

**ADAPTIVE PEER TO PEER SOCIAL NETWORKS FOR DISTRIBUTED
CONTENT BASED WEB SEARCH**

Le-Shin Wu (lewu@indiana.edu)

Department of Computer Science, School of Informatics

Indiana University, Bloomington

1900 East Tenth St., Bloomington, IN 47406, USA

Phone: 1-812-8579707, Fax: 1-812-8554829

Ruj Akavipat (rakavipa@indiana.edu)

Department of Computer Science, School of Informatics

Indiana University, Bloomington

1900 East Tenth St., Bloomington, IN 47406, USA

Phone: 1-812-8579707, Fax: 1-812-8554829

Ana G. Maguitman (agm@cs.uns.edu.ar)

Departamento de Ciencias, e Ingeniería de la Computación,

Universidad Nacional del Sur,

8000 Bahía Blanca, Buenos Aires, Argentina

Phone: 54-291-4595101 (Ext 2611), Fax: 54-291-4595136

Filippo Menczer (fil@indiana.edu)

Department of Computer Science, School of Informatics

Indiana University, Bloomington

1900 East Tenth St., Bloomington, IN 47406, USA

Phone: 1-812-8561377, Fax: 1-812-8561995

Keywords: Information search and retrieval, Web-Based Applications, Distributed Systems, IS Performance Evaluation.

ADAPTIVE PEER TO PEER SOCIAL NETWORKS FOR DISTRIBUTED CONTENT BASED WEB SEARCH

ABSTRACT

This chapter proposed a collaborative peer network application called 6Search (6S) to address the scalability limitations of centralized search engines. Each peer crawls the Web in a focused way, guided by its user's information context. Through this approach, better (distributed) coverage can be achieved. Each peer also acts as a search “servent” (server + client) by submitting and responding to queries to/from its neighbors. This search process has no centralized bottleneck. Peers depend on a local adaptive routing algorithm to dynamically change the topology of the peer network and search for the best neighbors to answer their queries. We present and evaluate learning techniques to improve local query routing. We validate prototypes of the 6S network via simulations with 70–500 model users based on actual Web crawls. We find that the network topology rapidly converges from a random network to a small world network, with clusters emerging from user communities with shared interests. We finally compare the quality of the results with those obtained by centralized search engines such as Google.

BACKGROUND AND MOTIVATION

Centralized search engines have difficulties in achieving good coverage of the Web (Lawrence & Giles, 1999) because the Web is large, fast-growing and fast-changing (Brewington & Cybenko, 2000; Fetterly et al., 2003; Ntoulas et al., 2004). Further, various biases introduced to address the needs of the “average” user imply diminished effectiveness in satisfying many atypical search needs. Examples of bias include interfaces (advanced search features are often buried and poorly documented), ranking (in favor of precision and popularity, to please the majority of users who do not look beyond the first few hits), and coverage (well connected pages are easy for a crawler to find and thus more likely to be indexed (Najork & Wiener, 2001)).

We identify the above limitations as problems of scale in spite of enormous progress in crawling (Cho & Garcia-Molina, 2002), indexing (Dean & Ghemawat, 2004), retrieval and ranking (Brin & Page, 1998), the “one engine fits all” model does not --cannot-- scale well with the size, dynamics, and heterogeneity of the Web and its users.

Topical or vertical search engines are one approach to address this problem. Effective topical crawling algorithms have been designed to support specialized portals (Chakrabarti et al., 1999; Menczer & Belew, 2000; Menczer et al., 2004). However, these effort are generally aimed to very limited information spaces — digital libraries, Web sites, databases — and are not designed to scale with searching the Web at large.

It is evident that distributed systems are part of the answer to the scale problem. Peer social networks are increasingly seen as a candidate framework for distributed Web search applications. A social network is a social structure between participants which are connected through various social relationships. In the real world, we discover that people can successfully find relevant information for questions by just asking the “right” people through their social network, although the network is extremely dynamic (for example, people may not be available all time, people may change their interests anytime, or people can decide not to respond to requests, etc.). Thus, a peer to peer (or P2P) social network searching system is a network that uses the social network as the basis to route queries for information retrieval. Each peer in the network acts just as a person in the social network:

- Peers are independent.
- A peer can enter and leave the network at any time.
- Peers learn and store profiles of other peers with a view to their potential for answering prospective queries.
- Peers discover new peers through their current neighbors.

By simulating the information finding mechanism in a social network of people, the peer network collectively tries to route the queries to the “right” peers according to some peer selection algorithms which predict the degree of match between queries and peers.

A P2P computer social network relies on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively few servers (Wikipedia, 2005). The most popular use of a P2P network is for file sharing. Applications such as Gnutella (<http://www.gnutella.com>), BitTorrent (<http://www.bittorrent.com>) and KaZaa (<http://www.kazaa.com>) allow peers to share content files, mostly media related, among peers without having to set up dedicated servers and acquiring large bandwidth to support the whole community. The P2P file sharing application is by no means replacing the dedicated servers in content distribution. It simply provides an alternative for content distribution by trading the speed and reliability of dedicated servers for the ease of sharing, lower cost, fault tolerance, and lower bandwidth requirement for a file sharer. In a similar way as P2P file sharing applications are used to facilitate content distribution, P2P applications can be developed to facilitate Web search.

There is extensive work on peer network searching applications in the AI and IR literature. (There are too many examples to list here; the reader is referred to the review in (Lua et al., Second Quarter 2005; Risson & Moors, 2004) as starting points.) One model proposed by the YouSearch project is to maintain a centralized search registry for query routing (similar to Napster), and moreover enrich the peers with the capability to crawl and index local portions of the Web (Bawa et al., 2003). Unfortunately, the central control in this approach makes it difficult to adapt the search process to the heterogeneous and dynamic contexts of the peer users.

A completely decentralized approach is the Gnutella model, in which queries are sent and forwarded blindly by each peer. The problems of this approach are that peers flooded by requests cannot manage the ensuing traffic, and that the topology is uncorrelated with

the interests of the peer users. As a result, the basic Gnutella model does not scale well with the number of users and queries. Adaptive, content based routing has been proposed to overcome this difficulty in the file sharing setting. NeuroGrid (Joseph, 2002) employs a learning mechanism to adjust metadata describing the contents of nodes. A similar idea has been proposed to distribute and personalize Web search using a query-based model and collaborative filtering (Pujol et al., 2003). Search however is disjoint from crawling, making it necessary to rely on centralized search engines for content.

An intermediate approach between the flood network and the centralized registry is to store index lists in distributed, shared hash tables (Suel et al., 2003). In pSearch (Tang et al., 2003) latent semantic analysis (Deerwester et al., 1990) is performed over such distributed hash tables to provide peers with keyword search capability. This is a promising approach, however Li et al. (2003) argue that full-text Web search is infeasible in both the flood model and the distributed hash table model.

Another alternative are hybrid peer networks, where multiple special directory nodes (hubs) provide construct and use content models of neighboring nodes to determine how to route query messages through the network (Lu & Callan, 2003). In hybrid peer networks, leaf nodes provide information and use content based retrieval to decide which documents to retrieve for queries.

In this chapter, we propose an alternative model for peer-based Web search, which uses the same idea of content based models of neighboring nodes but without assuming the presence of special directory hubs. Each peer is both a (limited) directory hub and a content provider; it has its own topical crawler (based on local context), which supports a local search engine---typically but not necessarily a small one. Queries are first matched against the local engine, and then routed to neighbor peers to obtain more results. Initially the network has a random topology (like Gnutella) and queries are routed randomly as in the flood model. However, the protocol includes a learning algorithm by which each peer uses the results of its interactions with its neighbors (matches between queries and responses) to refine a model of the other peers. This model is used to dynamically route queries according to the predicted match with other peers' knowledge. The network topology is thus modified on the fly based on learned contexts and current information needs. Similar ideas are receiving increasing attention in the P2P search literature (Crespo & Garcia-Molina, 2002; Kalogeraki et al., 2002; Yang & Garcia-Molina, 2002).

The key idea of the proposed peer search network is that the flooding problem can be alleviated by intelligent collaboration between the peers. This should lead to an emergent clustered topology in which neighbor communities tend to form according to clusters of peers with shared interests and domains. In fact we predict that the ideal topology for such a network would be a "small world" (Watts & Strogatz, 1998). This topology allows for any two peers to reach each other via a short path (small diameter) while maximizing the efficiency of communication within clustered peer communities. Following Milgram's famous experiments on "six degrees of separation" (Watts & Strogatz, 1998), we named our model 6Search (6S).

Outline and Contributions

After a brief introduction, we start from presenting the 6S protocol and algorithms including the message primitives, neighbor management, neighbor modeling and adaptive query routing. Then we describe the architecture of the 6S system which integrates the protocol and algorithm. In the remaining of the chapter, we explain how we conduct an experiment to test our 6S system based on real Web data and discuss our results. We also propose an alternative evaluation approach for peer network system. Finally, we conclude by summarizing our results and discussing future work. The following findings are our main contributions:

- the 6S nodes rapidly discover the content locality among their peers, displaying a topology that converges to a small-world network after each peer has routed as few as 5-6 queries, and this change in topology leads to an increase in the quality of the results;
- the collective search performance of the system improves as more sophisticated learning algorithms are employed by the peers to route queries, and as more network resources become available, conversely performance degrades softly as bandwidth and CPU cycles become more scarce;
- the 6S peers achieve a search quality that is comparable to that of Google, and significantly outperforms that obtained by a centralized search engine with the same resources (size of crawl set) as the 6S peer collective;
- the 6S algorithms scale very well up to 500 peers, the maximum number of users we were able to simulate in a closely controlled testing environment, giving us confidence for the public release of an open prototype.

