Freelib: a peer-to-peer-based digital library architecture*

Amrou A, Maly K, and Zubair M
Department of Computer Science, Old Dominion University
4700 Elkhron Ave, Ste 3300, Norfolk, VA 23529, USA

World Digital Libraries 1(2): 101–120

Abstract

Despite the creation of many digital libraries, there exists considerable untapped content with individuals in diverse communities. One major hindrance to dissemination is the centralized frameworks under which digital libraries are organized and deployed, thus limiting their accessibility, particularly in publishing. We have developed Freelib, a peer-to-peer-based digital library, which is self-sustainable and supports evolving communities with diverse interests. Freelib is built upon the existing work in the areas of OAI (Open Archive Initiative), and peer-to-peer and social networks. The key concept of Freelib is to dynamically form virtual communities based on users' common interest. In order to achieve this, the Freelib client analyses user access patterns and connects itself to other clients, the users of which show similar interest. In this paper, we present architecture of Freelib and report on its performance evaluation. We have built an event-based simulator, which enabled us to simulate networks of thousands of users. We study the performance gain of using Freelib over using small-world peer-to-peer networks alone. The performance evaluation shows that Freelib has significant performance gain over those networks in terms of recall, network bandwidth, and search response time, with minimal cost on the part of the end-user of the system.

* This work is supported in part by NSF grant 033547

Email aamr@cs.odu.edu; maly@cs.odu.edu; zubair@cs.odu.edu

Overview

Digital libraries serve the purpose of disseminating high-quality information through the use of structured and well-defined metadata. They provide efficient search and retrieval services. However, most digital libraries employ a centralized framework and use dedicated resources such as file servers, database servers, and Web servers. The use of centralized framework concentrates traffic and information in and around those servers. On the other hand, peer-to-peer systems utilize resources that are contributed by individual users. This gives peer-to-peer systems desirable criteria such as self-sustainability and high scalability in terms of managing large number of users (resources available grow with the number of users). However, users of peer-to-peer systems often suffer poor search performance due to the prohibiting bandwidth cost and the time needed for search queries to reach relevant peers, who may be far away on the peer-to-peer network. Freelib addresses these issues by providing a framework for building digital libraries on top of peer-to-peer network. It takes advantage of peer-to-peer networks to provide self-sustainability, scalability, and distribution of traffic and information. In addition, Freelib evolves peers into virtual communities, who share common interest to enhance the search performance. Note that these virtual communities are not static and can change with time as users' interest changes. The key benefit of Freelib comes from the fact that the search queries are always targeted to fewer, but more relevant, peers, even in the face of users' changing interest. This results in higher recall, faster searches, and conservation of network bandwidth. The enhanced performance of Freelib comes at minimal cost to the end-user of the system, mainly in terms of more storage requirements, although the complexity of the software has increased considerably.

The Freelib architecture has two overlay networks: the support network and the access network. The support network is based on the symphony protocol (Manku, Bawa, and Raghavan 2003). It maintains the connectivity of the network and enables new nodes to perform searches that are bound by a small-world property symphony networks. The access network is based on a user's access pattern, and it brings nodes sharing similar interest (friends) close to each other (in a virtual sense) and forms virtual community. When a node detects that it has enough friend links, it switches to the friend network for searching. The node then continues to use the friend network for search unless the size of results drops significantly. In this case, the node assumes that the community is not yet evolved and reverts to global search using the support network. Freelib ensures that the friend network always reflects the current user interest by capturing the user interest from patterns of user access to other peers rather than topic segmentation as in SETS (Bawa, Manku, and Raghavan 2003). The process of evolving users into communities of common interest is a dynamic and distributed process. Each node (1) monitors its user access patterns, (2) identifies peers who share same interest, (3) selects some of these peers according to a ranking criteria, and (4) establishes friend links to the selected peers. The process is repeated periodically by each node. This produces an ever evolving access network in which peers who share common interest form virtual communities.

We have presented the Freelib architecture in previous paper (Amrou, Maly, and Zubair 2004). We presented a preliminary evaluation of Freelib in (Amrou, Maly, and Zubair 2006). In this paper and the companion conference paper (Amrou, Maly, and Zubair 2006a), we present a more thorough performance study of Freelib. However, for completeness of the discussions and for the readers' convenience,

we shall briefly overview the Freelib architecture and the preliminary evaluation in this paper as well. In the next section, we explore the background and related work. In the next section, we provide a brief summary of the Freelib architecture, following which we report on the experiments. Finally, we conclude and describe our future work.

Related work

Digital library models span a wide spectrum, ranging from fully centralized architectures (ACM 2008; Alexandria 2008) to federated data providers (Arc 2008; Liu, Maly, Zubair, *et al.* 2001) to distributed ones (McNab, Witten, and Boddie 1998; NZDL 2008). Only recently, some digital libraries (Ding and Solvberg 2004; Walkerdine and Rayson 2004) have started to utilize the peer-to-peer model. Our Freelib architecture utilizes a peer-to-peer model. Hence, we discuss peer-to-peer architecture in more details in the remainder of this section. Some peer-to-peer systems such as Napster utilize a centralized index to perform efficient searches, and file downloads are done in a direct peer-to-peer fashion. Other systems such as Freenet (Clarke, Sandberg, Wiley, *et al.* 2000) and Gnutella use pure peer-to-peer techniques. Freenet sets anonymity as a goal. In order to achieve it, it uses DFS (depth-first search) and results are forwarded following the same path as the query, rather than sending directly to the requesting node. Gnutella, on the other hand, uses BFS (breadth-first search). Drawbacks and challenges faced by these and other peer-to-peer search protocols are described in Daswani, Garcia-Molina, and Yang (2003). DFS search is suitable for retrieval of items given in their identifier. It is not efficient when used for general keyword search. BFS is more suitable for the general keyword search. However, it is bandwidth inefficient, as the number of messages grows exponentially with the TTL (time-to-live) and average node degree (# messages = d^{TTL}, where d is the average degree of the nodes and TTL is the hop limit). Considerable effort has been devoted by researchers to address those peer-to-peer search issues. Some techniques were presented in Yang and Gracia-Molina (2002) to enhance peer-to-peer search performance. For example, iterative deepening was proposed to use the lowest TTL possible for processing each search query. Iterative deepening is done by sending searches with successively increasing TTL till the query is satisfied. Directed DFS is another technique that enhances selectivity of search queries. It tries to target relevant peers by sending search requests to those peers who provided good results in the recent past. A third technique is the use of local indices, in which local indices are built at each node, indexing the content on the peers within r hops. Other peer-to-peer systems utilize caching, replication, and the concept of super-peers to enhance the system performance. Kazaa is a peer-to-peer system that utilizes the concept of super-peers. It utilizes nodes having high network bandwidth to be super-nodes. Those super-nodes cache contents from other nodes and do most of the request forwarding and traffic. Other peer-to-peer protocols such as SETS (Bawa, Manku, and Raghavan 2003) and SSW (Li, Lee, Sivasubramaniam, *et al.* 2004) try to enhance the selectivity of the search query by trying to direct search queries to target peers that have relevant content. These systems cluster member nodes based on the similarity of the content such that nodes containing similar material are connected together. To measure the similarity of documents, these documents are represented in some data structure such as a keyword vector. A clustering algorithm is run on documents to cluster them. This approach is, however, complicated and does not evolve quickly when user's interest changes between topics. These approaches are data-centred and do not adapt with the user

interest as it evolves. For example, a researcher might have most of his publications in biology and at some point of time, start to shift interest to bioinformatics. A system like SETS will still connect such user to peers whose publications are in biology, which hinders the user from quickly discovering and connecting to his new community, that is, bioinformatics.

The Freelib architecture

The Freelib architecture is explained in detail in previous publications (Amrou, Maly, and Zubair 2004, 2006). However, for the reader's convenience, we overview the key parts of the architecture in this section. The Freelib architecture consists of two overlay networks: the symphony network and the access network. The symphony network is a small-world network, which is built based on the symphony protocol (Manku, Bawa, and Raghavan 2003). Small-world networks have the desirable criterion that the network diameter (the maximum number of hops between any two nodes) is small compared to the size of the network (Kleinberg 2000; Watts and Strogatz 1998). For symphony, the network diameter is $(\log^2 n)/K$, where n is the number of nodes (the network size) and K is the number of long contacts per node. The purpose of the symphony network is to provide a way for new nodes to search and discover their communities. New nodes use the symphony layer to submit and forward search queries. When the Freelib client at a node detects that it has enough Friend links, it switches to use the access network for submitting and forwarding all search queries. The access network is built such that nodes that share common interest evolve into virtual communities. Freelib utilizes adaptive and dynamic techniques for evolving these user communities. Each client monitors its user accesses, identifies peers who are of interest to its local user, and connects itself to few of those peers. To identify peers whose interest is similar to a user, the client maintains an access log and periodically performs a ranking process that uses the most recent accesses. The outcome of the ranking process is an ordered list of peers. The client tries to establish friend links to few peers from the ranked list. Every time the ranking process is performed, the friend links are updated to reflect the latest ranked list. The use of the most recent accesses in the ranking process ensures that the user's current interest is always reflected in the results of the ranking. The ranking measures and the details of the ranking process are explained in detail in Amrou, Maly, and Zubair (2004, 2006). The Freelib network architecture is shown in Figure 1.

Peer ranking

The purpose of peer ranking is to provide a ranked list of peers, which serves as a list of candidates for building friend links. For a node N_i, mutual access with other nodes consists of two components, incoming accesses and outgoing accesses. We need to design a ranking function that has the following characteristics and features.

- It increases monotonically as the number of accesses increase.
- It takes into consideration both the incoming and outgoing components of the access data.

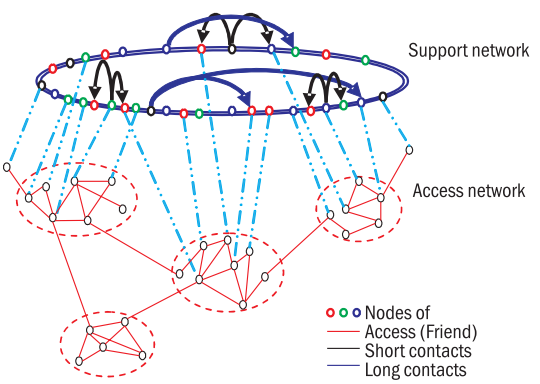

Figure 1 Freelib network architecture

- It allows us to assign different weights for each of the outgoing and incoming accesses.

The requirement that ranks monotonically increase with increasing accesses is clear, since more accesses imply more mutual interest. Including incoming accesses in the calculation is needed to reflect mutual interest rather than one party's interest. Incoming accesses usually are assigned lower weight compared to the weight for outgoing accesses. Equation (1) gives our peer ranking function. It shows the calculation of the rank assigned to node N_j by node N_i.

$$R_i(N_i) = \alpha \times \frac{a_{ij}}{\sum_j a_{ij}} + (1-\alpha) \times \frac{a_{ij}}{\sum_j a_{ij}} \quad (1)$$

The parameter α in Equation (1) determines the weight for the outgoing accesses relative to incoming accesses. Possible values of α range from 0.0 to 1.0, where α =1.0 discards incoming accesses in the ranking calculation; and α = 0.0 discards outgoing accesses. We have performed some experiments and found that α should be typically set to some value around 0.8. This value ensures that the outcome of the ranking at each node is mainly derived by the local user accesses, as it gives outgoing accesses higher weight relative to incoming accesses. This is very important, as outgoing accesses are usually order of magnitudes smaller than incoming accesses. For example, if a community contains 100 peers accessing each other, the incoming accesses will be 100 times the outgoing accesses on average. In addition, this value of α does not completely ignore incoming accesses. They still receive some weight and are reflected in the results of the ranking process as well.

The first partial term $a_{ij}/\Sigma a_{ij}$ represents the proportion of N_i outgoing accesses to node N_j. The second partial term $a_{ji}/\Sigma a_{ji}$ represents the proportion of N_i incoming accesses from node N_j. Notice that the value of each of these terms increases as the number of accesses (a_{ij} in the first and a_{ji} in the second) increases. Dividing by the total number of accesses in each of the two terms is not necessary. It, however, normalizes the ranks into values between 0.0 and 1.0 inclusive. In fact, R_i is a probability function with $0 \leq R_i(N_j) \leq$ and $\Sigma R_i(N_i) = 1$.

According to our ranking formula above, the more the mutual accesses with a node, the higher the rank it receives. After ranking peers, the ranking node sorts them in a non-descending order based on the ranks such that nodes with highest ranks occupy the top of the list. This ranked list is then used to build the friend links for the node. As an example, Table 1 shows the access counts for an example network of five nodes, N_1 to N_5. Table 2 gives the ranks calculated based on the access information in Table 1 using α = 0.8. Each row in this table gives the ranks calculated by the corresponding node. For example $R_1(N_2)$, which is the rank of node N_2 as calculated by node N_1, appears in the cell at the intersection of the first row and the second column. The value of $R_1(N_2)$, as given in the table, is 0.3303. The ranked list at node N_4 is the list nodes that have mutual access with node N_4 in a descending order of ranks. From the fourth row in Table 2, that list is N_3, N_5, N_2, and N_1.

Table 1 Access counts showing accesses, for example, network of five nodes (N_1 to N_5)

	N_1	N_2	N_3	N_4	N_5
N_1	0	17	10	13	10
N_2	7	0	9	11	15
N_3	3	11	0	14	8
N_4	6	7	14	0	11
N_5	8	5	11	6	0

Table 2 Ranks calculated based on the access information in Table 1, using $\alpha = 0.8$

	N_1	N_2	N_3	N_4	N_5
N_1	0	0.3303	0.185	0.258	0.2267
N_2	0.2183	0	0.2264	0.2445	0.3107
N_3	0.1121	0.2854	0	0.3747	0.2278
N_4	0.1854	0.1974	0.3584	0	0.2588
N_5	0.2588	0.2015	0.3297	0.21	0

Building the overlay topology according to the mutual interest

The ranked list at each node contains the list of peers who have mutual access with the node. The peers with highest mutual access occupy the top of the ranked list. Each node chooses its friends from the top of its ranked list. By doing this, we introduce spatial locality into the topology based on mutual access. In other words, Freelib nodes with similar interest get closer to each other on the topology. Consequently, the traditional peer-to-peer forwarding of search messages takes these messages to the most relevant nodes in few steps on the overlay topology. The ranked list can be utilized to build the access topology in two different ways to realize this objective. The first, and the simplest, is the routing table approach. The other is the active link approach. In the following subsections, we explain these methods and discuss the advantages and disadvantages of using each of them.

Routing table approach

In this approach, the ranked list is used as a routing table for sending out and forwarding search queries. When a node is about to send out or forward a search request, it sends the search message to the first R available peers on its ranked list. Each node that receives a search request needs to acknowledge it to confirm its availability. The advantages of this approach are that it is dynamic and simple. But it does need extra communication messages for sending back the acknowledgements. Another issue with this approach is that a node does not know and does not have control on the number of nodes keeping it as a friend (we refer to these as incoming friends of the node as opposed to the outgoing friends the node selects from the top of its ranked list). This could cause a popular node (one that every one is accessing) to be overwhelmed with too many requests.

Active link approach

In this approach, every node establishes active links to R peers from the top of the ranked list. These are full links that have a keep-alive mechanism such as pings. The number of friends per node, R, is typically 4 to 6. Every time the ranking process is performed, some friend links might need to be disconnected and some others might need to be established to reflect the most recent ranking results. This whole evolution process is transparent to the users of the system. When a node is saturated (that is, the number of its incoming friends Rin exceeds a certain threshold), it simply rejects any new requests for establishing friend links to it. Listing 3.2 outlines the algorithm for establishing the friend links given in the ranked list of peers as input. Listing 3.3 outlines the algorithm for processing a request for establishing a friend received by a node.

Like the routing table approach discussed in the previous section, this approach has some communication overhead due to establishing links and due to the pings. The ping (keep-alive) messages are needed in order to detect failure of friends. If every node establishes R friends, then we have $R \times n$ total friend links in a network of n nodes. Also the network will have total $c \times R \times n$ pings per unit time, where c is a constant that represents the number of ping messages per link per unit time. This is Ω (n) ping messages for the

whole network. Although this approach is more complex than the routing table approach presented in the previous section, we choose to use it in our design, as it gives each node control on the number of incoming friends.

Handling changing and multiple user interest

In this section, we discuss how our current ranking process handles changing and multiple-topic user interest. User interest usually shifts between topics over a period of time. When a user's interest shifts to a new topic, the user typically starts to submit searches and access peers from the new community. Our ranking process immediately considers new peers. In the transient period, however, these new peers receive low ranks compared to peers that represent the user's old community. As user's accesses to the new peers increase in numbers, they get reflected more and more in the ranking calculations. As the accesses to the new peers start to outnumber old accesses, the user gets connected to the new community. This transition might take long time, however. We are currently working on an enhanced ranking process, which will speed up the transition to the new community. The new ranking process uses aging techniques and access weights. Work in this area is in progress, and we shall report on it in future papers.

In addition to shifting interest, some users might have interest spanning multiple topics simultaneously. In this case, user accesses are typically split among the corresponding communities. Our current peer ranking process will be able to handle this scenario reasonably. If our user's interest is split among a number of topics, so will be his access links,

```
EstablishFriendLinks (RankedList rankedList) {
        create empty list newFriends
        for each peer in rankedList{
                if (peer is in currentFriends list)
                        add peer to newFriends;
                else{
                        try to establish friend link to peer;
                        if (result == OK)
                                add peer to newFriends;
                }
                if (number of peers in newFriends ≥R)
                        break;
        }
        for each peer in currentFriends {
                if (number of peers in newFriends < R){
                        if (peer is not in newFriends)
                                add peer to newFriends;
                } else {
                         if (peer is not in newFriends)
                                dropFriend(peer);
                }
        }
        currentFriends = newFriends;
}
```

Listing 1 Algorithm for establishing friend links

```
establishFriendRequest (PeerInfo peer) {
        if(numberOfIncomingFriends < R_in){
                add peer to incomingFriends;
                status = OK;
        }else{
                Status = SATURATED;
        }
        sendResponse(status);
}
```

Listing 3.3 Processing requests to establish a friend

which will be split over the corresponding communities. This may have some impact on recall. We are investigating new techniques to enhance Freelib handling of users with multiple-topic interest.

Implementation

We have implemented a peer-to-peer client, who implements the Freelib architecture. Figure 2 shows our highly modular client design. It consists of modules, each of which performs a certain function of a group of related functions. The key modules include the network manager module, which maintains the overlay topology, the messenger module, which implements a messaging framework, and the search/access modules, which implements the search and access protocols.

In the following subsections, we present various implementation modules. We group these modules according to the common functionality or service they provide and present each module group in separate

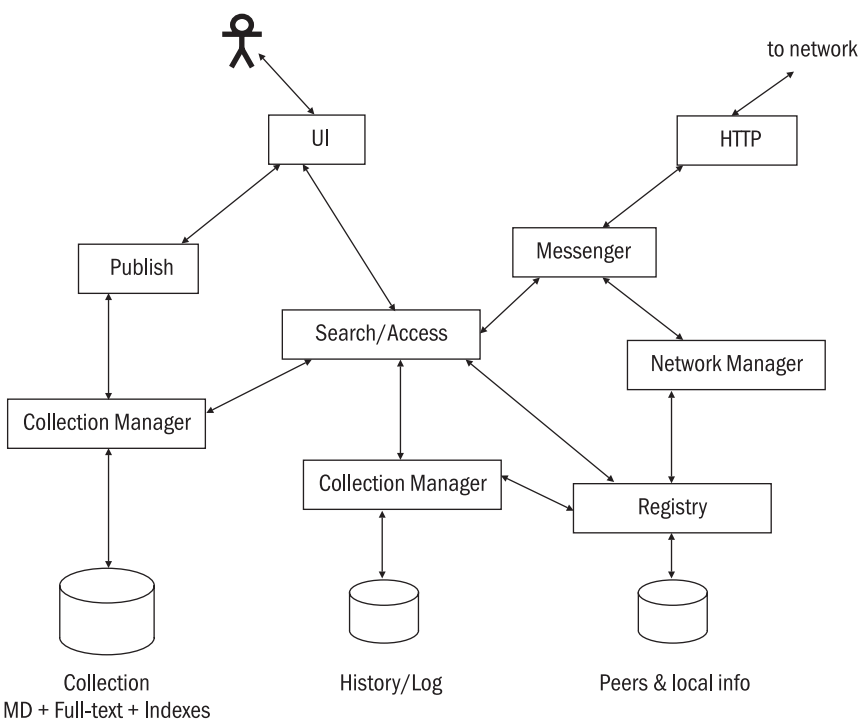

Figure 2 Freelib universal client design

subsection. For example, the modules that provide services directly to the user are in the user modules group. Other groups include the networks manager modules, which are responsible for implementing the various network and topology protocols; messaging modules, which are responsible for communication between peers; log and history modules, which are responsible for maintaining the access log, search history, and the registry, which is a central repository for peer info; and the collection manager responsible for indexing and storing metadata and full-text using the local file system of the machine running on the Freelib client.

User modules

User service modules include four modules that interact with the user and implement services consumed directly by the user. The first and immediately apparent to the user is the user interface module. This module provides the main graphical user interface through which the user interacts with the system and invokes various services, including publishing, searching, and retrieval/access. We borrowed most of Freelib graphical user interface from Kepler [Kep, Mal01]. The new features added in the case of Freelib include multiple tabs and some of the buttons on the main interface to invoke the new functions provided by Freelib. Figure 3 shows the main user interface of the Freelib client. At the top, there is a set of buttons, which enables the user to invoke two of the main services, namely publishing and search. The other buttons in this set are the settings button, which enables the user to configure the Freelib client; the connect/disconnect button, which enables the user to connect to and disconnect from the peer-to-peer network; and the help button, which displays help information to the user. To the right of the main buttons, an icon is displayed. This icon is bright when the client is connected to the network and is greyed when it is disconnected.

The middle area of the main user interface consists of overlay tabs for displaying the local collection and the results for the ongoing user search queries. The first tab is dedicated for displaying the items available in the local collection. For each ongoing search query, an additional tab is created to display the results for the query. The information inside each tab is presented in tabular form, with each row presenting various metadata for one item/document from the result set (or from the local collection in case of the first tab).

In addition to the main buttons at the top, another set of buttons is available at the bottom of the main user interface. These buttons include the exit Freelib button for the exiting Freelib client; the close tab button for closing the current tab, the view details button

Figure 3 Freelib universal client: main user interface

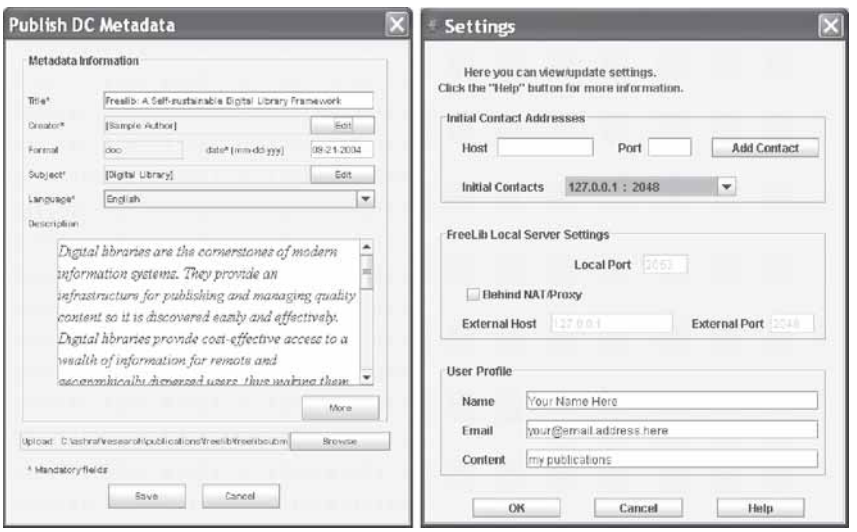

Figure 4 Freelib universal client; left: publishing tool; right: configuration tool

for viewing the detailed metadata for the selected item/document, and the download button initiating download of the selected item. The local collection tab is not closable. When it is the current selected tab, the close tab button is disabled. Closing a search tab effectively releases resources allocated for the result set associated with it. The user is always prompted when the exit Freelib or close tab is clicked before the corresponding action is taken.

Figure 4 shows the Freelib publishing tool and the Freelib configuration tool. The publishing tool is displayed when the user clicks the main publish button. It allows the user to provide various metadata fields and choose a document to publish. The configuration tool allows the user to edit configuration information such as port number to use and proxy information if user is behind proxy. It also allows the user to edit his/her user profile, including name, e-mail address, and description of local collection.

In addition to the user interface module, there are three other modules related to user services. These are the publish, search, and access modules. The publish module is invoked when the user submits metadata for publishing. Currently, this module only publishes to the local collection. In the future, replication to other nodes might be utilized for better availability of content. The search module implements the search protocol. It is invoked by the user interface module when the user enters a search query. This module sends out the search queries, collects results, and forwards them back to the user interface module. By default, this module searches the local collection as well. In the future, this feature, however, should be made configurable by the user. The access module is a simple module invoked by the user interface module when the user clicks the download button. It sends out access requests and handles the responses by saving the downloaded content into a specific subdirectory inside the directory structure of the installed Freelib client.

Messaging modules

There are two modules that implement the messaging framework. These are the messenger module and the HTTP module. The messaging framework is utilized by other modules that communicate with other Freelib nodes, for example, the search and access modules. The messenger module is invoked by the other module to deliver Freelib messages to other nodes. The messenger module supports both synchronous and asynchronous communication modes. In the synchronous mode, the sender thread blocks waiting for a response message. In the asynchronous mode, the sender thread returns immediately, and the messenger forwards response back to the module as it arrives. The HTTP module encapsulates Freelib messages in HTTP requests and sends them out to their destination. The HTTP module on the other end extracts the Freelib message from the incoming HTTP request and forwards it to the messenger module for delivery to the appropriate module. The messenger module exposes two public interfaces. These interfaces are the MessageHandler interface and the MessengerInterface. The messenger interface is implemented by the main messenger class. It specifies the methods that other modules can call to send out the various message types supported. The MessageHandler interface must be implemented by any module that wishes to receive Freelib messages. The single method declared by this interface is called by the messenger module to deliver incoming messages to the implementing modules.

Network modules

The network modules are the modules responsible for implementing and maintaining network protocols. They are four sub-modules that are included in the main network manager module. The first of these four modules is the ring manager module. The ring manager implements the join and leave protocols. In addition, it is responsible for maintaining the support network short contacts. The second module is the long contact module, which is responsible for establishing and maintaining the support network long contacts. The third module is the friend manager module. The friend manager is responsible for establishing and maintaining friends based on the ranked list of peers. The fourth and final module is the maintenance manager module, which is a helper module, which implements some functionality used by the other network modules for maintaining their corresponding portion of the network architecture. Each of these four modules implements the MessageHandler interface and registers with the messenger module as message handler similar to the search module. This enables these modules to use the messaging framework to send and receive messages.

Log, history, and the registry

These modules implement internal functionality, which does not directly involve any network communication nor use of the messaging framework. The log manager module is responsible for logging the incoming and outgoing access in which the local user is involved. This is the access log that is utilized by the peer ranking process to produce the ranked list of peer, which is in turn used for establishing the access contacts. The log manager is invoked by the access module whenever an incoming or outgoing access occurs. Every time the peer ranking process starts, it scans through the access log, calculates peer ranks, and produces the ranked list.

The history module is not implemented in the current version of the Freelib client. It is intended for maintaining a search history of the local user. The search history can be utilized in many ways. It can be used to infer useful information about the user interest. In addition, it can be made available to the user

as necessary for selectively repeating previous searches.

The registry module is a local repository of local and peer information. It contains the Freelib client configuration information and user profile information. In addition, it contains the most recent information about peer nodes.

The collection manager

The collection manager is responsible for maintaining the local collection. In the current implementation, the local collection contains the metadata, documents, and indexes for the content published by the local user. In the future, content replicated from other nodes could be maintained by the collection manager in a separate collection. The collection manager implements a DAO (data access object) design patterns to allow different implementations to be plugged into the client. The DAO pattern exposes a public interface, which declares the public methods used for data access. All the details of the implementation are hidden from the user of the module. The supported collection types in the current Freelib implementation are an XML file system based collection and a Lucene-indexed collection. The former uses XML files to save the metadata. The latter uses the Apache Lucene [Luc] open source software for indexing the metadata. Other collection types can be plugged as necessary, for example, a database collection using JDBC (Java database connectivity)/ODBC (open database connectivity).

Performance evaluation

In this section, we describe our Freelib simulator and outline the simulation process. We then describe the experiments and present the results.

The Freelib simulator

In our preliminary evaluation of Freelib (Amrou, Maly, and Zubair 2006), we utilized a cluster of 32 machines running Linux to operate a real Freelib network. By real, we mean running one copy of the Freelib client software for each participating user. The only exception to that was the user interaction with the system, which was automated. We succeeded in finishing our preliminary evaluation; however, due to the resource-intensive nature of such experiments, we were able to emulate only small networks of up to few hundreds of nodes. In order to evaluate our approach with larger networks, we developed an event-based simulator and utilized a Sun Sparc machine, which has eight processors and 32 GB of memory. We were able to simulate networks of thousands of users using the new simulator. Our simulator is implemented entirely in Java and could be run under various operating systems, including UNIX, LINUX, and Windows.

Objectives and performance measurements

The main objective of our performance study of Freelib is to assess the performance gain from the concept of community evolution. In order to perform this assessment, we compare the performance of searching using our access network against that of searching using the base symphony support network. Hence, the performance of the symphony serves as the baseline for comparison in all our results.

Our evaluation involves three different basic measurements. The first is the average recall over all search queries. We calculate the recall for each individual query by dividing the number of relevant results returned by the number of relevant results in the community. The second measurement is the average bandwidth usage, calculated as the number of application-level messages rather than real bandwidth. The third measurement is the response time, measured as the number of hops on the Freelib network topology rather than physical hops or clock time. In addition,

we calculate normalized recall, defined as recall per unit bandwidth (1 message). This is a measure that we use to combine and display both recall and bandwidth usage information on one graph. Our measurement of bandwidth and response time in terms of number of application-level messages and hops on the overlay topology, respectively, helps us avoid simulating low-level details of the network, which in turn allows us to simulate larger networks.

Experiment set-up

The simulator runs as one java process. The simulation steps are (1) creating and initializing the nodes; (2) publishing the metadata and documents at each node; documents are represented by keywords drawn from a set of keywords that represent the primary user community; (3) building the symphony network according to the symphony protocol (Manku, Bawa, and Raghavan 2003); (4) building the access network; and (5) nodes submitting searches, collecting results, and calculating various evaluation measurements. We model arrival of search queries as a Poisson process with search queries per minute. For example, assigning a value of 0.5 means the user submits a search every two minutes on average. For each search query, some items are randomly selected for access/download. Studies (Granka, Joachims, and Gay 2004; Joachims, Granka, Pang, et al. 2005; Silverstein, Marais, Henzinger, et al. 1999) have shown that users give attention to the first few (typically 10) top-ranked results. Our software simulates this behaviour by giving higher chance for downloading items that are closer to the top of the ranked list of results.

In our simulation, we studied both homogenous networks as well as heterogeneous networks. Homogenous networks are those networks in which all nodes are relatively comparable in terms of the collection size (number of documents published by the node) and the network resources (especially the bandwidth and the number of simultaneous network connections allowed). Heterogeneous networks, on the other hand, are networks that have some percentage of nodes with more powerful network resources and/or larger collections. We call those powerful nodes hub nodes or simply hubs. Although we studied both types of networks, we believe that heterogeneous networks are closer to the real world. We implemented hub nodes by publishing more documents for them as well as relaxing the constraint on the number of friend links they can have. We performed sensitivity analysis on these two parameters. For the number of friend links, we changed the threshold from twice to five times that of regular nodes. We found that increasing the threshold beyond twice that of regular nodes gives slight increase in performance gain. Thus, we chose threshold value for hubs to be twice that of a regular node. Regarding the collection size, we chose the number of documents of a hub node to be 5–10 times that of a regular node. We simulated different communities by having different sets of keywords for different communities. When a node is created, a community is selected for the node, and documents are published by selecting keywords from the chosen community. When a node submits search queries, it uses the keywords from its community. The important simulation parameters include the total number of nodes, the community sizes, and the percentage of hub nodes. In each run, we allow all the nodes to submit multiple search queries and average the results over all the queries. In these experiments, we compare Freelib and a network that only uses symphony, the small-world peer-to-peer network. The details of the experiments are given in the following section.

Experiments

We performed experiments for different networks with up to 4000 nodes, community sizes ranging from 1% up to 20% of the total number of nodes, and hub nodes ranging from 0% to 5% of the nodes. We select graphs from some of the simulated networks for presentation in this section and report in the text about networks with different parameters. Figure 5 shows recall and normalized recall as functions of TTL for networks of 4000 and 500 nodes, with community size of 100. In this figure, we show results for network with 5% hub nodes as well as networks with no hub nodes. We can see from the graphs in the figure that Freelib gives considerable gain in recall over basic symphony. Freelib reaches the same levels of recall in up to three hops (in the graph, the maximum number of hops was seven) less than symphony. The recall gain for other community sizes is similar and increases as the community size decreases compared to the network size as we shall discuss shortly. The same conclusion could also be drawn for the bandwidth usage shown in Figure 6. The graphs in the figure show bandwidth usage as number of Freelib messages versus the recall for different network sizes. The bandwidth savings increase as the community size decreases relative to the network size. In this figure, we show results for networks with 0 hub node. The results for 5% hub nodes are similar, with slight increase in the bandwidth. The increase in bandwidth usage for heterogeneous networks could be explained by the fact that hub nodes forward search queries to many more nodes than regular nodes. In Figure 7, we show response time as hops versus recall for network sizes of 4000 and 500 nodes and 5% hub nodes. The graph shows that Freelib enhances query response time by up to three hops. Figure 8 shows recall and normalized recall as functions of the network size for community sizes 100. This figure shows results for networks with 5% hubs as well as networks with no hubs. As we can see in the graphs of this figure, the recall deteriorates for symphony as the network size increases, while Freelib maintains the same high recall despite the increase in network size. This can be explained by the fact that Freelib effectively targets the relevant nodes in the community even if the network size is large. On the other hand, symphony recall level degrades as the network size increases because the relevant nodes are dispersed among larger number of nodes. Finally, Figure 9 shows recall and normalized recall as a function of the community size. The community sizes in this figure are percentage of the network size. As shown in the figure, the gain in both the recall and normalized recall increases as the community size relatively decreases compared to the network size. This behaviour is beneficial to Freelib. In fact, we believe that in the real world, community sizes will be much smaller percentage of the whole network than the figures that we use here, which in turn will allow Freelib to achieve even higher gains in performance.

Conclusion and future work

In this paper, we have compared the performance gains of Freelib over symphony. The comparison shows that Freelib provides considerable performance gains over symphony. The performance gains are significant in almost all cases, especially when the community size is relatively small compared to the network size, which we believe is the case in real peer-to-peer networks. Freelib gives better performance in terms of higher recall, lower bandwidth consumption, and better response time.

As a future work, we plan to study the performance of Freelib with even larger peer-to-peer networks. This might involve the

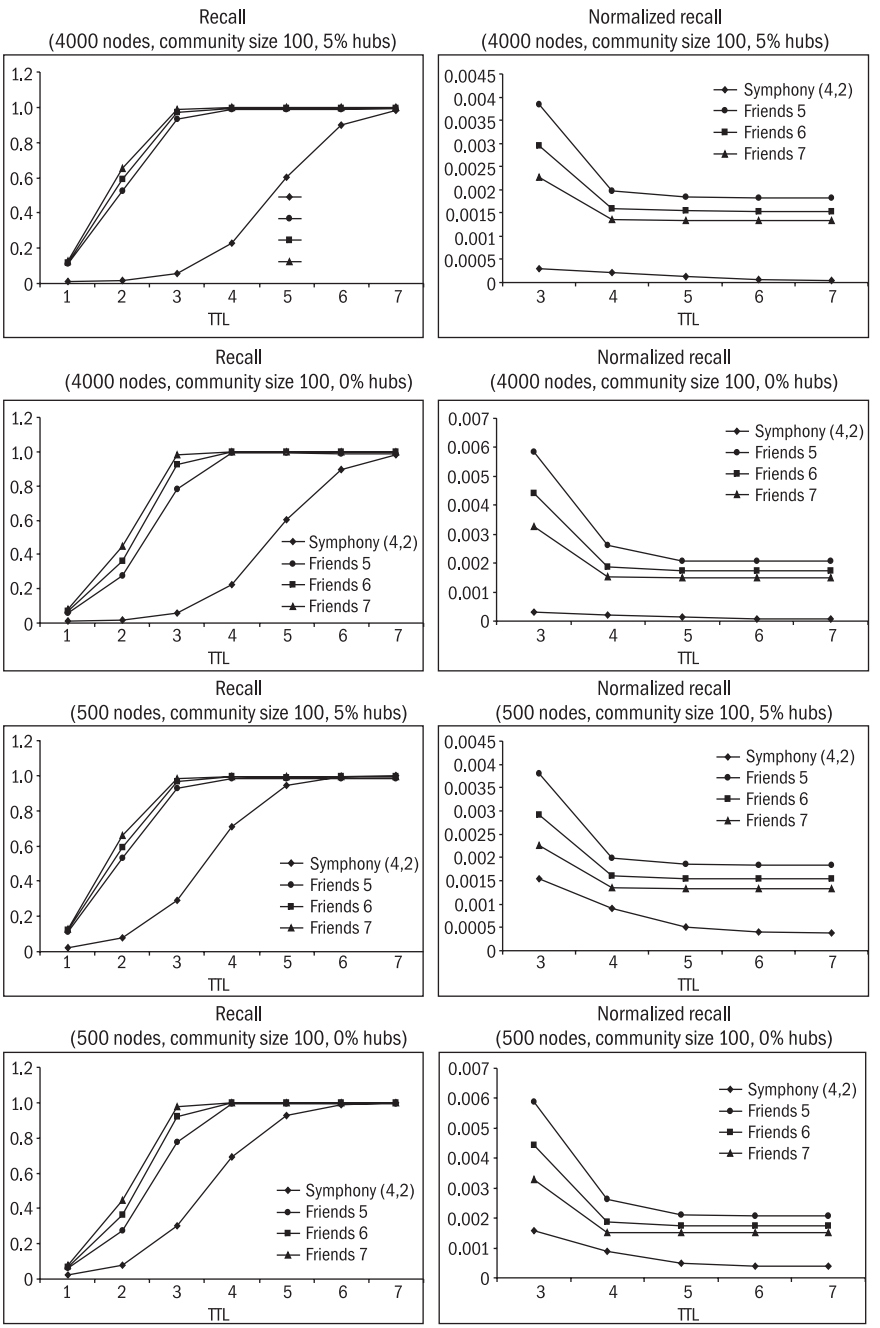

Figure 5 Recall and normalized recall vs. TTL (time to live; nodes: 4000, 500; community: 100; hubs: 0, 5%)

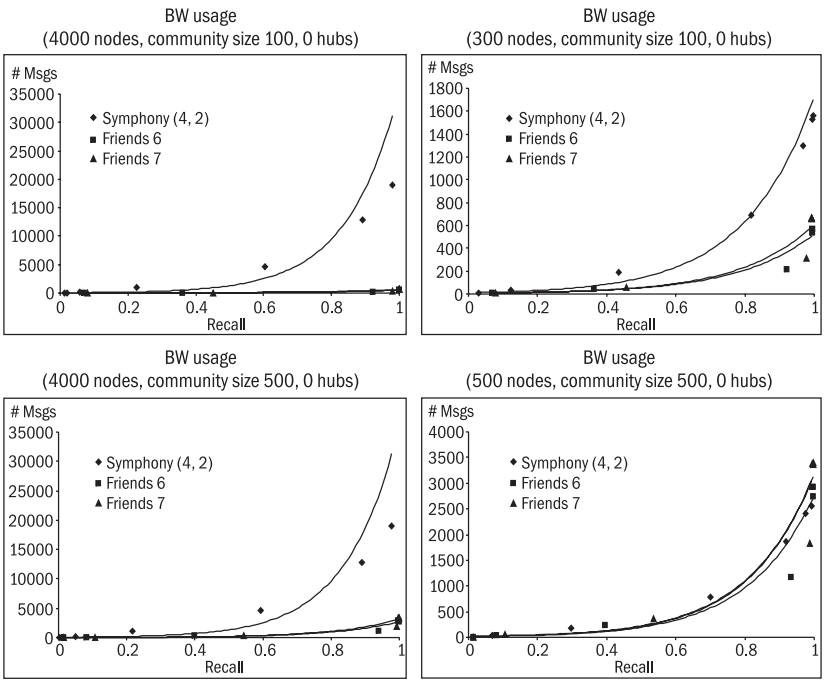

Figure 6 Bandwidth usage vs. recall (nodes: 500, 4000; community: 100, 500; hubs: 0)

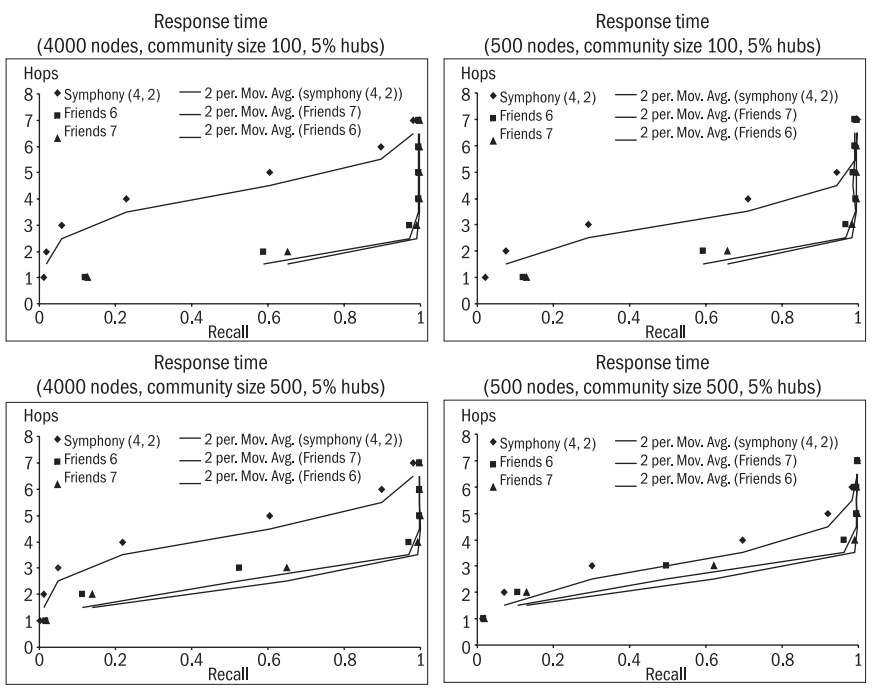

Figure 7 Response time vs. recall (nodes: 500, 4000; community: 100, 500; hubs: 5%)

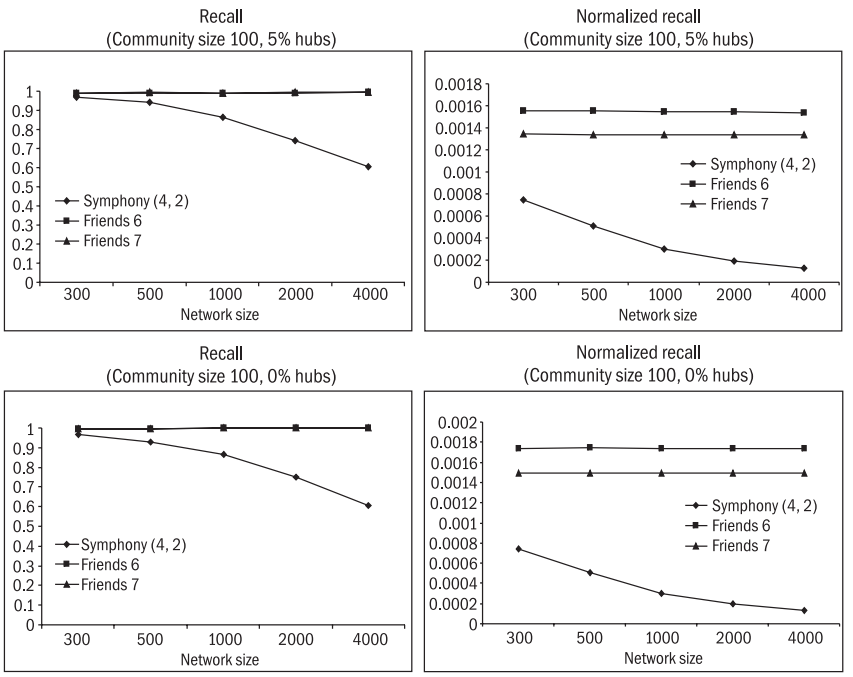

Figure 8 Recall and normalized recall vs. network size (community: 100, 500; hubs: 0, 5%)

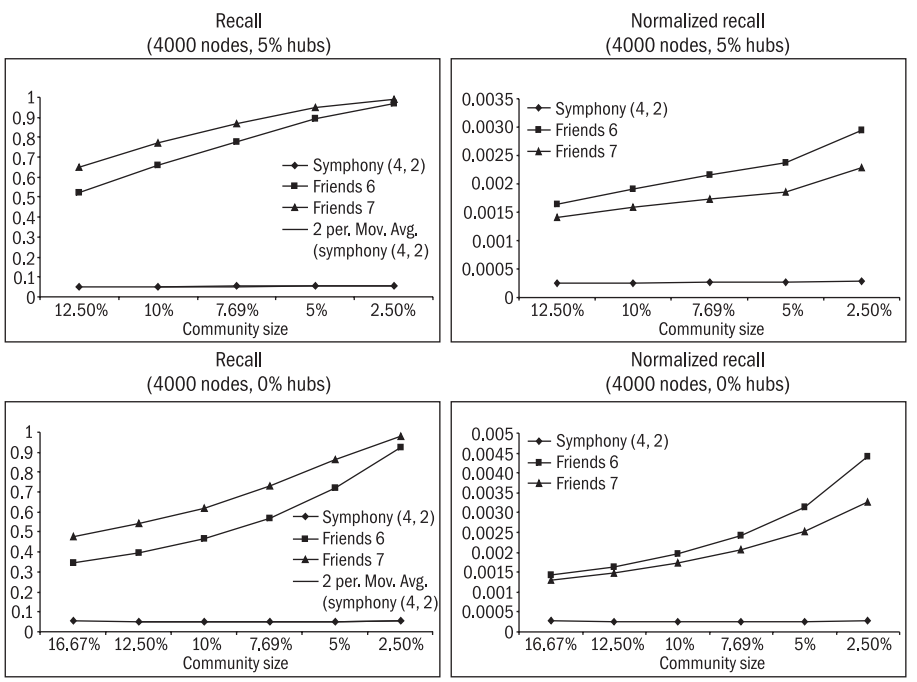

Figure 9 Recall and normalized recall as functions of community size (nodes: 4000; hubs: 0, 5%)

development of a distributed Freelib simulator, which would distribute the simulation load over multiple machines. In addition, we plan to introduce changes to our peer ranking techniques and study their effect on performance. Furthermore, we plan to study the effect of misleading accesses on the performance of Freelib. Misleading accesses are those accesses that seem to be random and not related to the user interest area. We shall devise techniques for identifying such accesses. We shall then introduce this type of accesses into the simulation and study the performance of the system. In the light of the results of such a study, we shall decide if we need to devise techniques and measures for eliminating those accesses from the ranking process.

References

ACM (Association for Computing Machinery). 2008
ACM digital library, home page
Details available at <http://portal.acm.org/dl.cfm/>, last accessed on 10 June 2008

Alexandria. 2008
Alexandria digital library project
Santa Barbara: University of California
Details available at <http://alexandria.sdc.ucsb.edu/>, last accessed on 10 June 2008

Amrou A, Maly K, and Zubair M. 2004
Freelib: a self-sustainable digital library for education community, pp. 15–20
In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-MEDIA'04)*
[World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-MEDIA'04), Lugano, Switzerland]

Amrou A, Maly K, and Zubair M. 2006
Freelib: peer-to-peer-based digital libraries, pp. 9–14
In *Proceedings of the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA 2006)*
[Proceedings of the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA 2006), Vienna, Austria, 18–20 April 2006]

Amrou A, Maly K, and Zubair M. 2006a
Performance evaluation of Freelib, a P2P-based digital library architecture
In *Proceedings of the International Conference on Digital Libraries (ICDL 2006)*
[Proceedings of the International Conference on Digital Libraries (ICDL 2006), New Delhi, India, 5–8 December 2006]

Arc. 2008
Arc project home page
Old Dominion University
Details available at <http://arc.cs.odu.edu/>, last accessed on 10 June 2008

Bawa M, Manku G S, and Raghavan P. 2003
SETS: search enhanced by topic segmentation
In *Proceedings of the 26 Annual International ACM SIGIR 2003*
[Proceedings of the 26 Annual International ACM SIGIR 2003, Toronto, Canada, 28 July to 1 August 2003]

Clarke I, Sandberg O, Wiley B, Hong T W. 2000
Freenet: a distributed anonymous information storage and retrieval system, pp. 46–66
In Designing Privacy Enhancing Technologies: Proceedings of International Workshop on Design Issues in Anonymity and Unobservability
[International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 2000]

Daswani N, Garcia-Molina H, and Yang B. 2003
Open problems in data-sharing peer-to-peer systems
In *Proceedings of the 9th International Conference on Database Theory (ICDT 2003)*
[9th International Conference on Database Theory (ICDT 2003), Siena, Italy, 8–10 January 2003]

Ding H and Solvberg I. 2004
Metadata harvesting framework in P2P-based digital libraries
In *Proceedings of the International Conference on Dublin Core and Metadata Application 2004*
[International Conference on Dublin Core and Metadata Application 2004, Shanghai, China, 11–14 October 2004]

Granka L A, Joachims T, and Gay G. 2004
Eye-tracking analysis of user behaviour in WWW search
In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*
[27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, University of Sheffield, UK, 25–29 July 2004]

Joachims T, Granka L, Pang B, Hembrooke H, Gay G. 2005
Accurately interpreting clickthrough data as implicit feedback
In *Proceedings of the 28th Annual ACM Conference on Research and Development in Information Retrieval (SIGIR)*
[28th Annual ACM Conference on Research and Development in Information Retrieval (SIGIR), Salvador, Brazil, 15–19 August 2005]

Kleinberg J. 2000
The small-world phenomenon: an algorithmic perspective
In *Proceedings of the 32nd ACM Symposium on Theory of Computing*
[32nd ACM Symposium on Theory of Computing, Portland, OR, USA]

Li M, Lee W, Sivasubramaniam A, Lee D. 2004
A small world overlay network for semantic based search in P2P systems, pp. 71–90
In *Proceedings of the Second WWW Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGRID'04)*
[Second WWW Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGRID'04), New York City, NY, May 2004]

Liu X, Maly K, Zubair M, Nelson M. 2001
Arc – an OAI service provider for digital library federation
D-Lib Magazine **7**(4), April 2001

Manku G S, Bawa M, and Raghavan P. 2003
Symphony: distributed hashing in a small world
In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*

McNab R J, Witten I H, and Boddie S J. 1998
A distributed digital library architecture incorporating different index styles, pp. 36–45
In *Proceedings of the IEEE Forum on Research and Technology Advances in Digital Library*
[IEEE Forum on Research and Technology Advances in Digital Library, Santa Barbra, CA, April 1998]

NZDL (New Zealand Digital Library). 2008
The New Zealand digital library home page
University of Waikato, New Zealand
Details available at <http://www.nzdl.org/fast-cgi-bin/library?a=p&p=home>, last accessed on 10 May 2008

Silverstein C, Marais H, Henzinger M, Moricz M. 1999
Analysis of a very large AltaVista query log
ACM SIGIR Forum 33 (1)

Walkerdine J and Rayson P. 2004
P2P-4-DL: digital library over peer-to-peer
In *Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P'04)*
[4th International Conference on Peer-to-Peer Computing (P2P'04), Zurich, Switzerland, 25–27 August 2004]

Watts D J and Strogatz S H. 1998
Collective dynamics of 'small-world' networks
Nature 393: 440–442

Yang B and Gracia-Molina H. 2002
Improving search in peer-to-peer networks
In *Proceedings of the 22nd Conference on Distributed Computing Systems (ICDCS'02)*
[22nd Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, 2–5 July 2002]