D.A. Menasce and L. Kanchanapalli, "Probabilistic Scalable P2P Resource Location Services," *Proc. ACM SIGMETRICS Performance Evaluation Rev.,* vol. 30, no. 2, pp. 48-58, 2002.
[16] C.R. Palmer and J.G. Steffan, "Generating Network Topologies that Obey Power Laws," *Proc. IEEE Globecom,* 2000.
[17] 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.
[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM,* 2001.
[19] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella Network," *IEEE Internet Computing,* 2002.
[20] J. Ritter, "Why Gnutella Can't Scale, No, Really," http://www.tch.org/gnutella.html, 2001.
[21] "Controlled Update Propagation in Peer-to-Peer Networks," *Proc. 2003 USENIX Ann. Technical Conf.,* M. Roussopoulos and M. Baker, eds., 2003.
[22] M. Roussopoulos and M. Baker, "Practical Load Balancing for Content Requests in Peer-to-Peer Networks," Technical Report cs.NI/0209023, Stanford Univ., 2003.
[23] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. Int'l Conf. Distributed Systems Platforms,* 2001.
[24] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking (MMCN),* 2002.
[25] S. Saroiu, K.P. Gummadi, R.J. Dunn, S.D. Gribble, and H.M. Levy, "An Analysis of Internet Content Delivery Systems," *Proc. Fifth Symp. Operating Systems Design and Implementation,* 2002.
[26] M.T. Schlosser and S.D. Kamvar, "Availability and Locality Measurements of Peer-to-Peer File Systems," *Proc. ITCom Conf.: Scalability and Traffic Control in IP Networks,* 2002.
[27] S. Sen and J. Wang, "Analyzing Peer-to-Peer Traffic across Large Networks," *Proc. ACM SIGCOMM Internet Measurement Workshop,* 2002.
[28] K. Sripanidkulchai, "The Popularity of Gnutella Queries and Its Implications on Scalability," http://www2.cs.cmu.edu/kunwadee/research/p2p/gnutella.html, 2001.
[29] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proc. INFOCOM,* 2003.
[30] 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.
[31] Y. Xie and D. O'Hallaron, "Locality in Search Engine Queries and Its Implications for Caching," *Proc. INFOCOM,* 2002.
[32] B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-Peer Networks," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.
[33] Y.B. Zhao, J.D. Kubiatowicz, and A.D. Joseph, "Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing," Technical Report UCB//CSD-01-1141, Univ. of Calif. Berkeley, 2001.

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.

Yunhao Liu received the BS degree in automation from Tsinghua University, China, in 1995, the MA degree from the Beijing Foreign Studies University, China, in 1997, and the PhD degree in computer science from Michigan State University in 2004. He is now an assistant professor of computer science at the Hong Kong University of Science and Technology. His research interests are in the areas of peer-to-peer computing, pervasive computing, distributed systems, network security, grid computing, and high-speed networking. He is a member of the IEEE and the IEEE Computer Society.

Pei Zheng received the PhD degree in computer science from Michigan State University. Before joining Microsoft, he was an assistant professor of computer science at Arcadia University from 2003 to 2005. His research interests include distributed systems, network simulation/emulation, and mobile computing. He is a member of the ACM, the IEEE, and the IEEE Computer Society.