6S PROTOCOL AND ALGORITHM

As shown in Figure 1, the 6S application (personal search engine) layer sits between the user and the peer network layer. The 6S peer network protocol acts as an application layer between the search engine and the network (TCP/IP) layer. The application also interfaces with the network using the HTTP protocol for crawling the Web.

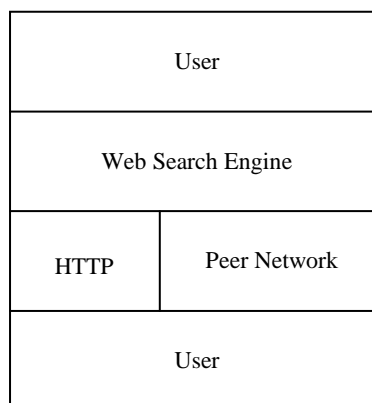


Figure 1: 6S protocol stack.

The 6S peer network layer provides the means to find results (hits) by querying the indexes built by peer search engines. When the user submits a query to its personal search engine, the latter can retrieve hits from its local index database and augment the results by searching the peer network for additional hits.

The design of our protocol is based on the following considerations:

1. peers are independent;
2. a peer can enter and leave the network at any time;
3. a peer should not be overwhelmed by other peers;
4. a query should not be propagated indefinitely;
5. a peer may choose not to respond or forward some queries;
6. the architecture should make it difficult to create denial of service (**DoS**) attacks using the service.

Below we discuss the message primitives that the protocol uses for communication. The following section discusses the algorithm and parameters of each message type.

Message Primitives

We do not wish to design an overly complex protocol, which could hinder the development of improved protocols in the future. In the following, we present the most fundamental primitives that we feel one cannot do without. Here we use a simple XML syntax to illustrate peer network messages. Our prototype protocol is based on these primitives. There are a few additional primitives that we are considering for future implementations as they would enable richer peer interactions and more sophisticated search and learning algorithms. Those are omitted for brevity. It should also be noted that from the network layer (TCP/IP) peers can identify each other during communication (from their IP addresses, say), so this information is omitted in the peer network primitives.

Query message

```
<Query>
  <ID></ID>
  <TTL></TTL>
  <timestamp></timestamp>
  <body>
    <word> kwd1 </word><weight> wt1 </weight>
    :
    <word> kwd2 </word><weight> wt2 </weight>
  </body>
  <ownerid></ownerid>
</Query>
```

The query message is used by a peer to pass its queries to other peers on the network. A query owner identification may optionally be attached to the query. We allow this option into the primitive since a peer may need to identify itself to other peers.

A peer sends a query consisting of its query keywords and corresponding weight of each keyword (weights can be 1 by default). Attached with each query are **ID**, **TTL** (time to live), and **timestamp**. An owner identification can be attached as a sign that one wants to discover new neighbors. The **ID** and **timestamp** are added to help differentiate each query. Given two peers, the **ID** has to be locally unique.

The purpose of **TTL** is to limit the forwarding of a query (see below) such that a query will not survive in the network too long and move too far from the originating peer. This is a standard technique to limit congestion and loops in any network protocol. The **TTL** is decreased for every forward and a query will not be forwarded when **TTL** reaches 0. In 6S we may allow for the amount by which the **TTL** is decreased by a peer to depend on local variations of the protocol.

Query response

```
<QueryResponse>
  <timestamp></timestamp>
  <body>
    <hit>
      <score></score>
      <url></url>
      <summary></summary>
      <info></info>
    </hit>
    :
  </body>
  <ownerid></ownerid>
</QueryResponse>
```

The query response is used by a peer to respond to other peers' search queries. As in the query message primitive, an optional owner **ID** is provided so that the responder may identify itself. Once a peer receives a query it will decide whether it should respond or not. If it decides to respond, it will match the query against its local index database and return N_h results (hits) in the response message.

Moreover, depending on the similarity between the query and its neighbor profiles, the peer may also select some of its neighbors and forward the query to them, as will be illustrated in detail later. Then the peer will forward its neighbors' responses, if any, back to the peer who originated the query.

We impose a restriction on the way that a peer replies to a forwarded query. The response must be sent to the neighbor that forwarded the query, and not directly to the peer who originated the query. This is because we want to prevent potential **DoS** attacks created by exploiting the response system. If a peer responds to a forwarded query by

sending a reply directly to the source, someone can inject a query with a large **TTL** into the network together with a spoofed return address. Then all the responses will be directed to that address, overwhelming the target machine.

Profile request

```
<ProfileRequest></ProfileRequest>
```

The profile request is needed to let a peer request profiles from others. The profile describes what a peer has indexed and is ready for sharing. We use the pull mechanism because it spares a peer from the load of having to propagate updates for its own profile. The cost of a peer having to request for profile information should be lower than that of having to keep track of all peers that store one's profile.

Profile response

```
<ProfileResponse>
  <body>
    <word></word>
    :
    <word></word>
  </body>
</ProfileResponse>
```

This primitive allows a peer to respond to a request for its own profile. Such a profile initially consists of a simple list of terms. Later, as peers learn about their neighbors' expertise from query responses, profiles are updated with more information. More complex profiles and updating algorithms will be described in more detail later.

Neighbor Management

Since our goal is to allow peers to form communities without centralized control, a peer needs to find new peers and evaluate their quality and match. In our design, we choose not to have peers aggressively flooding the network looking for other peers unless it is necessary to do so, such as when the peer enters the network for the first time or when all known peers are not available. Otherwise, a peer would discover new peers through its current neighbors. The process that we use in our prototype is to let a peer attach its contact ID with the query in the **ownerid** field. If the peer that receives a query wants to become a neighbor of the requesting peer, it will respond with its own contact ID in the **ownerid** field of the response message. The new neighbor peers can later contact each other directly. For example, illustrated in Figure 2, a peer sent out Query 1 with its contact ID attached. One of its Neighbor's Neighbor replied Query 1 and also attached the contact ID with response message. Since this peer had the contact information about its Neighbor's Neighbor now, for the next query – Query 2, it can send this query to its Neighbor's Neighbor directly without passing through other peers.

In addition to the mechanism for discovering new neighbors, we also need to consider the issue of how often or when a peer will want to find more neighbors. A simple

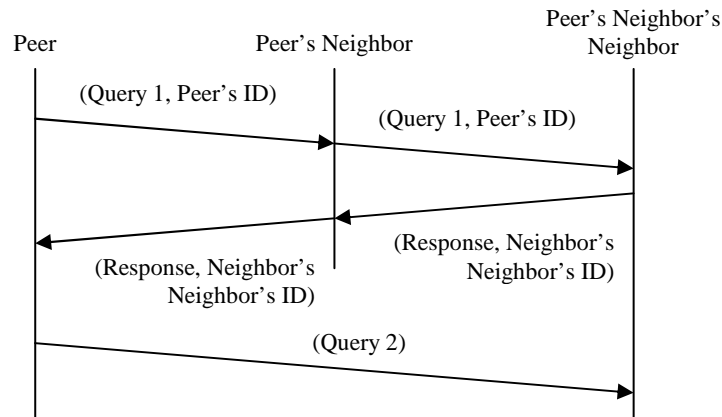


Figure 2: 6S neighbor discovery

approach is to give each peer a fixed number of slots for neighbors, N_n . This number can vary among peers depending on their bandwidth and computational power to process neighbor data. We assume that N_n is fixed for each peer. A peer will search for new peers when its neighbor slots are not full or when it wants to find better neighbors than the currently known peers.

Each peer may of course know about more than N_n peers. Let us call $N_k(t)$ the number of peers known at time t . This number can grow arbitrarily, but probably will be capped at some parameter determined by the peer application's available memory or storage. A peer must prune neighbor information as needed.

Many neighbor management algorithms in the P2P literature require peers to send update messages in order to maintain valid network information when peers leave the network. In contrast, a 6S peer does not need to send any message when it wants to leave the network because our query routing algorithm, which will be discussed later in detail, animatedly updates the neighbor profiles based on queries and responses in the system.

Neighbor Modeling and Adaptive Query Routing

6S relies on adaptive query routing to shape its dynamic network topology. To support adaptive query routing, each peer learns and stores profiles of other peers with a view to their potential for answering prospective queries. A neighbor profile is the information a particular peer maintains to describe its knowledge about what that neighbor stores in its search engine index and is ready for sharing. By checking profile information, peers try to increase the probability of choosing the appropriate neighbors to route their queries.

Akavipat *et al.* (2004) implemented a simple method to initialize and maintain peer profiles: first ask a neighbor for its description, defined as a list of n most frequent keywords in the neighbor's index; then perform a crude update to this list by adding query terms for which the neighbor returns good responses. The score of a keyword in such a neighbor profile is the highest similarity score of the responses a neighbor returns for that keyword. This method, albeit crude and fragile (due to its dependency on information

supplied by neighbors), was shown to give rise to an efficient network topology and promising initial results.

Let us now improve on the reliability and robustness of the simple learning algorithms above by introducing a better profile representation and a novel soft updating scheme. Interactions with peers reveal information of varying reliability. For example, a direct response to a query is telling about a peer's knowledge with respect to that query, but may also reveal (less reliable) information about the peer's knowledge relative to other queries. We want to capture all available information in profiles, but must discriminate information on the bases of its reliability. To this end, let each peer maintain two profile matrices, W^f and W^e for focused and expanded information, respectively. Each profile matrix has the same structure; rows correspond to terms and columns to peers. Thus an element w_{ip} of W is the contribution of term i to the profile of known peer p ($p = 1, \dots, N_k$).

Each peer starts with both neighbor profiles empty. After participating in query forwarding and responding, different updates will be made to each type of profile.

Focused profile: weights w_{ip}^f are updated initially based on p 's response to a neighbor profile request message, and successively through query-response interaction---namely for terms i in queries submitted or forwarded to p . When a peer receives responses to a query Q , it compares the incoming hits with its local hits. Based on this comparison, the peer makes an assessment about p 's knowledge with respects to terms $i \in Q$.

Expanded profile: weights w_{jp}^e are updated through query-response interaction analogously to the focused profile, but for terms $j \notin Q$ that co-occur with terms $i \in Q$ in a hit page d returned by p and have a higher term frequency: $TF(j, d) > \max_i TF(i, d)$. If a certain set of documents is a good response for a certain query, then it may as well be a good response for queries that are well represented in the set. Thus the expanded profile implements a form of query expansion, which we expect to speed up learning since queries are typically short and thus W^f is typically rather sparse.

The neighbor profile update algorithm should enable peers to quickly learn the dynamic properties of the network. Such dynamic properties include the network topology and peers' knowledge. Many neighbor knowledge update algorithm in the literatures as described in Background require peers to periodically send and receive update messages in order to maintain peer information. We want our neighbor learning algorithm to instead dynamically update the neighbor profiles according to the natural interactions in the system, namely queries and responses.

In Akavipat *et al.*'s implementation (Akavipat et al., 2004), a peer updated its neighbor profiles only when a score of any neighbor hit was better than at least one of the top N_h local scores. In such cases the query keywords were added into the neighbor profile with the best score of the neighbor hit as the new weight. Let us now modify the

algorithm so that the peer will always update its neighbor profile no matter whether the score of the neighbor hit is better or worse than the local score. Furthermore, instead of using the best score as the new value for term weights, we propose the following learning rule to update the weights of the query terms in the neighbor profile matrices:

$$w_{ip}(t+1) = (1 - \gamma) \cdot w_{ip}(t) + \gamma \cdot \frac{S_p + 1}{S_l + 1} \quad (1)$$

where t is a time step, S_p and S_l are the average scores of p 's hits and the local hits respectively in response to the query Q , and γ is a learning rate parameter ($0 < \gamma < 1$). The terms i subject to this learning rule depend on Q and the profile matrix (focused or expanded) as described above.

The actual set of N_n neighbors, i.e. those to whom queries are sent, is selected dynamically for each query at time t among the $N_k(t)$ known peers. Sophisticated algorithms have been proposed for determining the quality of peers (Kamvar et al., 2003). Here instead we propose a very simple adaptive routing algorithm to manage neighbor information and to use such information for dynamically selecting neighbors to query:

1. When a peer is first discovered, a profile is requested. A description for the peer is then initialized with the list of keywords contained in the peer's profile.
2. Responses from neighbors (and neighbors' neighbors, and so on) to query Q are evaluated and used to update the description of each known peer:
 - a. The hit average scores (S_p) of hits received from neighbors and the local hit average scores (S_l) are computed.
 - b. Update W^f using Equation 1 for terms in Q .
 - c. If $S_p > S_l$ update W^e using equation 1 for terms not in Q that occur in the hits received from neighbors more frequently than the query terms.
 - d. The discovery signal `ownerid` is sent with the next query to that neighbor.
 - e. New peers that respond to discovery signals are added to the list of known peers, with their profile.
3. For the next query Q' , known peers are ranked by similarity σ between the query and the peer profiles computed as follows:

$$\sigma(p, Q') = \sum_{i \in Q'} [\alpha \cdot w_{ip}^f + (1 - \alpha) \cdot w_{ip}^e] \quad (2)$$

where α is a reliability parameter regulating the contributions of focused and expanded profiles. Typically $0.5 < \alpha < 1$ to reflect higher confidence in focused profile weights as they come from direct responses to queries.

4. The top N_n ranked among known peers are selected as neighbors and sent Q' .
5. Goto step 2.

6S ARCHITECTURE

Using the protocol described earlier, we create the architecture of a peer as shown in Figure 3. Each peer has a local search engine with its own index database. The peer not only processes its own local queries but also the queries that are passed to it by other peers. The peer contains five basic modules, allowing us to easily test and modify each component separately. The system is implemented in Java to take advantage of a number of code libraries available from other sources.

The **User Interface module** is where the peer search system accepts queries from the user and displays the results back to the user. When the user enters a query, this module distributes it to three other modules (Combinator, Local Search, and Neighbor Information) where the query is further processed.

The **Local Search module** handles the search task on a local index created from shared personal files, bookmarked pages, and pages crawled by the local Web crawler. We use the open-source search engine Nutch (<http://lucene.apache.org/nutch/>) as the local indexing and database code.

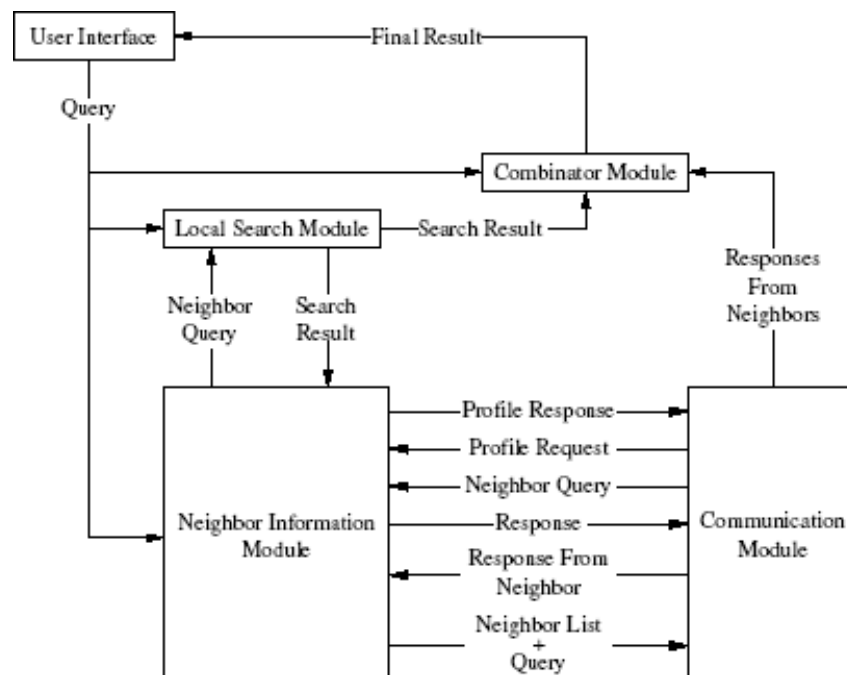


Figure 3: 6S peer architecture

For the topical crawler, we use a *best-N-first* search algorithm developed by Menczer *et al.* (Menczer *et al.*, 2004; Pant, Bradshaw *et al.*, 2003; Pant, Srinivasan *et al.*, 2003), which has been proven very effective against a number of crawling algorithms. A detailed description of this crawling algorithm is outside the scope of this chapter and can be found in the above references. Briefly, the crawler is given a set of seed URLs to start from and a set of topic keywords. The URLs to be visited are prioritized by the similarity between the topic and the page in which a URL is encountered. Some additional

mechanisms guarantee that the crawler is sufficiently exploratory. This crawler is also publicly available from <http://informatics.indiana.edu/fil/IS/JavaCrawlers>.

The peer crawler can be seeded with pages bookmarked by the user, or hits returned by a search engine based on a user profile, or pages visited recently by the user. The topic keywords, if not given explicitly by the user, can be extracted from the user profile or from the queries submitted by the user to search engines during the day.

The results of a search are sent to different modules based on the origin of the query. If the query comes from the user, the results are sent first to the Combinator module to be merged with hits obtained from other peers, and they are also sent to the Neighbor Information Module to assist in evaluating neighbors' responses to that query. If the query comes from another peer the results will be sent back to that peer by the Neighbor Information and Communication modules.

The **Neighbor Information module** handles how a peer responds to the others, which includes evaluating qualities of each neighbor and determining which known peers are best neighbors for sending or forwarding a particular query. The module contains a database that stores known peer information, which is continually updated according to the algorithm, which is described earlier, each time that a response is received. The module also handles how much information is provided in response to neighbor requests for a peer profile.

As mentioned earlier, the evaluation of a new peer begins with a description received from that peer. Since Nutch provides an interface for retrieving the highest frequency terms from a search index, we use this as a simple way for a peer to create its own profile, to be sent to other peers upon request. This is done by extracting the most frequent terms from the local index database.

When the Neighbor Information Module receives a query, whether from a user or from other peers, it dynamically selects a set of neighbors from its database of known peers, based on the query. The N_n parameter can be set by the user to limit the maximum number of neighbors to whom the module can forward queries.

The **Combinator module** combines and re-ranks the hits obtained from the Local Search Module with those contained in responses received from peer neighbors.

The **Communication module** acts as the interface between the peer application and the peer network layer. It is responsible for all communication with other peers. The tasks of this module include passing queries, results and other messages between the other modules of the local peer and the external peers.

EXPERIMENTAL SETUP

To analyze the behavior of 6S peer network interactions, we created a simulator to do two different types of simulations that allow us to model synthetic users and run their queries over real indexes obtained from actual distributed Web crawls. The goal of the simulator in our experiments outlined below is to study the statistics of 6S's emergent peer network topology and the feasibility of the 6S framework using large peers and using a large number of peers.

Our simulator takes a snapshot of the network for every time step. In a time step of the simulator, all of the peers process all of their buffered incoming messages and send all of their buffered outgoing messages. This may include the generation of a local query as well as responding to the queries received by other peers and forwarding them. The pseudo code for the simulator is shown in Figure 4.

There are $N = 70$ peers in our first simulation experiment and $N = 500$ peers in our second simulation. In order to study whether the adaptive routing algorithm of the 6S network can generate network topologies that capture the interests shared by user communities, thus reducing query flooding problems, we modeled synthetic users belonging to 7 (for the first simulation) and 50 (for the second simulation) different groups of 10 users each. Each group is associated with a general topic. Each peer has its own search engine database, but for the peers in a given group the search engines are built by topical crawlers focusing on the same topic. For example if a group's topic is "sports", then all the peer search engines in this group focus on different aspects of sports. Two points are to be emphasized here. First, while the simulated peers in this experiment are associated with relatively narrow topics, this is not a 6S general requirement; peer topics can have arbitrary generality matching single users or communities of users. Second, while we simulate these communities to see if the peer network can discover them, any individual peer has no more knowledge about other peers in its group than about all other peers.

Group topics are chosen from the Open Directory (ODP, <http://dmoz.org>) to simulate the group structure, according to a simple methodology developed to evaluate topical crawlers (Srinivasan et al., 2005). The topics corresponding to the groups in our first simulation (with 70 peers) are shown in Table 1. For each group, we extract a set of 100-200 URLs from the ODP subtree rooted at the category node corresponding to the group's topic. Random subsets are assigned to the peer crawlers as seeds. So the search engines within each group differ from each other according to the different sets of crawled pages (starting from different sets of seeds). We use the same strategy to setup our second simulation but instead of 7 topics we choose 50 different topics.

No.	Topic keywords
1	Science/Environment/Products and Services
2	Science/Astronomy/Software
3	Science/Social Sciences/Archaeology
4	Shopping/Home and Garden/Cleaning
5	Sports/Tennis/Players
6	Recreation/Boating/Boatbuilding
7	Computers/Software/Freeware

Table 1: The seven ODP topics used to model communities of users with shared interests in our first simulation.

Given a set of topic keywords and a set of seed URLs, the best-N-first crawler was run offline for each peer to harvest the pages that would be indexed to build the peer's search engine. For our first simulation, we crawled around 10,000 Web pages for each peer (for a total of 700,000 pages). And for our second simulation, we crawled around 1,000 Web pages for each peer (for a total of 500,000 pages). The Nutch package was then used to index these pages and build each peer's search engine.

Each peer is allowed to know about all of the other peers ($N_k = 69$ for first simulation and $N_k = 499$ for second simulation) and to have $N_n = 5$ neighbors. At the beginning of each experiment, the peer network is initialized as a random Erdos-Renyi graph, i.e., each peer is assigned 5 random neighbors drawn from a uniform distribution, irrespective of groups.

Each peer in our experiments has 10 queries as its own local queries. The queries are related with the peer's group topic. The queries used in our experiments are 3-5 word strings such as "environmental products services" and "manufacturing selling system parts." The queries for each peer in our first simulation were generated by randomly picking keywords from the ODP descriptions of the Web sites whose URLs were used as seeds for the peer's crawler. The queries for each peer in our second simulation were generated by extracting the title words of a Web site. If a title had more than 5 words, then we randomly picked 5 words from the title as a query. The peer that has a local query from a certain Web site and the peer that used the URL of this Web site as a seed for the peer's crawler must belong to the same group. Since we have 10 peers in one group and 10 queries per peer, we have 100 queries per group and a total of 700 and 5,000 queries in the first and second simulation respectively.

Finally we set the profile learning rate to $\gamma = 0.3$ (Equation 1), the profile reliability parameter to $\alpha = 0.8$ (Equation 2), and the TTL to 3. We ran the simulator for about 10,000 time steps for the first simulation (corresponding to 1,000 queries issued per peer) and 1,200 time steps for the second simulation (corresponding to 120 queries issued per peer). Since there are only 10 distinct queries per peer, each query is submitted several times in the course of a simulation. In these simulations the peers have static content, as only one crawl takes place per peer. Therefore it is not necessary to request a peer's profile more than once.

Our first experiment was performed on IU's AVIDD-B Linux cluster with 208 2.4-GHz Prestonia processors using a General Parallel File System and a gigabit Ethernet connection to Abilene Internet2 and Internet. Each 10,000 page crawl took less than 1 hour. The 70 crawls could be run in parallel. A complete simulation run took approximately 6 hours. Our second experiment was distributed over 5 dual 2.8-GHz Linux machines, each running 100 peers. A complete simulation run took approximately 24 hours.

```

Initialize all peers
Initialize simulated network at random
While not terminated
  For each peer
    If user submits a query
      Process query on local search engine
      Send query to appropriate neighbors
    EndIf
    If a response to a previous query is received
      If response is for local user
        Evaluate neighbor
        Combine hits with other hits
        Output to user
      Else
        Forward response to query sender
      EndIf
    EndIf
    If a query is received
      Process query on local search engine
      Send response back
      If TTL > 0
        Decrease TTL
        Forward query to appropriate neighbors
      EndIf
    EndIf
    If a request for profile is received
      Generate profile
      Send profile to requester
    EndIf
  EndFor
EndWhile

```

Figure 4: Simulator pseudocode

ANALYSIS FOR FEW LARGE PEERS

Let us analyze the results obtained from the first simulation, in which we model a relatively small network with relatively large peers (i.e., indexing relatively large crawl sets). Here we consider only the simple learning algorithm for query routing, while later in this chapter we consider the richer profile representation with query expansion and the soft update rule.

Emerging Network Topology

With the purpose of showing the variation of the network topology at different simulation time steps, we need to introduce two network statistics, the *cluster coefficient* and the *diameter*. The cluster coefficient for a node is the fraction of a node's neighbors that are also neighbors of each other. This was computed in the directed graph based on each peer's N_n neighbors. Thus, in our simulation, with $N_n = 5$, the total number of possible directed links between neighbors is $N_n(N_n - 1) = 20$. The overall cluster

coefficient C is computed by averaging across all peer nodes. The diameter D is defined as the average shortest path length ℓ across all pairs of nodes. Since the network is not always strongly connected, some pairs do not have a directed path ($\ell = \infty$). To address this problem, we use the harmonic mean of shortest paths:

$$D = \left(\frac{N}{N-1} \sum_{ij} \ell_{ij}^{-1} \right)^{-1} \quad (3)$$

where N is the number of nodes. The diameter D thus defined can be computed from all pairs of nodes irrespective of whether the network is connected. C and D are measured at each time step in a simulation run

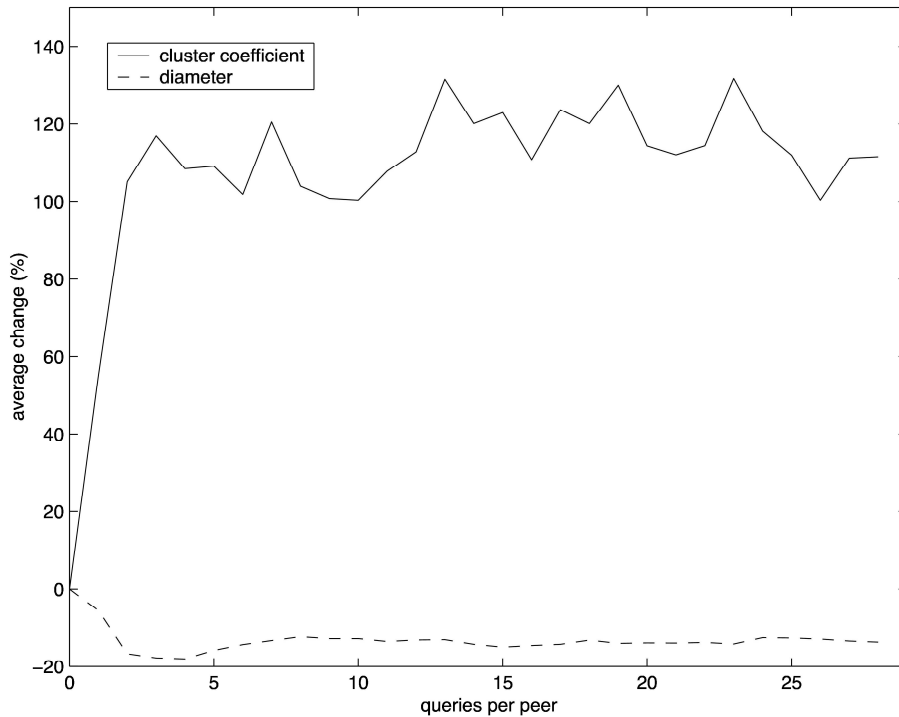


Figure 5: Small world statistics of the 6S peer network

Figure 5 shows that the 6S diameter remains roughly equal to the initial random graph diameter, while the cluster coefficient increases very rapidly and significantly, stabilizing around a value twice as large as that of the initial random graph after only 5 queries per peer. These conditions define the emergence of a small world topology in our peer network (Watts & Strogatz, 1998). This is a very interesting finding, indicating that the peer interactions cause the peers to route queries in such a way that communities of users with similar interests cluster together to find quality results quickly, while it is still possible to reach any peer in a small number of steps.

To illustrate the small world phenomenon, Figure 6 shows the dynamics of the peer network topology. We see the change both for the whole network and for the neighborhood of a single group (corresponding to Topic 7 in Table 1). The 10 nodes corresponding to peers in this group are placed around the 8 o'clock position. On the left

we see the initial random connections; on the right we see the connections in the final network. One can observe that there are more local (within group) links and fewer long (cross-group) links on the right-hand-side, revealing the emergence of local clusters in the network topology as the semantic locality is discovered among peers.

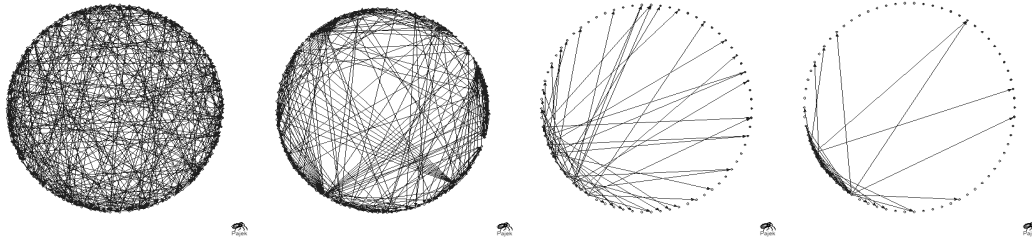


Figure 6: Peer network connectivity for all groups (left 2s) and for one of the groups (right 2s). Left: initial neighbor links. Right: final neighbor links.

Quality of Results

In order to compare the performance of the 6S network approach with the traditional centralized search engine approach, we need to evaluate the quality of results obtained through 6S, and compare them to the results obtained from centralized search engines based on the same queries. We build a centralized search engine using the same amount of network resources as a 6S run; we crawled and indexed 700,000 pages from the same seeds but using a traditional (breadth-first) crawler rather than a topical crawler. We issue the same queries used for 6S and collect 100 top hits for each query.

We want to use precision-recall plots as a tool to compare the performance between different types of search engines. To calculate precision and recall values it is necessary to obtain a relevant set of pages for each query. As a context for relevance, we must consider that queries are submitted in our simulation by model users. To capture the users' relevance contexts we extend each of the 700 peer queries with a single most frequent term from the profile of the peer submitting the query. Each extended query is submitted offline to a separate centralized search engine, built just for evaluation purposes, that combines the 70 peers' search engine databases; the top 100 hits returned are used as the relevant set of each query. For example, to get the relevant set of peer 4's query "environmental products services," we extend the above query with the most frequent term "health" in peer 4's local search engine database. So the query used to obtain the relevant set is "environmental products services health." Note that we are not granting 6S an unfair advantage because these profile terms used to obtain relevant sets are not used by 6S peers when processing queries.

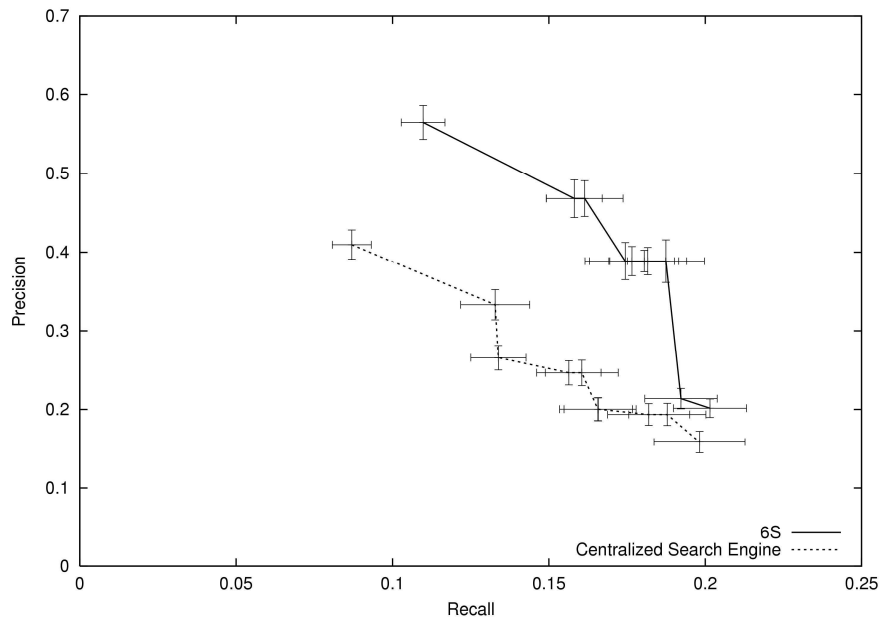


Figure 7: Precision-recall plots for 6S and the centralized search engine. Error bars correspond to standard errors of precision and recall

Figure 7 shows the precision-recall plots comparing quality of results by 6S and the centralized search engine. 6S significantly outperforms the centralized search engine. This occurs because of the collaboration among peers – queries are successfully routed to those peers who can return highly relevant hits owing to their stronger focus relative to user interests.

Figure 8 shows that performance improves as peers learn to route queries to the appropriate neighbors, and as the number of hits N_h that peers return in response to queries increases. Performance is measured by the F-measure, which combines precision and recall through their harmonic mean. The relatively small improvement due to the simple learning algorithm motivated our design of more sophisticated adaptive query routing schemes, and suggests that most of the advantage enjoyed by this version of 6S (cf. Figure 7) is due to focused coverage rather than to the rudimentary learning algorithm. The effect of N_h is larger; more communication can only improve performance – in the limit of complete communication, the network would combine all the focused crawls in a centralized fashion. However, there is a cost associated with communication: network traffic grows linearly with N_h . Yet, as N_h goes from 5 to 10 and traffic doubles, performance improves by less than a factor of 2. More experiments are needed to study the trade-off between network traffic and search effectiveness.

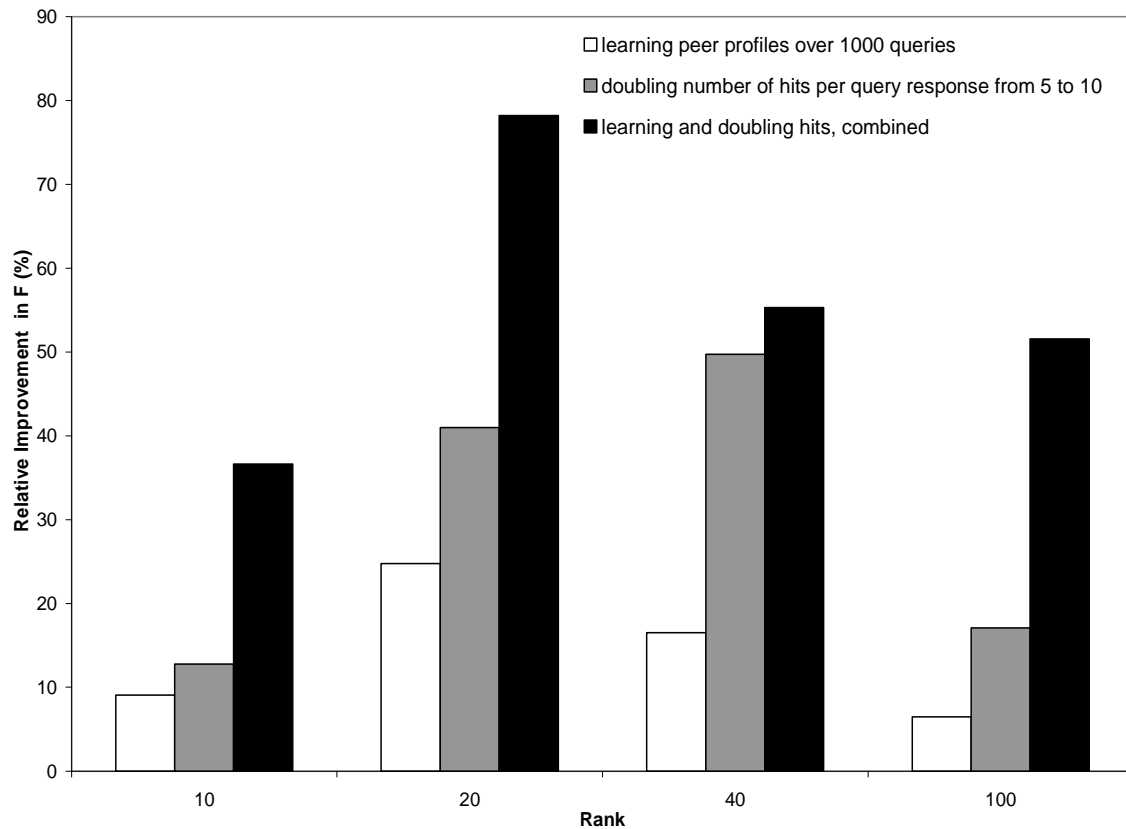


Figure 8: Relative improvement in F-measure due to query routing based on simple learning and to increasing the number of hits per query response N_h from 5 to 10, plotted versus the total number of top hits considered. The gray bars measurements are taken at the beginning of the simulation.

ANALYSIS FOR MANY SMALL PEERS

Let us analyze the results obtaining from the second simulation, in which we model a larger network (one order of magnitude more peers) with relatively light-weight peers (one order of magnitude smaller crawls). Here we focus on the evaluation of the different learning algorithms supporting query routing. We also want to see if peers with similar interests can still find each other even though the network is much larger and the ratio of related peers is smaller compared to the first simulation ($1/50$ rather than $1/7$). Finally we will compare the performance of 6S with that of a real-world centralized search engine, namely Google.

Emerging Network Topology

Even with larger network size, our experiment shows that with adaptive query routing, a peer can still quickly find another peer with the similar focus. Due to the smaller ratio between the size of peer groups and the size of the network, the clustering coefficient does not grow appreciably from its random network value in this simulation. Yet we observe in Figure 9 that the average fraction of neighbors that are in the same

interest group as a peer increases significantly and rapidly (within 6-7 queries issued) with time. At regime, 30% of a peer's neighbors belong to the same group as the peer on average. This shows that even with a larger network the topology evolves to match the content locality among peers. We also find (not shown in Figure 9) that the expanded neighbor profile and soft learning rule for neighbor profile update each contribute to increasing the ratio of connections within groups, improving the locality of the network.

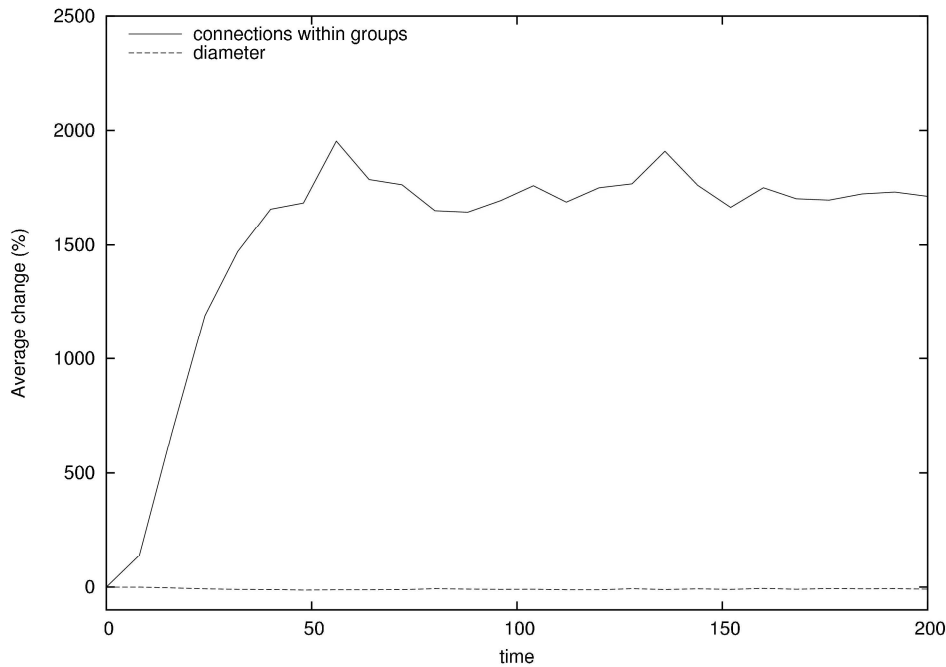


Figure 9: Average fraction of neighbors that are connected to peers in the same group as themselves, and diameter of peer network versus time. Time is measured in number of simulation steps, and one query is issued by each peer every 8 steps. So 200 steps are equivalent to 25 queries per peer.

Quality of Results

To evaluate the query routing algorithm, here we use two baseline query routing algorithms that do not employ the expanded profile W^e . (1) *simple* updates W^f by replacing w_{ip}^f with the best hit score from p ; (2) *soft update* uses W^f with the update rule in Equation 1. The relevant sets in the second simulation are simply the sets of URLs classified by the ODP under the same topic as the page whose title is used as query. We show precision-recall snapshots in Figure 10. The snapshots are made at time steps 8, 504 and 1000. Already at the start we observe a difference in performance between the learning algorithms. One might be surprised by such a difference after the first query since all peers in each simulation begin with empty profiles. However, during the 4 time steps the first query took to propagate (it can only travel as far as half the round trip)

adaptive peers in the query path had already learned about their neighbors, hence they could better forward the query.

Besides showing that all query routing schemes take advantage of the learning and improve their performance over time, Figure 10 also confirms that the more sophisticated learning algorithms outperform the simpler ones, with the best performance achieved by combining expanded profiles and the soft profile update rule.

As a last analysis we wanted to compare the quality of the average results obtained by 6S peers with those returned by a real-world search engine. To this end we queried the Google Web API. As a summary performance measure we employed the commonly used average precision at 10, $\langle P_{10} \rangle$. As shown in Table 2, the difference in performance between the two systems is not statistically significant, suggesting that 6S can be competitive with much larger search engines – the number of pages indexed by Google is about 104 times larger than those of the entire 6S network in our simulation.

	$\langle P_{10} \rangle$	$\sigma_{\langle P_{10} \rangle}$	95% Confidence Interval
Google	0.079678	0.00095	(0.0778, 0.0816)
6S	0.078380	0.00062	(0.07714, 0.07962)

Table 2: Average precision at 10 of Google and 6S.

The comparison with Google must be interpreted carefully. The pages used as relevant sets in this experiment (ODP pages) are well known to Google, and using their titles as queries allowed Google to retrieve and rank very highly the pages with those titles. However, 6S peers can exploit their context and share their knowledge via collaboration during the search process, while Google has a single, universal ranking function and cannot exploit such context. Thus Google did not rank as highly pages that our model users considered relevant because highly related to the page used to compose the query. Another factor to be considered is that Google may have returned other relevant pages which were not in our relevant sets; our automatic assessment methodology would not allow us to give credit for those. Despite this caveat, we find the comparative result very encouraging.

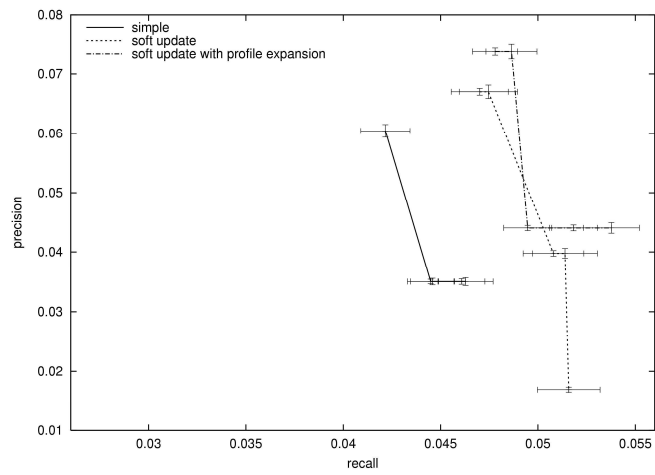
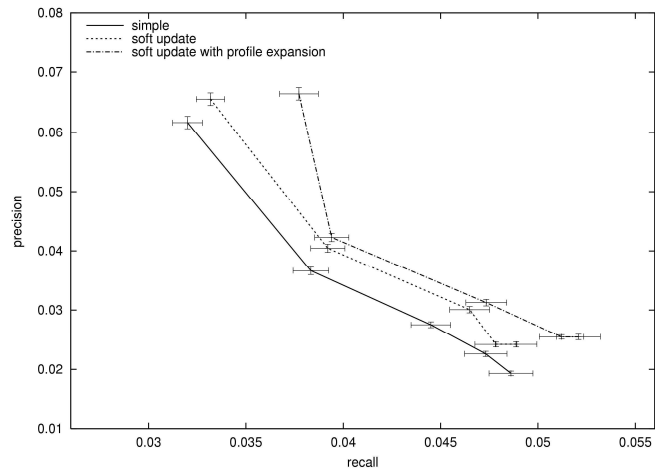
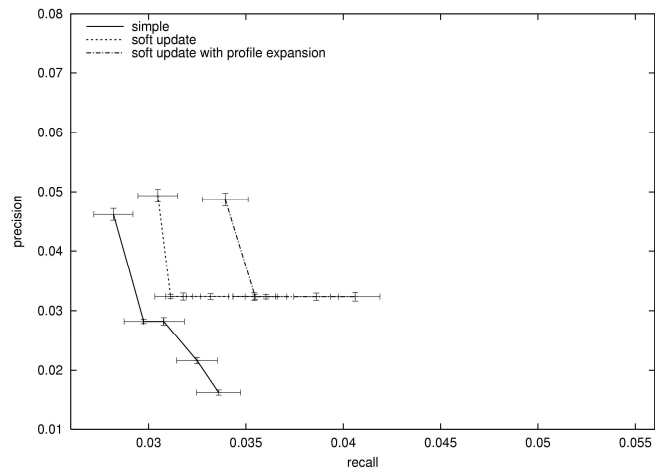


Figure 10: Precision recall plots for three learning schemes, taken at the start of the simulation (top), after 504 time steps or 63 queries per peer (middle), and after 1000 time steps or 125 queries per peer (bottom).

ALTERNATIVE EVALUATION

In the evaluation discussed earlier, we plot precision versus recall, a standard technique in information retrieval, in order to evaluate the quality of results obtained through either different search engines or different query routing algorithms. But the precision-recall plots for search evaluation on real Web data have a drawback, which is the construction of the relevant sets for queries. The most intuitive way for generating the relevant set of a query is using human assessment. People can make their decisions about whether a Web page is relevant to a query or not after viewing the content of the page. But it is impossible for people to access all the Web pages in the Internet. To overcome this recall problem, we used two different approaches to construct the relevant set of queries in simulations of a 6S P2P network with few large peers and a 6S P2P network with many small peers. In our first simulation, we appended a user's context term to the original query, and then submitted this modified query into a centralized search engine which combined all of the network peers' knowledge. We picked the top 100 hits returned by the centralized search engine as the relevant set for this query. In our second simulation, the relevant set of a query was the set of URLs classified by the ODP under the same topic as the page whose title was used as that particular query. These two approaches were designed for a fair evaluation that would in one case take context into account, in the other use an independent source of relevance assessments (the ODP). However, it is difficult to eliminate all bias (either in favor of a centralized search engine or in favor of 6S) when comparing two completely different search paradigms. For example, the number of Web pages indexed by Google is much larger than those in the 6S system. So when doing the precision and recall computation, some query results returned by the centralized search engine might be classified as irrelevant simply because they are not in the predefined relevant set of our experiment, even if they were actually relevant.

Leake et al. (2005) introduces two novel criterion functions for evaluating retrieval performance: *global coherence* and *coverage*. These two functions generalize the well known IR measures of precision and recall. However, in contrast to precision and recall, the measures of global coherence and coverage do not require that all relevant resources be precisely identified. Instead, these measures are applicable as long as an approximate description of the potentially relevant material is available.

Let us review the definitions of *global coherence* and *coverage*. Assume $R = \{r_1, \dots, r_m\}$ is a set containing approximate descriptions of potentially relevant material, where each r_i is a collection of keywords. Let $A = \{a_1, \dots, a_n\}$ be the set of retrieved resources, with a_i also represented as a collection of keywords. A measure of *similarity* between a retrieved resource a_i and a relevant resource r_j can be computed using, for example, the *Jaccard coefficient*, defined as:

$$\text{Similarity}(a_i, r_j) = \frac{|a_i \cap r_j|}{|a_i \cup r_j|} \quad (4)$$

Then, the *accuracy* of resource a_i in R is defined as follows:

$$Accuracy(a_i, R) = \max_{r_j \in R} Similarity(a_i, r_j) \quad (5)$$

The accuracy of a retrieved resource a_i provides an estimate of the precision with which the keywords in a_i replicate those of relevant resources.

Once the *Accuracy* of each retrieved result has been computed, it can be used to obtain a measure of *Global_Coherence* as follows:

$$Global_Coherence(A, R) = \frac{\sum_{a_i \in A} Accuracy(a_i, R)}{|A|} \quad (6)$$

The *Global_Coherence* function measures the degree to which a retrieval mechanism succeeded in keeping its focus within the theme defined by a set of relevant resources. This is similar to the IR notion of precision, except that we use a less restrictive notion of relevance: by using a measure of accuracy instead of considering exact matches we overcome the drawback of binary classification of relevancy.

It is important to note that a high *global coherence* value does not guarantee acceptable retrieval performance. For example, if the system retrieves only a single resource that is similar to some relevant resource, the *global coherence* value will be high. Because search mechanisms should also maximize the number of relevant resources retrieved, we introduce a *coverage* factor to favor those strategies that retrieve many resources similar to a target set of relevant resources. We define a criterion function able to measure *coverage* as a generalization of the standard IR notion of recall:

$$Coverage(A, R) = \frac{\sum_{r_i \in R} Accuracy(r_i, A)}{|R|} = \frac{\sum_{r_i \in R} \max_{a_j \in A} Similarity(a_j, r_i)}{|R|} \quad (7)$$

A performance evaluation based on our criterion functions requires access to a set of terms taken to characterize potentially relevant resources (a target set R) for a given query. For our task we used the ODP directory to construct relevant sets as follows.

Let t_1, \dots, t_m be third level topics in the ODP directory and let q_1, \dots, q_m be m queries associated with these topics. With the aim of constructing a relevant set R_i for each query q_i , we extract the descriptions of URLs from the ODP subtrees rooted at the topic t_i . Each $r \in R_i$ is then defined as a set of keywords extracted from these descriptions and it represents a potentially relevant result for query q_i .

To verify whether the *global coherence* and *coverage* measures can be used as performance evaluation tools, we conducted a preliminary experiment. The goal of this experiment was to compare the *global coherence* and *coverage* measures applied to a set of *On-Topic* results (i.e., results focused on the topic under consideration) against the performance measures applied to a *Random* set of results. We expected our performance evaluation measures to return significantly higher values for the *On-Topic* set than for the *Random* one. In this experiment, we used the same $m = 50$ ODP topics from our second simulation and applied the procedure described above to construct the relevant set R . For a given topic, the *On-Topic* retrieved set ($A_{On-Topic}$) was created using 10 URLs within that topic subtree in the ODP directory. To construct the *Random* retrieved set (A_{Random}), we used a method similar to the one used to construct the *On-Topic* set but, instead of extracting URLs from the subtree under the relevant topic, we randomly selected 10 URLs from the whole ODP directory. Finally, our performance evaluation framework was validated by comparing the *global coherence* and *coverage* of the *On-Topic* and *Random* retrieved sets. The results, which are included in Table 3 (together with an evaluation for 6Search performance to be described next), show that the *On-Topic* retrieved set truly has significantly better performance than the *Random* retrieved set both in terms of *global coherence* and *coverage*. This outcome also indicates that the *global coherence* and *coverage* for the *On-Topic* and *Random* sets are feasible upper and lower bounds for measuring the performance of a search system.

<i>Global Coherence</i>	μ	σ	95% Confidence Interval
Random	0.036	0.016	(0.032,0.041)
6S	0.052	0.020	(0.047,0.058)
On-Topic	0.063	0.020	(0.057,0.068)
<i>Coverage</i>	μ	σ	95% Confidence Interval
Random	0.015	0.006	(0.013,0.016)
6S	0.024	0.012	(0.020,0.027)
On-Topic	0.033	0.013	(0.030,0.037)

Table 3: Average and standard deviation of *Global Coherence* and *Coverage* of *On-Topic*, *6S*, and *Random* across 50 topics.

Let us now apply this evaluation approach for assessing the performance of the 6S system. To this end, we selected one query from each group in our second simulation (for a total of $m = 50$ queries) and considered the top $N = 10$ hits retrieved by the 6Search system. Table 3 shows that the quality of the results returned by the 6S system is significantly better than the *Random* baseline both in terms of *global coherence* and *coverage*.

DISCUSSION

In this chapter we introduced a collaborative peer network application called 6Search, with which we intend to study the idea that the scalability limitations of centralized search engines can be overcome via distributed Web crawling and searching. We also described adaptive routing algorithms to dynamically change the topology of the peer network based on commonality of interests among users, so as to avoid the problem of flooding queries which has plagued other attempts to search over peer networks. The results presented here seem to support the idea that adaptive routing can work with real data and that critical network structure can emerge spontaneously from the local interactions between peers, capturing the locality of content interests among them. Our experiments also suggest that 6Search can outperform centralized search engines, which cannot take advantage of user context in their crawling and searching processes.

One can observe a sharp drop in precision as recall increases (Figures 7 and 10), which corresponds to the drop in F-measure as each peer considers more hits (Figure 8). The reason is that each neighbor contributes a small number N_h of hits, so in order to increase recall a peer must consider a larger pool of neighbors, some of which may belong to different topical communities.

One of the challenges in effectiveness comparison is how to evaluate different systems such as 6S and centralized search engines in an unbiased way. By our preliminary experiment results, we have shown that the *global coherence* and *coverage* are promising approaches to compare the performance of different systems. These measures confirm that 6S can provide users with relevant results.

Let us briefly discuss redundancy of coverage. We believe that minimizing overlap between pages indexed by peers is neither desirable nor practical. Clearly one would not want all peers to be identical, but this is a very unlikely scenario; peers will be driven by user profiles built from their daily queries, their stored documents, their bookmarks, etc. Such profiles will generate heterogeneous profiles and lead to broad coverage of the Web. Redundancy will likely occur for popular pages likely to be of interest to a large number of people. This kind of redundant coverage is good for both performance (local data yield faster results) and robustness (duplication ensures availability).

FUTURE WORK

We are currently extending the evaluation with *global coherence* and *coverage* to 5,000 queries to better quantify the performance of our 6S system. We will also repeat the same computation for the results returned by Google to compare the distributed and centralized approach.

As a project in its infancy stage, 6S has many directions for further development. One technique proposed in a semantic Web setting where peers query for RDF data (Tempich et al., 2004) that we intend to test for Web searching is query relaxation, whereby a peer assumes that a neighbor may have knowledge about a topic/query if it has knowledge about a more specific version of the topic/query. While our application is

arguably more difficult due to the unstructured nature of generic Web pages, we hope that the promising scalability results obtained for semantic Web data will generalize to Web IR.

A number of improvements and extensions of the 6S network architecture and protocols are under consideration. Additional IR techniques such as various lexical similarity functions and term weighting schemes can be applied, as well as richer representation for profiles (e.g., LSI (Deerwester et al., 1990)).

A robust algorithm is to be developed for combining hits from peers in the Combinator Module, thus allowing for heterogeneous scoring by peer search engines. Strategies based on semi-supervised learning have proven effective for merging results in hierarchical peer networks, where peers can aggregate query-based document samples from neighbors into centralized (hub) databases (Lu & Callan, 2004). In a framework like 6S this may be possible to a limited extent as we do not require special hubs. We are designing an appropriate randomized ranking function to allow for probabilistic updates of peer profiles.

Additional learning algorithms will be analyzed for adaptive query routing. For example, one could mine the streams of queries and responses that are forwarded through a peer. In the Gnutella v0.6 file sharing network, peers tend to issue queries that are very similar to their own content (Asvanund et al., 2003). This suggests that a profile should be updated based on queries in addition to query responses. Another possibility is to extend the 6S protocol by including requests for profiles of a neighbor's neighbors. Several promising heuristics for adaptive query routing proposed in the literature (Crespo & Garcia-Molina, 2002; Kalogeraki et al., 2002; Yang & Garcia-Molina, 2002) will be explored. Referral should also be investigated as an alternative mechanism for adapting the network topology based on local reinforcement interactions (Singh et al., 2000). Finally, we plan to study the use of reinforcement learning algorithms for identifying good neighbors from not only their individual performance but also that of their neighborhoods.

In parallel with the above algorithmic extensions, implementation of a working 6S server (Server + Client) application is under way. Figure 11 offers a view of the current prototype's user interface. We are developing a prototype based on the JXTA framework (Waterhouse, 2001), which will integrate the 6S protocol, topical crawler, document index system, search engine system and network communication system; we plan to release the prototype to the open-source community. Testing the peer communication protocols in real environments over TCP/IP will allow us to study the robustness of the system from a security standpoint, e.g., with respect to DoS attacks. Can malicious users gain unfair advantage or disrupt the network? Most importantly, the prototype is necessary in order to move from simulated to real users in the evaluation of the proposed approach. For example, it would not be sufficient to simply test the system on real queries that are publicly available because these are not labeled or associated with particular users, and therefore do not capture the relationships that exist between different users. Peer collaborative Web search is based on real users driving the interaction between peers so that the network can discover, form, and leverage communities of users with common interests. Testing the prototype in a realistic setting will also allow us to

tune our protocols and algorithms. For example, while a peer may decide not to share its knowledge with other peers, we will consider whether the information available to a peer should be dependent on the information it is willing to share. Finally, JXTA provides for mechanisms to bootstrap a peer into the network. Simple mechanisms employed by many file sharing peer networks rely on a registry for first joining the network. An advantage of our approach is that adaptive query routing should rapidly adjust the connections of the new peer and prevent overload on the registry.

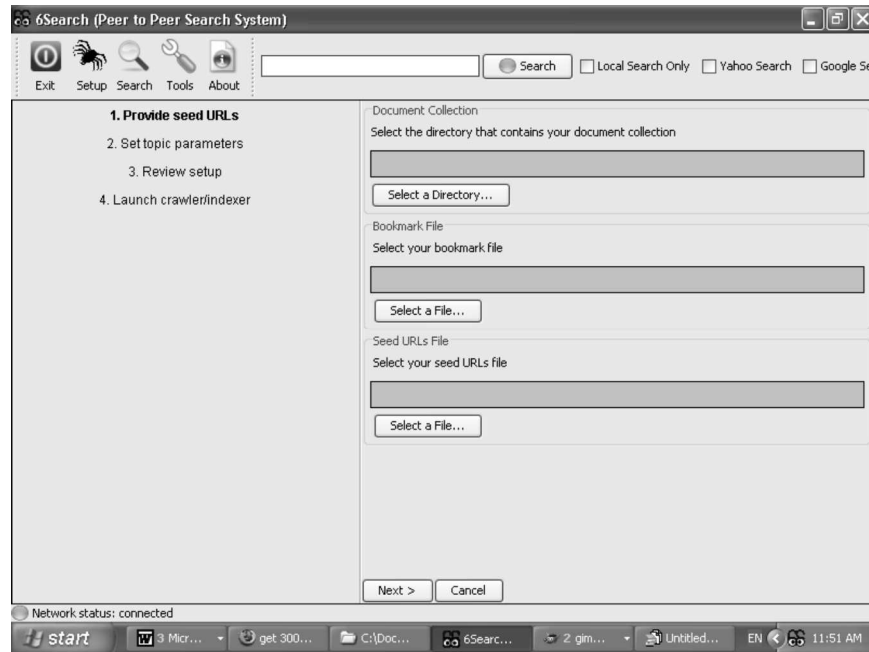


Figure 11: A screen shot of the 6S application.

REFERENCE

- Akavipat, R., Wu, L.-S., & Menczer, F. (2004) *Small World Peer Networks in Distributed Web Search*. Paper presented at the Alt. Track Papers and Posters 13th International World Wide Web Conference, New York, NY, USA.
- Asvanund, A., Bagala, S., Kapadia, M., Krishnan, R., Smith, M., & Telang, R. (2003) *Intelligent Club Management in P2P Networks*. Paper presented at the Workshop on Economics of Peer to Peer Systems, Berkeley, CA, USA.
- Bawa, M., Jr, R. B., Rajagoplan, S., & Shekita, E. (2003) *Make it Fresh, Make it Quick -- Searching a Network of Personal Webservers*. Paper presented at the 12th International World Wide Web Conference, Budapest, Hungary.
- Brewington, B. E., & Cybenko, G. (2000) *How dynamic is the Web?* Paper presented at the 9th International World Wide Web Conference, Amsterdam, Netherlands.
- Brin, S., & Page, L. (1998) The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks* 30(1-7), 107-117.

- Chakrabarti, S., Berg, M. v. d., & Dom, B. (1999) Focused Crawling: A new approach to Topic-Specific Web Resource Discovery. *Computer Networks* 31(11-16), 1623-1640.
- Cho, J., & Garcia-Molina, H. (2002) *Parallel crawlers*. Paper presented at the 11th International World Wide Web Conference, Honolulu, Hawaii, USA.
- Crespo, A., & Garcia-Molina, H. (2002) *Routing Indices for Peer-to-Peer Systems*. Paper presented at the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria.
- Dean, J., & Ghemawat, S. (2004) *MapReduce: Simplified Data Processing on Large Clusters*. Paper presented at the 6th Symposium on Operating System Design and Implementation (OSDI04), San Francisco, CA.
- Deerwester, S., Dumais, S., GW, F., Landauer, T., & Harshman, R. (1990) Indexing By Latent Semantic Analysis. *Journal of the American Society for Information Science* 41, 391-407.
- Fetterly, D., Manasse, M., Najork, M., & Wiener, J. (2003) *A Large-Scale Study of the Evolution of Web Pages*. Paper presented at the 12th International World Wide Web Conference, Budapest, Hungary.
- Joseph, S. (2002) *NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks*. Paper presented at the International Workshop on Peer-to-Peer Computing, Pisa, Italy.
- Kalogeraki, V., Gunopulos, D., & Zeinalipour-Yazti, D. (2002) *A Local Search Mechanism for Peer-to-Peer Networks*. Paper presented at the 11th International Conference on Information and Knowledge Management (CIKM'02), McLean, VA, USA.
- Kamvar, S., Schlosser, M., & Garcia-Molina, H. (2003) *The EigenTrust Algorithm for Reputation Management in P2P Networks*. Paper presented at the 12th International World Wide Web Conference, Budapest, Hungary.
- Lawrence, S., & Giles, C. (1999) Accessibility of information on the Web. *Nature* 400, 107-109.
- Leake, D., Maguitman, A., & Reichherzer, T. (2005, July) *Exploiting Rich Context: An Incremental Approach to Context-Based Web Search*. Paper presented at the International and Interdisciplinary Conference on Modeling and Using Context, CONTEXT'05, Paris, France.
- Li, J., Loo, B., Hellerstein, J., Kaashoek, F., Karger, D., & Morris, R. (2003) *On the feasibility of peer-to-peer web indexing and search*. Paper presented at the 2nd International Workshop on Peer-to-Peer Systems, Berkeley, CA, USA.
- Lu, J., & Callan, J. (2003) *Content-based retrieval in hybrid peer-to-peer networks*. Paper presented at the 12th International Conference on Information and Knowledge Management (CIKM'03), New Orleans, LA, USA.
- Lu, J., & Callan, J. (2004) *Merging retrieval results in hierarchical peer-to-peer networks*. Paper presented at the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Sheffield UK.

- Lua, E. K., Crowcroft, J., & Pias, M. (Second Quarter 2005) A Survey and Comparison of Peer-to-Peer Overlay Network Schemes *IEEE Communications Surveys and Tutorials* 7(2).
- Menczer, F., & Belew, R. (2000) Adaptive Retrieval Agents: Internalizing Local Context and Scaling up to the Web. *Machine Learning* 39(2-3), 203-242.
- