10-2006

Exploiting Geographical and Temporal Locality to Boost Search Efficiency in Peer-to-Peer Systems

Hailong Cai
University of Nebraska-Lincoln, hcai@cse.unl.edu

Jun Wang
University of Central Florida, juwang@mail.ucf.edu

Exploiting Geographical and Temporal Locality to Boost Search Efficiency in Peer-to-Peer Systems

Hailong Cai and Jun Wang, *Member*, *IEEE*

Abstract—As a hot research topic, many search algorithms have been presented and studied for unstructured peer-to-peer (P2P) systems during the past few years. Unfortunately, current approaches either cannot yield good lookup performance, or incur high search cost and system maintenance overhead. The poor search efficiency of these approaches may seriously limit the scalability of current unstructured P2P systems. In this paper, we propose to exploit two-dimensional locality to improve P2P system search efficiency. We present a locality-aware P2P system architecture called *Foreseer*, which explicitly exploits *geographical* locality and *temporal* locality by constructing a *neighbor* overlay and a *friend* overlay, respectively. Each peer in Foreseer maintains a small number of neighbors and friends along with their content filters used as distributed indices. By combining the advantages of distributed indices and the utilization of two-dimensional locality, our scheme significantly boosts P2P search efficiency while introducing only modest overhead. In addition, several alternative forwarding policies of Foreseer search algorithm are studied in depth on how to fully exploit the two-dimensional locality.

Index Terms—Foreseer, unstructured peer-to-peer systems, geographical locality, temporal locality, search efficiency.

1 Introduction

Unstructured peer-to-peer (P2P) systems are creating a large portion of network traffic in today's Internet due to their good support for content lookup and sharing. A P2P system typically involves thousands or millions of live peers in the network. Providing object location service in such a large-scale system requires an efficient search technique to lookup contents shared by individual peers. A good search scheme has to meet two goals: 1) high performance, which tries to deliver high quality service (high success rate and low response latency) to the end users, and 2) low cost, which reserves system resources to sustain a large number of users. A combination of both factors consequently motivates us to formally introduce the notion of *search efficiency*.

Webster dictionary interprets efficiency in computing domain as the extent to which software performs its intended functions with a minimum consumption of computing resources. Applying this definition in P2P systems, we can define *search efficiency* as the ratio of search performance to search cost for a given scheme: $E = P/C$, where E, P, and C stands for search efficiency, performance, and cost, respectively. Previous work defines or measures P2P search efficiency with a focus on search cost while the performance is either ignored or considered separately [14], [35]. In this paper, we evaluate the efficiency of a search scheme by taking both factors into account. As a result, the search efficiency is able to directly tell how fast and accurately a scheme locates desired contents involving minimum system resource consumption.

Although a lot of search schemes have been presented and studied during the past few years, they have limitations on addressing the search efficiency problem. Existing search solutions could be either *blind* or *informed* [33]. Blind search is based on query flooding or random walks, in which no content indexing information is used [1], [5], [15], [24]. As a result, a peer may have to try repeated walks due to previous walk failures or meaningless walks toward free-riding peers. This is because the peers do not know what contents, if at all, are shared on their neighbors before the query is actually transmitted. This kind of blindness in content lookup usually leads to substantial search cost and low search efficiency. A straightforward solution to resolve the blindness without using a centralized index server is to maintain distributed indices among peers. Intelligent BFS [20], APS [32], Local Indices [36], and Routing Indices [8] are examples of this class. In order to effectively direct searches and obtain an acceptable hit rate, however, the indices to be maintained would be extraordinarily large and, hence, the overhead involved in indices update may become prohibitively expensive, thus partially offsetting the benefits of the indices themselves. We have to seek a new way to resolve the above problems.

Previous studies [13], [18] have shown that current P2P systems have both *geographical* locality and *temporal* locality. For node A, its physically nearby node B exhibits geographical locality if it is likely to offer *quality* service to node A in the near future. The rationale is that, queries

- *H. Cai is with the Department of Computer Science and Engineering, University of Nebraska at Lincoln, Lincoln, NE 68588. E-mail: hcai@cse.unl.edu.*
- *J. Wang is with the School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816. E-mail: juwang@mail.ucf.edu.*

Manuscript received 3 Jan. 2005; revised 7 Sept. 2005; accepted 6 Dec. 2005; published online 24 Aug. 2006.
Recommended for acceptance by P. McKinley.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0002-0105.

served by proximate peers are likely to show better service quality (such as small hop number and short response time) than remote peers. Fessant et al. [13] found that peers requesting an object may mostly get it from peers in their own country, thus achieving low latency and network usage. On the other hand, node C, which has successfully served requests from node A in the past, exhibits temporal locality if it is likely to be able to offer further service to node A in the near future. This locality has been extensively studied and exploited in previous research work [7], [9], [29], [34]. From a system designer's point of view, the temporal locality implies good directions of query propagation while the geographical locality indicates a preference to nearby peers (instead of remote ones) as long as they can answer the query. It remains a challenge how to simultaneously consider both locality properties in overlay construction and search algorithm design, such that the search efficiency can be further improved.

In this paper, we attempt to improve search efficiency in decentralized unstructured P2P systems by exploiting two-dimensional locality and distributed indices. Our major contributions are as follows:

1. We develop a locality-aware P2P system architecture called *Foreseer*, which novelly constructs two orthogonal overlays: a *neighbor overlay* based on geographical locality, and a *friend overlay* based on temporal locality.
2. We propose an efficient, locality-aware search algorithm that is performed in two phases (*local matching* and *selective dispatching*) for query resolution and forwarding.
3. Based on different properties of the neighbor and friend overlays, we develop several advanced forwarding policies that can be implemented in our Foreseer search algorithm. We carefully study and analyze these policies and validate them through comprehensive experiments.

Trace-driven simulation results show that Foreseer can improve the search performance by up to 63 percent and significantly reduce the search cost by more than 91 percent, compared with other baseline search schemes. By both improving the search performance and reducing the search cost, Foreseer boosts the search efficiency by a factor of up to 35 while introducing only modest system maintenance overhead.

2 Related Work

We review several representative search schemes in unstructured, decentralized P2P system architectures in this section. Search mechanisms in DHTs are also discussed.

2.1 Blind Searches

Without content indexing information, blind searches propagate a query to a sufficient number of nodes to satisfy a request. The early version of Gnutella [17] uses flooding, which often incurs a lot of query messages and limits its scalability. One possible solution is to reduce the number of redundant messages by forwarding queries only to a subset of neighbors. The neighbors are either randomly picked or selectively chosen based on their capability of answering a query. Lv et al. [24] suggest random walk, in which a query is forwarded to a randomly chosen neighbor at each step until there are sufficient responses. Adamic et al. [1] recommend that the search algorithm bias its walks toward high-degree nodes. GIA, designed by Chawathe et al. [5], exploits the heterogeneity of the network and employs a search protocol that biases walks toward high-capacity nodes. Unfortunately, in these approaches, a query search may require multiple walks due to previous walk failures or undergo meaningless walks toward free-riders. Recently, Gkantsidis et al. [16] propose a generalized search scheme that uses a given budget to limit the total number of messages produced in a search. Although this scheme is able to achieve search performance of flooding with a limited bandwidth consumption, its performance largely depends on the measurement of the criticality of the overlay connections and the allowed search cost.

2.2 Informed Searches

In contrast to blind searches, informed searches maintain and utilize content information to guide the query propagation directions. Unlike Napster-like systems that keep all indexing information on centralized servers, most recent systems distribute the content information among participating peers to avoid single point of failures. Intelligent BFS [20] maintains query-neighbor tuples on each peer. These tuples map classes of queries to neighbors who have answered most of the queries that are related. This technique tries to reuse paths that were used for previous queries of the same class, but cannot be easily adapted to object deletion and node departures. In addition, its search accuracy highly depends on the assumption that nodes specialize in certain documents. In APS [32], each node keeps a local index of the relative probability for each object it requests per neighbor. This approach saves bandwidth, but may suffer long delays if the walks fail.

Local Indices, proposed by Yang et al. [36], suggests each node maintain the content indices of other nodes within a certain radius r, and queries are answered on behalf of all of them. If r is small, however, the indices cannot satisfy many queries; whereas if r is big, the indices update will be very expensive. This approach introduces a simple way to implement indices in unstructured P2P systems, but does not consider any kind of localities. In Routing Indices [8], each node stores an approximate number of documents from every category that can be retrieved through each outgoing link. This technique can be efficient for searches, but it also requires appropriate document categorization and a relatively high content replication ratio. PlanetP, proposed by Cuenca-Acuna et al. [10], summarizes the contents on each peer using Bloom filters, and disseminates new information to the entire system by gossiping.

2.3 Searches on DHTs

Built on top of structured overlays, Distributed Hash Tables (DHTs) are another alternative for keyword searches. They are different from the aforementioned schemes in the strategy of document distribution. In unstructured systems, a document can be placed on any peer and each node maintains the metadata of its own document set. In order to

support complex searches, DHTs distribute and locate contents according to a partition that maps each keyword to peers who are responsible for it. As a consequence, nodes in unstructured overlays are able to handle queries locally while in DHTs the results set intersection may consume some network bandwidth for multiterm searches.

Li et al. [21] present several optimizations for searching in DHTs, such as caching, Bloom filters and document clustering. Reynolds and Vahdat [25] adopt database techniques and Bloom filters to make the results join more efficient. Recently, Tang et al. [30] propose a hybrid indexing structure that processes multiterm queries locally at the cost of maintenance and storage. Similarly, Shi et al. [28] develop another index partitioning scheme, the multilevel partitioning to improve keyword search performance and efficiency.

Besides DHTs and unstructured overlays, researchers also propose hybrid P2P systems that attempt to combine the advantages and avoid the problems of both approaches. Loo et al. [23] present a hybrid search infrastructure, in which popular items are located using traditional flooding while rare items are searched using a DHT query. Based on random walks, hybrid overlay structure (HONet) [31] organizes peers into structured clusters with network proximity and creates connections between clusters through random walks. HONet extends DHT routing to support hierarchical structure and uses intercluster connections to expedite the routing process.

2.4 Searches with Semantic Locality

Semantic clustering has been studied and successfully exploited in a lot of research work. Semantic overlay networks (SON), proposed by Crespo et al. [9], organizes nodes into semantic groups according to a predefined content classification. A query is first sent to an appropriate SON and then floods only to other peers in that SON. Cohen et al. [7] use guide-rules to organize nodes that bear semantic similarity into an associative network. Queries are guided using possession-rules until they are satisfied or resource limits exceeded. The performance of these schemes largely depends on the quality of content classification or guide-rules which, on one hand, require some stability for performance gains, and on the other hand, should be able to adapt to changes in peers preference. This limitation stems from the tight couple of system design with the semantic classification which tends to change over time.

In other schemes, semantic locality is only exploited as an incremental improvement to traditional flooding or random walk mechanisms. For example, Sripanidkulchai et al. [29] present interest-based shortcuts that are generated and updated after each successful query, and used to serve future requests. The Acquaintance, proposed by Cholvi et al. [6], takes a similar approach. Voulgaris et al. [34] study three alternative strategies of maintaining semantic links, two of which are further evaluated in another paper [19]. Unlike SONs and associative search, this kind of scheme does not rely on content classification. Instead, the semantic links are created based on previous query hits, which implicitly capture the temporal locality. In these schemes, however, semantic links are only used as possible routing shortcuts, and in case they fail, the search has to resort to traditional search methods such as flooding or random walks.

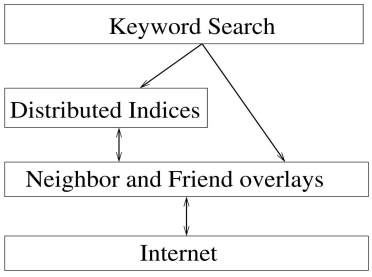

Fig. 1. System architecture of Foreseer built on top of the Internet infrastructure.

3 Foreseer System Architecture

Foreseer is comprised of three components at different layers, as shown in Fig. 1. The neighbor and friend overlays are built on top of the Internet by exploiting geographical and temporal locality, respectively. The indices implemented by Bloom filters are distributed according to the relationships between peers within the two overlays. By directing searches along the overlay links and resolving queries by the distributed indices, Foreseer sustains a high search efficiency for keyword searches with a low maintenance cost.

The rationale behind Foreseer comes from real life. Everyone has neighbors who live nearby and friends who live further away. Neighbors and friends constitute one's social connections. One gets to know his neighbors upon settlement, and makes friends when doing business with someone else. Consequently, friends and neighbors show different characteristics: friends are able to serve future requests with a high probability (temporal locality), while neighbors can offer quality service such as quick response and low resource consumption if they happen to possess the requested objects (geographical locality).

Suppose each person has a business card and knows about others only through their business cards. Upon receiving a new business request, one wants to get it solved efficiently. Thus, he first looks at his business cards of his friends and neighbors. If these cards imply that a friend and/or neighbor can help, he immediately contacts that person. If none of them can help, he passes the request to his friends and/or neighbors who, in turn, seek help from their friends and neighbors. The design of this approach is based on two facts: 1) the well-maintained relationship between peers indicates the most promising directions for future searches and 2) the business card enables one to predict whether its neighbors or friends can help or not before the request is transmitted.

3.1 Content Filters

In Foreseer, the business card refers to a peer's content filter derived by computing the Bloom filter [2] on all its shared contents. A Bloom filter is a hash-based data structure that efficiently represents a set to support membership queries. The membership test returns false positives with a predictable probability, but it never returns false negatives.

Given an optimal choice of hash functions, we can obtain the minimum probability of false positive as $(\frac{1}{2})^k$, or $(0.6185)^{\frac{m}{n}}$, where k is the number of hash functions used, m is the number of bits in the filter, and n is the number of elements in the set. In this paper, we use Counting Bloom filters proposed by Fan at al. [12] to summarize the contents shared on each peer.

The content filters can be created in a straightforward way on each peer. Assume D_p is the set of documents shared on node p, and $K_p = \{kw | kw$ appears in $d_i, d_i \in D_p\}$ is the set of keywords that appear in any document in D_p. The content filter of node p, denoted by $\mathcal{F}_{content\_p}$, is initialized by hashing all the keywords in K_p and setting the corresponding bits to 1. Clearly, free-riders have a null content filter since they share nothing and K_p is empty. If the number of hash functions in use is fixed, the cardinality of the maximum keyword set K_{max} determines the space requirement for the filter with the least false positive rate as $m = \frac{nk}{\ln 2} = \frac{|K_{max}|k}{\ln 2}$. We believe that the size of the maximum keyword set will not be arbitrarily large for several reasons. First, the number of shared files on most peers is limited. The measurement studies on Gnutella [27] indicate that about 75 percent of the clients share no more than 100 files, and only 7 percent of the peers share more than 1,000 files. The results in [13] show that 68 percent of 37,000 peers share no files at all (free-riders), and most of the remaining clients share relatively few (between 10 and 100) files. Second, the documents on the same peer tend to share common topics. The overlap of semantics among documents on one peer reduces the number of unique keywords to be mapped to the content filter. Third, according to [13], [18], most files shared in current P2P systems are multimedia streams, where only a few unique keywords can be derived from one document. Even with $|K_{max}| = 10,000$ and $k = 8$, the length of the filter with least false positive rate is $m = \frac{10,000 \times 8}{\ln 2} = 114,416$ bits $= 14.4KB$. When transmitted over the network, this filter can be packed into several IP packets. For those peers who share few files and keywords, we use a compressed representation of the filter as a collection of 2-tuples (i, x), which means that the ith bit is set for x times. Only the first number in each tuple (location of a 1 in the filter) is transmitted over the network.

However, as the network size increases, some peers may share so many files that the cardinality of their keyword sets becomes larger than current K_{max}. This problem can be solved in two ways. One solution is to migrate some contents from heavily loaded peers to lightly loaded peers, or increase the length of the filters according to the maximum keyword set K_{max}. The other way is to use Bloom filters with varying size. While this approach releases the constraint on the maximum keyword set, it complicates the system design in other aspects. First of all, the hash functions have to be revised to support variable filter length. We may define a set of universal hash functions $\{h_1, h_2, \ldots, h_k\}$ agreed among all nodes. When mapping or querying any item on a filter $\mathcal{F}$ with length $l(\mathcal{F})$, we have to use a different set of hash functions ranging from 0 to $l(\mathcal{F}) - 1$, for example, by defining them as $\{h'_1, h'_2, \ldots, h'_k\}$, where $h'_i = h_i \bmod l(\mathcal{F})$. Furthermore, a node may have to compute the query filter multiple times using different lengths for a search request. We choose Bloom filters with fixed size in this paper for simplicity.

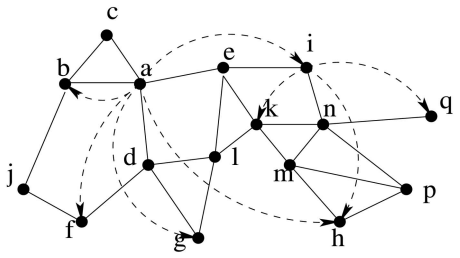

Fig. 2. Illustration of the neighbor and friend overlay. For clarity, this figure only shows the friend links from nodes *a* and *i*.

3.2 Locality-Aware Overlay Construction

Foreseer constructs two orthogonal overlays: 1) the neighbor overlay, which captures geographical locality, and 2) the friend overlay, which captures temporal locality. For a node with a request, its friends have a better chance to serve while its neighbors can provide instant response if they have the contents. In addition to maintaining its own content filter, each node p saves copies of the content filters of the peers in both its neighbors list $N(p)$ and friends list $F(p)$. If a peer becomes a neighbor and a friend at the same time, it is allowed to act as both a neighbor and a friend. To limit the number of filters one peer maintains, we restrict the size of N and F as follows: For node p, $n_{min} \leq |N(p)| \leq n_{max}$, $f_{min} \leq |F(p)| \leq f_{max}$, where n_{min}, f_{min} and n_{max}, f_{max} are the lower bound and upper bound for the number of neighbors and friends, respectively. Fig. 2 illustrates both overlays in a simple network where $N(a) = \{b, c, d, e\}$ and $F(a) = \{b, f, g, h, i\}$.

3.2.1 Finding and Maintaining Neighbors

The bidirectional neighbor overlay is constructed with network proximity, so that only peers that are physically nearby can become each other's neighbors. As the joining node discovers several nearby peers, it tries to make them its neighbors as long as they can accept more neighboring connections. The content filters are transmitted along with the replies so that the new peer can initialize its neighbors list quickly. If the number of its neighbors is smaller than the lower bound, the node issues a PING_NEIGHBORS message to its current neighbors. The current neighbors, in turn, propagate this message to their neighbors. Upon receiving this message, peers with less than the maximum number of neighbors reply positively along with their content filters. This process repeats until the new arrival peer has a minimum number of neighbors.

Geographical locality implies that an object near the querying peer is more likely to be quickly reached than distant objects, thus minimizing network latency and bandwidth consumption. The construction of the neighbor overlay ensures that each peer keeps a list of its nearby peers, and resolves the query locally if the requested object can be found on any of its neighbors.

3.2.2 Making and Refreshing Friends

The friend overlay is constructed as a directed graph independent of the physical network topology. Unlike bidirectional friendships in real life, the friend relationship

in this paper is designed to be unidirectional. Each peer knows a number of friends. Each peer may also be a friend of other peers, who are called its *reciprocal friends* and denoted by F^{-1}. Tracking a reverse direction of the friends relationship, this list is used to notify those reciprocal friends of its content filter updates when necessary. In Fig. 2, $\{a, i\} \subseteq F^{-1}(h)$, and any filter update on node h would cause node a and i to take corresponding action: updating filter copies of node h accordingly.

It is obvious that any peer who has ever answered a request from node p should be a candidate of p's friends, according to temporal locality principles. However, when a brand new peer issues its first query, it has no friends to consult. To mitigate this problem, we recommend an active "friends making" stage for the new node as soon as it builds up its neighbors list. To find potential friends, new node p sends out PING_FRIENDS messages to its neighbors, who in turn forward this message to their friends. Upon receiving this request, peer q checks whether it can be accepted ($|F^{-1}(q)| < f_{max}^{-1}$, where f_{max}^{-1} is the maximum number of a peer's reciprocal friends) or not. Those peers who can accept this "friends making" request will reply to p along with their content filters. Based on these replies, node p can fill out its initial friends list by selecting those peers who have more 1's in their content filters, because the documents shared on these peers contain more keywords.

Node p's friends are ordered and replaced in an LRU manner as new information is learned. After each successful transaction, p has a chance to refresh its friends list. If the serving peer is already one of p's friends, this peer comes to the top of the list because it is the most recently used. If the serving peer is not on p's friends list, and $|F(p)| < f_{max}$, this peer becomes a new friend of p with the highest priority. However, if $|F(p)| = f_{max}$, p has to remove a least recently used friend and insert the new friend as the most recently used. To reduce bandwidth consumption, the content filter of the new friend node can be piggybacked with the downloading traffic toward node p. When an old friend q is replaced, the friendship connection is dropped and node q removes p from its reciprocal list as well.

With these semantic links, Foreseer can easily provide incentives for peers to share contents by limiting the number of friends. As a proof of concept, we can define the maximum number of friends node p can have as $f_{max}(p) = f_0 + k \cdot |F^{-1}(p)|$, where f_0 and k are positive constants and $F^{-1}(p)$ is the number of p's reciprocal friends. Since free-riders share nothing, they have no reciprocal friend (i.e., $F^{-1} = \emptyset$). If f_0 is set to 0, then free-riders will have no friends at all. At current stage, Foreseer does not implement this feature since our concern is mostly focused on the system search efficiency rather than fairness.

3.3 System Maintenance

Due to indices distribution and update, Foreseer brings design complexity and maintenance cost. With this in mind, however, we develop and adopt several techniques to maximize performance gain in search efficiency with minimum overhead. We use Bloom filters to summarize the contents shared on peers. The space efficiency of Bloom filters reduces the bandwidth consumed to transfer the indices. Moreover, the number of friend and neighbor links maintained on each node is limited, so that nodes are not overloaded by the index distribution and updates.

3.3.1 Object Publishing and Removal

When a node is about to share new files, this information should be quickly made visible to its neighbors and reciprocal friend peers. To do this, the peer extracts keywords from new documents, and selects new keywords, if any, to map to its content filter. Any change in the filter is recorded and sent in an update message to all peers in its lists N and F^{-1}. At the same time, the counters associated with corresponding bits are also updated locally on the peer. The process of removing a document is similar to this object publishing procedure.

3.3.2 Node Join and Departure

When a new node joins the system, it needs to set up its neighbor and friend relations as described in Section 3.2. When a node departs, it notifies all its neighbors and reciprocal friends. A total of ($|N| + |F^{-1}|$) small messages are involved per update if necessary. The nodes that receive this notification message simply remove the node from their neighbors or friends lists, along with the corresponding content filter copies. Like the caching scheme used in the Gnutella system, the departing node caches its neighbors and friends on its local disk. When it rejoins the system, it first tries to contact the old neighbors and friends to build up its initial relations quickly. Recent research results [3], [18] show that the node departure-and-rejoin pattern is a common feature of current P2P systems. By this caching scheme, not only is the join process simplified and speeded up, but the workload of bootstrapping nodes is also reduced.

When a node fails unexpectedly, it has no chance to notify other peers of its absence. But, when live nodes try to contact it, they would find this node has already departed. Since each node maintains multiple neighbors and friends, random node failures do not affect the overall system performance.

4 Foreseer Search Algorithm

The most fundamental but challenging task a P2P system has to fulfill is the content lookup and location, mostly triggered by keyword searches. In existing systems, overlay links only function as a logical connection. But, in Foreseer, friend and neighbor links not only connect peers to form a two-dimensional overlay, but also supply good search directions, by which queries can be efficiently answered. This section describes the Foreseer search algorithm and studies several alternative forwarding policies in detail.

4.1 Basic Search Algorithm

Table 1 shows the main process of object lookup algorithm in pseudocode. When initiating a new query request or receiving a query message that contains one or more terms $kw_1, kw_2, \ldots kw_r$, node p runs a search algorithm that consists of two phases: *local matching* and *selective dispatching*. In the local matching phase, node p computes the *query filter* $\mathcal{F}_{query}$ by mapping all the query terms, and compares it with the *content filter* $\mathcal{F}_{content\_q}$ for each node $q \in N(p) \cup F(p)$ by the logical "AND" operation. If $\mathcal{F}_{query} \wedge \mathcal{F}_{content\_q} = \mathcal{F}_{query}$, then there is a match, indicating that node q seems to have the

TABLE 1
Foreseer Search Algorithm in Pseudocode

```
Search (M, P)
//M is the query message, P is the forwarding policy in use
{
    K = extractKeys(M);
    F_query = filter(K);
    for q ∈ N(p) ∪ F(p)
    {
        if F_query ∧ F_content_q = F_query
        {
            result = match(M, q);
            if (result == true) return;
        }
    }

    if (checkTTL(M,P) == false) return;
    X = forwardPeers(P);
    for q ∈ X
        forward(M, q);
}
```

document containing all the keywords with a high probability. Otherwise, none of p's neighbors or friends has the requested document. This matching is conducted on node p locally and consumes no network bandwidth.

The query message is then selectively forwarded based on the result of the first phase. If there is a match, i.e., the query is likely to be answered by one of p's neighbors or friends q, the message is sent to q, which looks up its local folder for the document that matches the query. If a false positive occurs, however, the query is returned to node p. In either situation, whether the local matching fails or a match turns out to be a false positive, the query message needs further forwarding to other peers until matching documents are found or the maximum number of hops are travelled, as accomplished in the selective dispatching phase. Similar to other search approaches, a randomly generated identifier is assigned to each query message and saved on passing peers for a short while so that the same message is not handled by the same peer again. Since this identifier is only useful for a running query session, the timeout could be very small, for example, one or several seconds. The high efficiency of our search algorithm is obtained from three key ideas in our scheme as follows:

1. The peer who issues the query tries to resolve the query *locally* without any network bandwidth consumption. If successful, only one more message is needed to reach the first matching document with a high probability.
2. As soon as the local matching is successful at some nodes, the query is resolved and only one more message is needed for success confirmation in case of no occurrence of false positives.
3. If the local matching fails, the query will be selectively forwarded along the friend links or the neighbor links or both, and conduct local matching at each node encountered. This query dispatching is selective based on different features of the two orthogonal overlays instead of randomly or probabilistically as in other search approaches.

As shown in Fig. 2, where $N(a)=\{b,c,d,e\}$ and $F(a)=\{b,f,g,h,i\}$, a query from node a for objects shared on these nodes can be resolved locally and satisfied in one hop. In other cases, the query needs to be spread out according to the forwarding policies explained in the next section.

4.2 Forwarding Policies

In order to further study different functions of friend links and neighbor links, we introduce the notion of *potential friends* and *potential neighbors* in Foreseer. For a node p, any other peer p_1 is called one of p's potential friends if it can be reached by following friend links from node p for i hops, where i is the distance of this potential friend. Node p's current friends can be considered as a special case of potential friends with a distance 1. Compared to other "strange" peers, p's potential friends have a better chance of answering queries from node p, i.e., they are likely to become node p's friends in the future. For example, suppose $r\in F(q)$ and $q\in F(p)$, documents shared on node r, which tend to interest node q, may also interest node p because p is likely to request more contents from q in the near future according to temporal locality. This locality transitivity stems from interest overlap between the transactions from node r to node q and transactions from node q to node p. However, when there are more nodes on this friend link chain, the degree of interest overlap will be weaken. This implies that as the distance increases, the probability of a potential friend to answer queries from p and become p's new friend decreases.

Similarly, any other peer p_2 is called one of p's potential neighbors if it can be reached by following neighbor links from node p for j hops, where j is the distance of this potential neighbor. Node p's current neighbors can be considered as a special case of potential neighbors with a distance 1. Due to physical proximity maintained by neighbor links, peers on a path containing only neighbor links are potential neighbors. When a peer's current neighbor departs or fails, its potential neighbors may become new neighbors, according to our neighbor overlay construction method. For node p, its potential neighbors with a small distance are nearby and have a good potential to be its neighbors in the future. But, those potential neighbors with a long distance may or may not be nearby to node p, so their potential is indeterministic on the whole.

For a query issued from node p, if it is not locally resolved and answered by one of p's neighbors or friends, it will be spread out along neighbor links or friend links or both. As we discussed above, following friend links approaches potential friends, while following neighbor links approaches potential neighbors. This enriches Foreseer with a lot of flexibilities in realizing selective dispatching in the aforementioned search algorithm. According to different propagation priorities and degree of walking parallelism, we develop the following representative forwarding policies.

4.2.1 Friend First Search

This policy is denoted as *FFS*: $\{F^{h_1}N^{h_2}\}$. For a query issued from a node p_0, this policy firstly propagates the query along friend links for up to h_1 hops, and then along the neighbor links for up to h_2 hops, until the query is answered. Let h be the current hop count of the query message. For node p processing this query, given $h<h_1$, the

query is forwarded to its friends. If $h_1 \leq h < h_1 + h_2$, the query will be forwarded to its neighbors. The query stops travelling when $h = h_1 + h_2$.

Policy FFS searches along the friend links before walking on the neighbor links based on several reasons. First, as we explained, the potential friends have a better chance to answer the query than other "strange" peers. The temporal locality maintained in the friend overlay is further exploited in this way. Second, the construction of the friend overlay implies that the friend links point to peers who share many objects and never refer to free-riders. These peers have a better chance of answering the query than other peers sharing few or no files. Third, the construction of the friend overlay ensures that, by following friend links, the query quickly scatters over a large network diameter and reaches distant peers in few hops. By giving search priority to potential friends, this policy is able to answer many queries in few hops and avoid a lot of meaningless messages involving free-riders. However, the friend overlay may consist of disconnected subgraphs because some unpopular or new documents may not be reachable by following existing friend links. To ensure a high success rate, this policy propagates the query along neighbor links after walking h_1 hops in the friend overlay. At this stage, free-riders may serve as an intermediate router for the query messages.

Let $F^i(p)$ be the set of potential friends of node p with a distance i, and $F^0(p) = \{p\}$. Formally,

$$F^i(p) = \{q | r_0 = p, r_i = q, r_{k+1} \in F(r_k), \quad 0 \leq k \leq i-1\}.$$

If the requested object resides on any peer

$$\underline{p_i \in N(f^i(p)) \cup F(f^i(p))}, \forall f^i(p) \in F^i(p), 0 \leq i \leq h_1,$$

it can be resolved in i hops and reach the destination peer in $i+1$ hops. At each hop, the query touches some new potential friends, and checks the content filters of their neighbors and friends. If the query fails in the friend overlay, it spreads by following the neighbor links. Similarly, we use $N^j(f^{h_1}(p))$ to denote the set of potential neighbors of node $f^{h_1}(p)$ with a distance j, where $f^{h_1}(p) \in F^{h_1}(p)$. Formally,

$$N^j(f^{h_1}(p)) = \{q | r_0 = f^{h_1}(p) \in F^{h_1}(p), r_j = q, r_{k+1} \in N(r_k), \\ 0 \leq k \leq j-1\}.$$

If the requested object resides on any peer

$$\underline{p\prime_j \in N(n^j(f^{h_1}(p))) \cup F(n^j(f^{h_1}(p)))}, \forall n^j(f^{h_1}(p)) \in N^j(f^{h_1}(p)), \\ 1 \leq j \leq h_2,$$

it can be resolved in $h_1 + j$ hops and reach the destination peer in $h_1 + j + 1$ hops.

4.2.2 Neighbor First Search

Similar to FFS, this policy is represented as *NFS*: $\{N^{h_1}F^{h_2}\}$. As the name implies, for a query issued from node p_0, this policy propagates first along neighbor links for up to h_1 hops and then along friend links for up to h_2 hops, until the query is answered.

Let $N^i(p)$ be the set of potential neighbors of node p with a distance i, and $N^0(p) = \{p\}$. Formally,

$$N^i(p) = \{q | r_0 = p, r_i = q, r_{k+1} \in N(r_k), \quad 0 \leq k \leq i-1\}.$$

If the requested object resides on any peer

$$\underline{p_i \in N(n^i(p)) \cup F(n^i(p))}, \forall n^i(p) \in N^i(p), 0 \leq i \leq h_1,$$

it can be resolved in i hops and reach the destination peer in $i+1$ hops. If the query fails in the neighbor overlay, it spreads by following the friend links. Similarly, we use $F^j(n^{h_1}(p))$ to denote the set of potential friends of node $n^{h_1}(p)$ with a distance j, where $n^{h_1}(p) \in N^{h_1}(p)$. Formally,

$$F^j(n^{h_1}(p)) = \{q | r_0 = n^{h_1}(p) \in N^{h_1}(p), r_j = q, r_{k+1} \in F(r_k), \\ 0 \leq k \leq j-1\}.$$

If the requested object resides on any peer

$$\underline{p\prime_j \in N(f^j(n^{h_1}(p))) \cup F(f^j(n^{h_1}(p)))}, \forall f^j(n^{h_1}(p)) \in F^j(n^{h_1}(p)), \\ 1 \leq j \leq h_2,$$

it can be resolved in $h_1 + j$ hops and reach the destination peer in $h_1 + j + 1$ hops.

This policy exploits the temporal locality only locally at hop $h = 0$, and after that it directs queries along neighbor links before walking along friend links. Due to the geographical locality maintained by the neighbor overlay, the query can reach p_0's potential neighbors with a short latency. As a result, queries answered at this stage are likely to be returned with a low response time. However, NFS may *not* be a good forwarding policy because of its low success rate. According to neighbors overlay construction, propagation along neighbor links cannot quickly reach distant peers who may have the requested documents. Furthermore, peers touched by following friend links after traveling along neighbor links are no longer node p_0's potential friends. Hence, following friend links at this stage does not show any advantage over other search algorithms without locality exploitation. We include NFS in our discussion because it helps to understand the algorithm and the differences among alternative forwarding policies.

4.2.3 Concurrent Two-Dimensional Search

We develop another more aggressive forwarding policy denoted as *CTS*: $\{F^{h_1} // N^{h_2}\}$. This policy propagates the query through both friend links (for up to h_1 hops) and neighbor links (for up to h_2 hops) simultaneously. Let h be the current hop count of the query message, which is being processed by a node p. In this policy, given $h < min(h_1, h_2)$, the query is forwarded to both its friends and neighbors. If $min(h_1, h_2) \leq h < max(h_1, h_2)$, the query will be forwarded to either its neighbors (if $h_1 < h_2$) or to its friends (if $h_1 > h_2$). Notice that the initiating peer's potential neighbors and potential friends are subsets of the peers visited. The aggressiveness of this policy contributes to its high success rate and low response time. However, more query messages are generated in this policy for a query since more peers are visited.

Like the other two policies, we use $(F|N)^i(p)$ to denote the set of peers with i hops distance along neighbor or friend links from node p, and let $(F|N)^0(p) = \{p\}$. Formally,

$$(F|N)^i(p) = \{q | r_0 = p, r_i = q, r_{k+1} \in F(r_k) \cup N(r_k), \\ 0 \leq k \leq i-1\}.$$

TABLE 2
The Efficiency of Different Forwarding Policies in the Two-Dimensional Overlay

Policy	Hop (i)	Aggregated delay	Messages	Nodes foreseen
FFS: $\{F^{h_1}N^{h_2}\}$	$i \in [0, h_1]$	$i \times d_f$	f^i	$(f+n)f^i$
	$i \in (h_1, h_1+h_2]$	$h_1 d_f + (i-h_1)d_n$	$n^{i-h_1}f^{h_1}$	$(f+n)n^{i-h_1}f^{h_1}$
NFS: $\{N^{h_1}F^{h_2}\}$	$i \in [0, h_1]$	$i \times d_n$	n^i	$(f+n)n^i$
	$i \in (h_1, h_1+h_2]$	$h_1 d_n + (i-h_1)d_f$	$f^{i-h_1}n^{h_1}$	$(f+n)f^{i-h_1}n^{h_1}$
CTS: $\{F^{h_1}//N^{h_2}\}$ $(h_1 \geq h_2)$	$i \in [0, h_2]$	$i \times d_f$	$(f+n)^i$	$(f+n)^{i+1}$
	$i \in (h_2, h_1]$	$i \times d_f$	$f^{i-h_2}(f+n)^{h_2}$	$f^{i-h_2}(f+n)^{h_2+1}$
CTS: $\{F^{h_1}//N^{h_2}\}$ $(h_1 < h_2)$	$i \in [0, h_1]$	$i \times d_f$	$(f+n)^i$	$(f+n)^{i+1}$
	$i \in (h_1, h_2]$	$h_1 d_f + (i-h_1)d_n$	$d^{i-h_1}(f+n)^{h_1}$	$n^{i-h_1}(f+n)^{h_1+1}$

If the requested object resides on any peer

$$\underline{p_i \in N((f|n)^i(p)) \cup F((f|n)^i(p))}, \forall (f|n)^i(p) \in (F|N)^i(p),$$
$$0 \leq i \leq min(h_1, h_2),$$

it can be resolved in i hops and reach the destination peer in $i+1$ hops. If $h_1 = h_2$, then the query will terminate when the number of hops h is equal to h_1. Otherwise, if $h_1 > h_2$, the query will continue travelling along friend links for up to $h_1 - h_2$ hops unless it is answered. We use $F^j((f|n)^{h_2}(p))$ to denote the set of peers with j hops distance along the friend links from node $(f|n)^{h_2}(p)$, where $(f|n)^{h_2}(p) \in (F|N)^{h_2}(p)$. Formally,

$$F^j((f|n)^{h_2}(p)) = \{q | r_0 = (f|n)^{h_2}(p) \in (F|N)^{h_2}(p),$$
$$r_j = q, r_{k+1} \in F(r_k), \quad 0 \leq k \leq j-1\}.$$

If the requested object resides on any peer

$$\underline{p\prime_j \in N(f^j((f|n)^{h_2}(p))) \cup F(f^j((f|n)^{h_2}(p)))},$$
$$\forall f^j((f|n)^{h_2}(p)) \in F^j((f|n)^{h_2}(p)), 1 \leq j \leq h_1 - h_2,$$

it can be resolved in h_2+j hops and reach the destination peer in h_2+j+1 hops. The case of $h_1 < h_2$ can be analyzed similarly.

Table 2 compares the three forwarding policies, in terms of aggregated delay, the number of messages produced, and the number of peers foreseen at each hop. We use f and n to denote the average number of one peer's friends and neighbors, respectively, and d_f and d_n to denote the average delay by following a friend link and a neighbor link, respectively. According to the overlay construction, friend links usually have a longer latency than neighbor links. Thus, we have $d_f > d_n$, and use d_f to estimate the latency for each hop in CTS policy when $i \leq min(h_1, h_2)$.

5 Experimental Methodology

We develop a trace-driven simulator to evaluate the performance of Foreseer compared with other state-of-the-art P2P systems. We describe the experimental methodology in this section and present the simulation results and our analysis in the next section.

5.1 Experiment Setup

We choose several representative search schemes, such as Flooding (FLD), Interest-Based Shortcuts (IBS) [29], Local Indices (LI) [36], and Routing Indices (RI) [8] as baselines. We configure each scheme according to its default configuration to guarantee a fair comparison, as shown in Table 3. The Foreseer parameters are selected according to the experimental results.

We choose the Transit-Stub model [37] to emulate a physical network topology for all testing systems. This model constructs a hierarchical Internet network with 51,984 physical nodes randomly distributed in an Euclidean coordinate space. We set up nine transit domains, with each containing, on the average, 16 transit nodes. Each transit node has nine stub domains attached. Each stub domain has an average of 40 stub nodes. Nine transit domains at the top level are fully connected, forming a complete graph. Every two transit or stub nodes in a single transit or stub domain are connected with a probability of 0.6 or 0.4, respectively. There is no connection between stub nodes in different stub domains. The network latency is set according to the following rules: 50 ms for intertransit domain links; 20 ms for links between two transit nodes in a transit domain; 5 ms for links from a transit node to a stub node; and 2 ms for links between two stub nodes in a stub domain. We randomly pick peers out of these 51,984 nodes to construct the testing P2P systems in our experiments. Notice that only

TABLE 3
The Baseline and Foreseer System Configuration Parameters

Baselines	Configuration parameters
Flooding (FLD)	TTL=7.
Interest-based Shortcuts (IBS)	TTL=7, size of shortcut list $s = 10$.
Local Indices (LI-1)	TTL=6, index radius $r = 1$, search policy $P = \{0, 3, 6\}$.
Local Indices (LI-2)	TTL=6, index radius $r = 2$, search policy $P = \{0, 3, 6\}$.
Routing Indices (RI)	TTL=6, decay of RI $A = 4$.
Foreseer	TTL=6, $n_{min} = 2$, $n_{max} = 10$, $f_{min} = 4$, $f_{max} = 8$, $f_{max}^{-1} = 20$. Forwarding policies FFS, NFS and CTS.

TABLE 4
The Metrics Used in Our Experiments

Metric	Explanation
Success rate	The percentage of successfully answered queries among all queries for which there exists at least one matching document.
Response time	The average response time to find the first matching document. Since the processing time at a node is negligible compared to the network latency, we ignore the queuing latency and computation overhead of Bloom filter.
Relative distance	The distance actually travelled by consecutive nodes encountered along the route, and then normalized to the shortest distance between the source and the destination node, as defined in [26].
Messages produced	The average number of messages produced while searching an object that matches a query.
Nodes touched	The average number of nodes touched by the query messages during the search.
Free-riders touched	The average number of free-riders touched by the query messages during the search.

some of the physical nodes participate in the P2P system while all nodes contribute to the network latency for messages passing by.

For baseline schemes, we apply the crawled topology data of Gnutella network downloaded from the Limewire Web site [22] to set up the logical network connections. The average node degree of this topology is 3.35. Foreseer builds its own neighbor and friend overlays as described in Section 3.2. It is notable that the average node degree of Foreseer is greater than in other schemes. As the experimental results show, however, Foreseer produces much less search messages than other approaches. In each run of the trace replaying, we randomly select 5 percent nodes to depart and 5 percent nodes to fail on the fly to emulate dynamic activities in P2P systems.

We measure the search efficiency using a variety of metrics shown in Table 4. The first three metrics demonstrate how well a system conducts searches for a given query (search performance), while the last three metrics indicate the bandwidth consumed in finding the first matching document (search cost). For a failed query, the response time is set to the predefined response timeout, 1,000 ms. This is because the system has to wait for the timeout before the query failure is acknowledged. Besides these metrics, we also compare the indices maintenance overhead involved in both LI schemes and Foreseer because their work in updating indices affects the entire system performance and its scalability.

5.2 Trace Preparation

Because there is no real-world trace publicly accessible that contains keyword query and download history information needed in our experiments, we carefully rebuild such a trace by processing a content distribution trace of an eDonkey [11] system obtained from [13]. The eDonkey trace, probed during the first week of November 2003, contains the names of 923,000 files shared among 37,000 peers. More analysis of this trace, such as file popularity distribution, can be seen in [4]. This trace only contains a snapshot of the system, but we need a query trace. To reasonably reconstruct a keyword query trace, we conduct the following preprocessing:

1. First, we set the current trace (a file distribution snapshot) as the system final state S_{final}, and create an initial system snapshot S_{init} containing less documents. To do this, we construct a set of all the file items with more than one copies in S_{final}. And, from this set, we randomly pick 60 percent items and assume that one copy of each item is downloaded from another peer. By removing these items from S_{final}, we have the initial system state S_{init}. Since only one copy is removed for the selected items, the resultant initial system state keeps similar properties, such as popularity distribution, with the observed system snapshot S_{final}.
2. By comparing the system initial state S_{init} and final state S_{final}, we can create a document requesting trace in terms of target document.
3. Then, we conduct a simple lexical analysis and extract keywords from each document by converting its file name to a stream of words. We then calculate the total number of occurrence per keyword. The less occurrence, the higher weight a keyword has.
4. When the above jobs are completed, we transform the document requesting trace into a keyword query trace. To do this, we choose several query terms from keywords that have relatively high weights out of the requested document in each event, since these terms are more effective to reflect the search reality than those with light weights.
5. When the keyword query trace is restored, we feed it into each testing system, replay the queries, and collect the results.

6 Experimental Results

We conduct comprehensive experiments to evaluate the search efficiency of Foreseer compared with other approaches. First, we compare the Foreseer forwarding policies and study the different functions of neighbor links and friend links at each hop. Second, we pick up one instance from each forwarding policy, replay the query trace to test search efficiency of Foreseer and other baseline systems in terms of search performance and cost. After that, we examine the system maintenance of Foreseer versus LI schemes as other schemes either do not use indices, or trigger indices update in a different manner [8]. We finally conduct sensitivity study of the system parameters to investigate their impacts.

6.1 Forwarding Policies

To compare forwarding policies of Foreseer search algorithm, we randomly choose 20,000 peers, and run the query

TABLE 5
Performance and Cost of Different Forwarding Policies of Foreseer

Policy	Success rate	Average hops	Average response time (ms)	Relative distance	Number of query messages	Number of nodes touched	Number of free-riders touched	Relative search efficiency
$FFS:\{F^1N^5\}$	9.88%	3.90	917	2.16	38	32	18	16.97
$FFS:\{F^2N^4\}$	39.03%	3.95	694	2.99	138	119	64	24.39
$FFS:\{F^3N^3\}$	82.16%	3.95	401	3.70	366	335	151	33.50
$FFS:\{F^4N^2\}$	98.53%	3.91	327	4.31	530	485	118	34.02
$FFS:\{F^5N^1\}$	99.58%	3.90	334	4.51	608	500	25	29.35
$FFS:\{F^6N^0\}$	99.07%	3.90	338	4.52	621	490	0	28.24
$NFS:\{N^1F^5\}$	97.12%	4.52	339	4.34	559	464	1	30.67
$NFS:\{N^2F^4\}$	86.28%	4.97	389	4.02	378	336	3	35.11
$NFS:\{N^3F^3\}$	46.92%	5.29	650	3.41	151	144	5	28.61
$NFS:\{N^4F^2\}$	12.29%	4.96	899	2.41	45	42	7	18.18
$NFS:\{N^5F^1\}$	3.43%	4.44	970	1.53	23	21	9	9.20
$NFS:\{N^6F^0\}$	2.52%	3.48	977	1.14	13	11	6	11.87
$CTS:\{F^6//N^2\}$	99.31%	3.64	294	3.87	644	506	10	31.39
$CTS:\{F^6//N^4\}$	99.76%	3.52	260	3.49	640	538	112	35.87
$CTS:\{F^6//N^6\}$	100%	3.53	258	3.45	641	553	129	36.19
$CTS:\{F^2//N^6\}$	55.37%	3.77	554	2.59	215	183	101	27.82
$CTS:\{F^4//N^6\}$	98.95%	3.53	261	3.43	635	554	150	35.72
Flooding	98.76%	4.84	427	5.95	13841	8134	5515	1

events on these peers. The results of the forwarding policies with typical h_1 and h_2 values are shown in Table 5. Although not all cases of CTS policy are presented due to space limitation, the trend is shown.

The FFS policy shows an increasing success rate and decreasing response time when it travels more along friend links before turning to the neighbor overlay. This is because more potential friends are visited. However, if the policy only searches in the friend overlay, as in $FFS:\{F^6N^0\}$, the search performance begins to decrease. This is because without walking along neighbor links, the query is not able to reach some unpopular documents that may be isolated from the friend overlay components. By comparing the search performance of policy $FFS:\{F^4N^2\}$, $FFS:\{F^5N^1\}$, and $FFS:\{F^6N^0\}$, we can see the functions of neighbor links in two aspects: 1) they increase the overlay connectivity, namely, improving the search success rate; 2) neighbor links have short delay and can help reduce search response time.

As for policy NFS, when the search propagates more on the neighbor links, the success rate drops sharply since it can not reach distant peers due to physical proximity of neighbor links. Because of low success rate, few friends can be made after successful queries, which further reduces success rate in future queries. With a low success rate, many queries fail and suffer from long response latency. For this policy, only the first two instances, $NFS:\{N^1F^5\}$ and $NFS:\{N^2F^4\}$ show an acceptable success rate.

Compared to FFS, CTS policy shows even better search performance because it tries to exploit both the temporal and geographical locality since the beginning of a search. But, it incurs relatively higher search cost due to a broader search scope at each hop. For the same reason, a larger portion of free-riders are touched during a search in this forwarding policy.

We also present the performance and cost of Flooding when running the query trace, and use them to normalize the search efficiency of Foreseer forwarding policies, as shown in the last column of Table 5. Taking the most important factors into consideration, we measure a search scheme's performance by its success rate and average response time, and measure the cost by the average number of query messages. Thus, the search efficiency is computed as $E = P/C = \frac{\pi}{t \cdot m}$, where π is the success rate, t denotes the average response time, and m stands for the average number of query messages. The relative search efficiency is computed by setting the search efficiency of Flooding as the base unit, and normalizing the results of Foreseer policies. Although the highest efficiency each policy can have is comparable, it is shown that CTS may obtain the highest in the policy $CTS:\{F^6//N^6\}$. Among all the possible instances of each policy, we choose the one with highest search efficiency as the one to be used in the following experiments. Unless explicitly expressed, FFS refers to $FFS:\{F^4N^2\}$, NFS to $NFS:\{N^2F^4\}$, and CTS means $CTS:\{F^6//N^6\}$ in the rest of this paper.

In order to demonstrate the probability of answering a query by a peer's relations at each step, we collect the number of queries resolved by friends and by neighbors, respectively, at hop $h=0,1,\ldots,6$ in each of the three forwarding policies. Notice that $h=0$ indicates a successful local matching at the peer who issues the query.

The results are shown in Figs. 3, 4, and 5. It can be seen that, in each policy, more than 20 percent queries are resolved locally ($h=0$) due to temporal locality in the workloads and well constructed friend relationships between peers. These queries are answered very efficiently with minimum network bandwidth consumption by only one message. As the hop number increases and the search touches more peers in the friend and/or neighbor overlay, both the friend and neighbor links serve an increasing

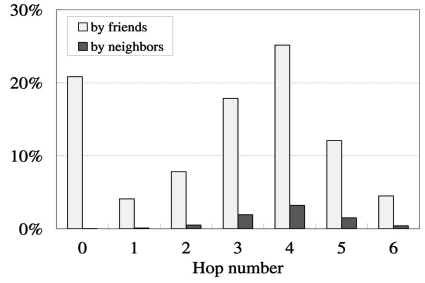

Fig. 3. Distribution of queries resolved by friends and neighbors at each hop, with forwarding policy FFS.

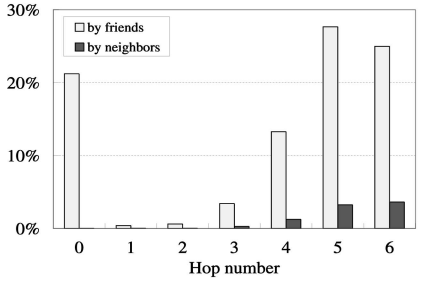

Fig. 4. Distribution of queries resolved by friends and neighbors at each hop, with forwarding policy NFS.

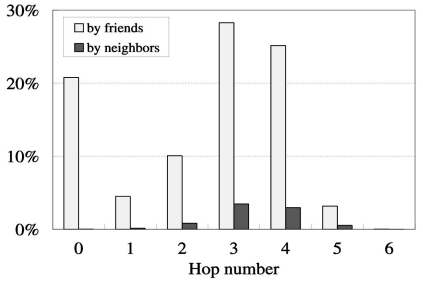

Fig. 5. Distribution of queries resolved by friends and neighbors at each hop, with forwarding policy CTS.

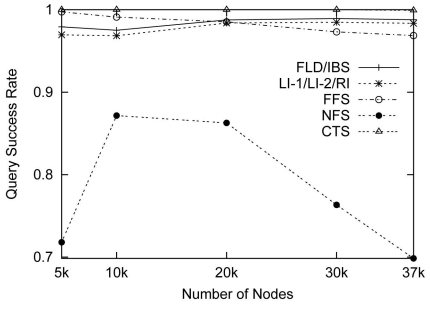

Fig. 6. Comparison of search performance in query success rate.

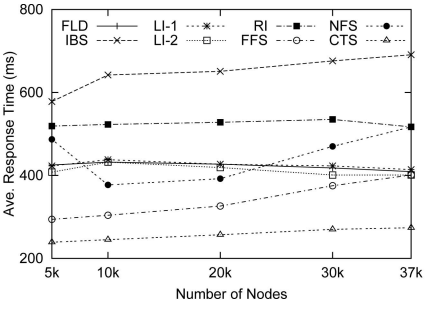

Fig. 7. Comparison of search performance in average response time.

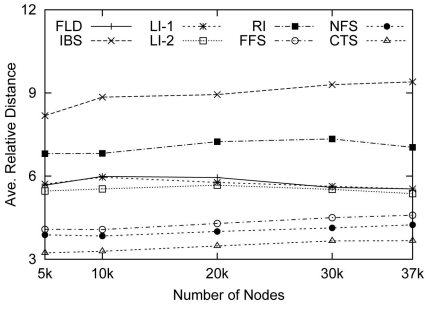

Fig. 8. Comparison of search performance in relative distance.

number of queries until the hop number reaches 3 in CTS, 4 in FFS, or 5 in NFS. When the hop number continues to increase, however, the number of queries answered by either type of links decreases since most of the queries are already satisfied. One exception is that the neighbors at hop 6 serve more queries than neighbors at hop 5 in NFS due to relatively low success rate of this forwarding policy.

Compared to NFS, policies FFS and CTS can serve more queries in fewer hops. This is because both policies look up the initiating peer's potential friends when directing queries, and these potential friends are more likely to answer the queries than other "strange" peers in the overlay.

These figures also show that for each hop number, the friend links serve many more queries than the neighbor links in each policy. One reason is that each peer maintains up to eight friends while the average number of neighbors is only around 2.43, which implies that there are many more friend links existing in the system than neighbor links. Another reason is that a large portion of neighbor links point to free-riders while no friend links do. Although it seems that the neighbor links do not contribute much to the search performance, they also play a role in Foreseer: They increase the success rate by connecting isolated nodes that do not have many friend relationships, and reduce the response delay and relative distance, as shown in Table 5. This is important because the search is often followed by a content download which favors a small relative distance.

6.2 Search Efficiency

In this experiment, we run the query trace with a varying number of peers, and compare the search efficiency of Foreseer algorithm against the baseline systems in terms of search performance and cost.

6.2.1 Search Performance

The performance of each algorithm is tested and compared in terms of query success rate, response time and relative distance as shown in Figs. 6, 7, and 8. Based on similar search methods, Flooding and IBS, both LI schemes and RI have very close success rate, and they are shown together in Fig. 6. We notice that, all the baseline systems have an average success rate around 98 percent, while Foreseer policies FFS and CTS can achieve an even higher success rate of up to 100 percent. Because of uncontrolled data

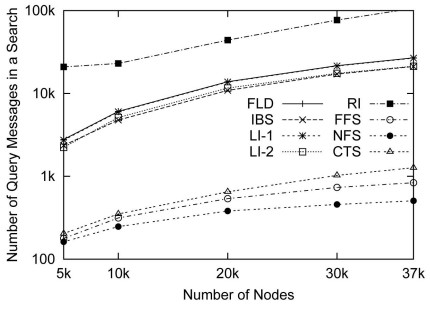

Fig. 9. Comparison of search cost in the number of messages produced in a search.

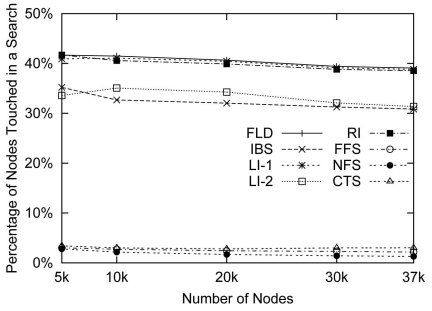

Fig. 10. Comparison of search cost in the percentage of nodes touched in a search.

placement and finite TTL for query messages, no current search algorithms in unstructured P2P systems can guarantee a 100 percent success rate. However, only a negligible percentage of queries may fail in both Foreseer policies, which is quite satisfactory for most users.

As for response time and relative distance, our results show that IBS suffers the longest response time and the largest relative distance although around 20 percent queries are answered by the shortcuts. In IBS, a peer first contacts all of its shortcuts to see if they can answer the query. If no shortcut peer has the requested object, however, the search has already been delayed before the peer floods the query to its neighbors. By maintaining content filter copies, a peer in Foreseer can resolve the query locally on behalf of its neighbors and friends, and does not need to try any of them before fanning out the query. Compared to IBS, Foreseer reduces the average response time and relative distance by up to 63 percent.

Compared to other baseline systems like Flooding, Local Indices and Routing Indices, Foreseer reduces the average response time by up to 53 percent and the average relative distance by up to 52 percent. The benefit stems from Foreseer's ability to exploit both temporal and geographical locality to propagate queries. Following the friend links enables Foreseer to reach the destination peer in few hops while following the neighbor links enables Foreseer to answer the queries with short network latency.

Among the three Foreseer policies, CTS shows the shortest response time and lowest relative distance due to its aggressive searching. NFS has the longest response time since the system has to wait for many unsuccessful queries to time out. But, if the query is successful, it is answered with a short relative distance because of the physical proximity maintained by the neighbor links.

6.2.2 Search Cost

Flooding has poor system scalability because its blind flooding results in a large number of redundant messages and touches too many unrelated peers during the object searches. Routing Indices produces even more query messages since a node may have to receive and forward multiple query messages for the same query session due to its DFS (Depth First Search) search algorithm. Other baseline systems also require a lot of messages if the query is not satisfied by the shortcuts (in IBS) or the local indices (in LI schemes). We collect the total number of query messages and touched nodes in all the queries, and compute the average number of messages produced and the average percentage of touched nodes in a search, as shown in Fig. 9 and Fig. 10, respectively. Compared with IBS, which shows the best results among the baseline systems, all Foreseer forwarding policies can reduce both the number of messages and the percentage of touched nodes by more than 91 percent. In our experiments, Foreseer only touches less than 3 percent live nodes for each query on average. From Fig. 9 and Fig. 10, we can see that both IBS and LI-2 show capabilities of reducing the number of redundant messages and touched nodes. IBS achieves this improvement by exploiting temporal (interest) locality, while LI-2 by maintaining abundant index information. By combining their advantages, Foreseer improves the search performance and simultaneously slashes the search cost.

One of the valuable features of the friend overlay is that no free-riders can become a friend since they share nothing and cannot serve any query. Therefore, the search will never touch free-riders while propagating along friend links. In other systems, however, even a peer knows that some of its neighbors are free-riders (by looking at the indices as in LI schemes), it still sends the query to them when fanning out the query. We conduct the experiments, calculate the percentage of free-riders among nodes touched in a search and depict the results in Fig. 11. More than 67 percent of nodes touched in baseline systems are free-riders, while only less than 34 percent of nodes touched in Foreseer policies could be free-riders. This avoids a lot of meaningless query messages in other systems since free-riders involved in the query cannot serve a query directly.

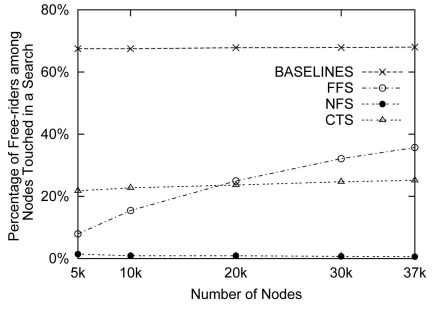

Fig. 11. Comparison of search cost in the free-rider percentage among nodes touched in a search.

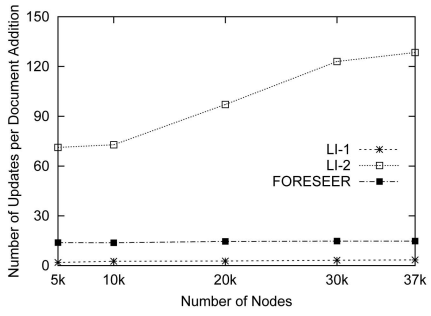

Fig. 12. Comparison of indices maintenance cost in the number of update messages produced for this purpose.

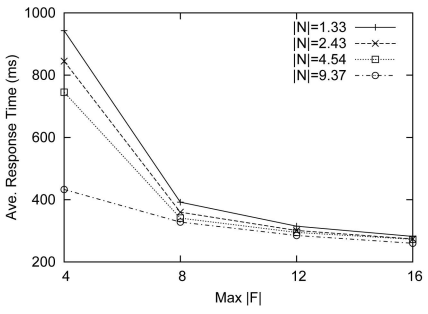

Fig. 14. Sensitivity of search performance to peer's number of neighbors and friends, in terms of response time.

Since NFS policy firstly searches a few potential neighbors, but never comes back to the neighbor overlay, the percentage of free-riders in touched nodes is steadily as low as 2 percent. As the network scale increases, both FFS and CTS policies visit relatively more free-riders when they propagate the query along neighbor links. And, the percentage for FFS increases at a faster speed because when there are more nodes in the system, more free-riders have to be visited to reach other peers that are isolated from the friend overlay.

Based on the above results, we can compute how much improvement on search efficiency Foreseer (policy FFS and CTS) can achieve compared to the baseline systems (using the search efficiency equation $S = P/C$). Compared to Flooding, IBS, LI, and RI schemes, Foreseer on average improves the search efficiency by a factor of around 35, 18, 24, and 38, respectively. Notice that RI does not work well in our experiments, possibly because the documents have a rather low replication ratio, which makes it difficult for the routing indices to spread over the network.

6.3 System Maintenance Costs

When a query is answered, the peer who issued that query has a new document to be published to other peers (we assume this is a requirement). We compare the number of messages used to update indices in LI schemes and Foreseer, as shown in Fig. 12. The data of Foreseer is computed by averaging the results obtained with all the three forwarding policies. It is clear that LI-1 only needs to send several update messages after a query on the average. But, for LI-2, since each peer stores the indices of files shared on all the nodes within radius $r=2$, an index update results in a large number of update messages.

With an average of 13 update messages after each query, Foreseer pays a modest cost for its good search efficiency as seen in the previous sections. Furthermore, by using Bloom filters, the update messages are quite small and do not consume much network bandwidth. For an object addition, a peer only needs to transmit the locations of changed bits in its content filter. Suppose $T=100$ unique keywords are extracted from the document and $k=8$, $m=8KB$ for the Bloom filter implementation. Each changed bit requires $B=2\ bytes$ to specify its location in the filter. The information to be sent out is limited by $L \leq T \times k \times B = 1,600\ bytes$, which can be easily packed in few IP packets.

6.4 Scheme Optimization

We study Foreseer's sensitivity to the number of neighbors and friends by running query events on 20,000 peers with various configuration parameters. Since peers keep making new friends after their queries are satisfied until they have the maximum number of friends, the upper bound of friends (Max $|F|$) approximates the number of friends each node maintains in the system. On the other hand, a peer may have a lower bound number of neighbors and will not look for new neighbors until some of its current neighbors depart. We collect the number of neighbors for each node and compute the average value as $|N|=1.33$ for $n_{min}=1$ and $n_{max}=5$, $|N|=2.43$ for $n_{min}=2$ and $n_{max}=10$, $|N|=4.54$ for $n_{min}=4$ and $n_{max}=20$, and $|N|=9.37$ for $n_{min}=8$ and $n_{max}=40$.

As shown in Figs. 13, 14, 15, and 16, we present the search performance and cost of Foreseer search algorithm in terms of query success rate, response time, number of query

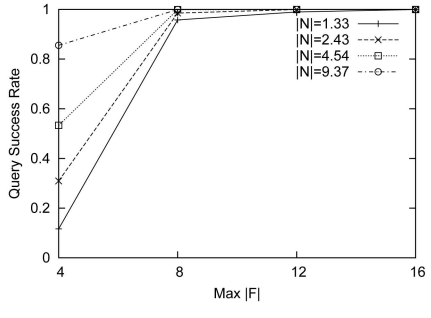

Fig. 13. Sensitivity of search performance to peer's number of neighbors and friends, in terms of query success rate.

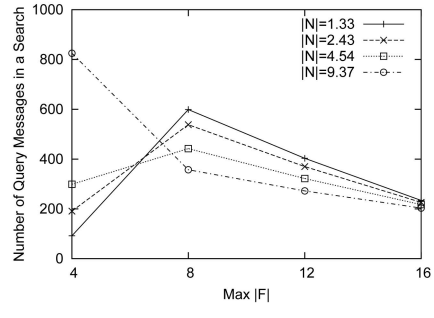

Fig. 15. Sensitivity of search cost to peer's number of neighbors and friends, in the number of query messages produced.

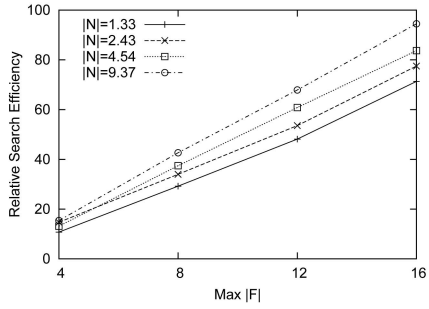

Fig. 16. Sensitivity of search efficiency to peer's number of neighbors and friends, normalized to Flooding scheme.

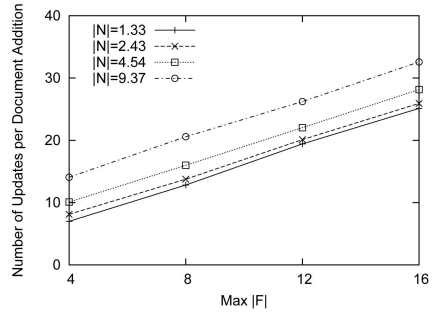

Fig. 17. Sensitivity of indices maintenance cost to peer's number of neighbors and friends, in the number of update messages.

messages, and search efficiency relative to Flooding scheme. Due to space limitation, we only present the results of FFS policy, and other policies show the same trend.

We notice that when $Max|F|=4$, a large portion of queries fail because the query could only reach a small number of nodes due to the upper bound on the number of friends. When $Max|F|>4$, as shown in these figures, the search performance keeps increasing, and the number of query messages produced keeps decreasing as more neighbors and more friends are maintained on peers. And, thus, the search efficiency steadily increases as the peers maintain more relations in the network. However, a larger number of neighbors and friends also results in more indices update workloads, as shown in Fig. 17.

7 Conclusions

In this paper, we propose a new P2P system architecture called Foreseer, which constructs two orthogonal overlays based on geographical and temporal localities and maintains distributed indices for objects shared on peers' neighbors and friends. By selectively directing searches along the friend links and neighbor links, Foreseer is able to achieve an extremely high search efficiency with modest maintenance overhead. Simulation results show that Foreseer can boost the search performance by up to 63 percent, with regard to response time and relative distance, and slash the search cost by more than 91 percent in terms of the number of query messages produced and nodes touched, compared with other state-of-the-art P2P systems. The performance improvement and cost reduction altogether contribute to a high search efficiency of Foreseer search algorithm. We also discuss several alternative forwarding policies in detail and study their efficiency by experiments.

Acknowledgments

This work is supported in part by the US National Science Foundation under grants CNS-0615263, CNS-0621526, CNS-0509480, and CCF-0429995, and by the US Department of Energy Early Career Principal Investigator Award DE-FG02-05ER25687. The authors would like to thank Dong Li, Xiaoyu Yao, and Peng Gu for their valuable feedback. They would also like to thank the anonymous reviewers for their helpful and constructive comments.

References

[1] L.A. Adamic, R.M. Lukose, A.R. Puniyani, and B.A. Huberman, "Search in Power-Law Networks," *Physical Rev. E*, vol. 64, no. 4, pp. 46135-46143, 2001.

[2] B.H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Comm. ACM*, vol. 13, no. 7, pp. 422-426, 1970.

[3] H. Cai and J. Wang, "Caching Routing Indices in Structured P2P Overlays," *Proc. 34th Int'l Conf. Parallel Processing (ICPP '05)*, pp. 521-528, June 2005.

[4] M. Castro, M. Costa, and A. Rowstron, "Debunking Some Myths about Structured and Unstructured Overlays," *Proc. Second Symp. Networked Systems Design and Implementation (NSDI '05)*, May 2005.

[5] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-Like P2P Systems Scalable," *Proc. ACM SIGCOMM '03*, pp. 407-418, Aug. 2003.

[6] V. Cholvi, P. Felber, and E. Biersack, "Efficient Search in Unstructured Peer-to-Peer Networks," *Proc. 16th ACM Symp. Parallelism in Algorithms and Architectures (SPAA '04)*, pp. 271-272, June 2004.

[7] E. Cohen, A. Fiat, and H. Kaplan, "Associative Search in Peer to Peer Networks: Harnessing Latent Semantics," *Proc. IEEE INFOCOM '03*, pp. 1261-1271, Mar. 2003.

[8] A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS)*, pp. 23-34, July 2002.

[9] A. Crespo and H. Garcia-Molina, "Semantic Overlay Networks for P2P Systems," technical report, Stanford Univ., 2003.

[10] F.M. Cuenca-Acuna, C. Peery, R.P. Martin, and T.D. Nguyen, "PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities," *Proc. 12th Int'l Symp. High Performance Distributed Computing (HPDC-12)*, pp. 236-246, June 2003.

[11] eDonkey, http://www.edonkey2000.net/, 2003.

[12] L. Fan, P. Cao, J.M. Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Networking*, vol. 8, no. 3, pp. 281-293, 2000.

[13] F. Le Fessant, S.B. Handurukande, A.-M. Kermarrec, and L. Massoulié, "Clustering in Peer-to-Peer File Sharing Workloads," *Proc. Third Int'l Workshop Peer-to-Peer Systems (IPTPS)*, pp. 217-226, Feb. 2004.

[14] P. Ganesan, Q. Sun, and H. Garcia-Molina, "YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology," *Proc. IEEE INFOCOM '03*, pp. 1250-1260, Mar. 2003.

[15] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," *Proc. IEEE INFOCOM '04*, pp. 120-130, Mar. 2004.

[16] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid Search Schemes for Unstructured Peer-to-Peer Networks," *Proc. IEEE INFOCOM '05*, pp. 1526-1237, Mar. 2005.

[17] Gnutella, 2003, http://rfc-gnutella.sourceforge.net/developer/index.html.

[18] P. Krishna Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP-19)*, pp. 314-329, Oct. 2003.

[19] S. Handurukande, A.-M. Kermarrec, F. Le Fessant, and L. Massoulie, "Exploiting Semantic Clustering in the eDonkey P2P Network," *Proc. 11th ACM SIGOPS European Workshop*, Sept. 2004.

[20] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," *Proc. 11th Int'l Conf. Information and Knowledge Management (CIKM '02),* pp. 300-307, Nov. 2002.
[21] J. Li, B.T. Loo, J.M. Hellerstein, and M.F. Kaashoek, "On the Feasibility of Peer-to-Peer Web Indexing and Search," *Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS),* pp. 207-215, Feb. 2003.
[22] Limewire, http://www.limewire.org, 2006.
[23] B.T. Loo, R. Huebsch, I. Stoica, and J. Hellerstein, "The Case for a Hybrid P2P Search Infrastructure," *Proc. Third Int'l Workshop Peer-to-Peer Systems (IPTPS),* pp. 140-151, Feb. 2004.
[24] 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 (ICS '02),* pp. 84-95, June 2002.
[25] P. Reynolds and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching," *Proc. ACM/IFIP/USENIX Int'l Middleware Conf. (Middleware),* pp. 21-40, June 2003.
[26] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms (Middleware),* pp. 329-350, Nov. 2001.
[27] S. Saroiu, P. Krishna Gummadi, and S.D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking Conf. (MMCN),* Jan. 2002.
[28] S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, and M. Chen, "Making Peer-to-Peer Keyword Searching Feasible Using Multi-Level Partitioning," *Proc. Third Int'l Workshop Peer-to-Peer Systems (IPTPS),* pp. 151-161, Feb. 2004.
[29] K. Sripanidkulchai, B.M. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proc. IEEE INFOCOM '03,* pp. 2166-2176, Mar. 2003.
[30] C. Tang and S. Dwarkadas, "Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval," *Proc. First Symp. Networked Systems Design and Implementation (NSDI '04),* pp. 211-224, Mar. 2004.
[31] R. Tian, Y. Xiong, Q. Zhang, B. Li, B.Y. Zhao, and X. Li, "Hybrid Overlay Structure Based on Random Walks," *Proc. Fourth Int'l Workshop Peer-to-Peer Systems (IPTPS),* Feb. 2005.
[32] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks," *Proc. Third IEEE Int'l Conf. P2P Computing,* pp. 102-109, Sept. 2003.
[33] D. Tsoumakos and N. Roussopoulos, "A Comparison of Peer-to-Peer Search Methods," *Proc. Int'l Workshop Web and Database (WebDB),* pp. 61-66, June 2003.
[34] S. Voulgaris, A.-M. Kermarrec, L. Massoulie, and M. van Steen, "Exploiting Semantic Proximity in Peer-to-Peer Content Searching," *Proc. 10th Int'l Workshop Future Trends in Distributed Computing Systems (FTDCS '04),* pp. 238-243, May 2004.
[35] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed Caching and Adaptive Search in Multilayer P2P Networks," *Proc. 24th Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 219-226, Mar. 2004.
[36] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 5-14, July 2002.
[37] E.W. Zegura, K.L. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," *Proc. IEEE INFOCOM '96,* pp. 594-602, Mar. 1996.

Hailong Cai received the bachelor's degree from the Department of Computer Science and Technology at the Huazhong University of Science and Technology in 1997, and the master's degree from the Institute of Software, Chinese Academy of Science in 2000. He is a PhD candidate in the Computer Science and Engineering Department at the University of Nebraska, Lincoln. His research interests include distributed computing, storage systems, and computer network systems.

Jun Wang received the PhD degree in computer science and engineering from the University of Cincinnati in 2002. He received the BE degree in computer engineering from the Wuhan Technical University of Surveying and Mapping (merged into Wuhan University since 2000) and the ME degree in computer engineering from the Huazhong University of Science and Technology, China. He is an assistant professor in the School of Electrical Engineering and Computer Science at the University of Central Florida. From August 2002 to August 2006, he was an assistant professor in the Computer Science and Engineering Department at the University of Nebraska, Lincoln. His research interests include high-performance and low-power I/O architecture, file and storage systems, parallel and distributed computing, cluster and grid computing, and performance evaluation. He has received several research grants from the US National Science Foundation and the US Department of Energy Early Career Principal Investigator Award program. He is a member of the IEEE, the ACM, USENIX, and SNIA.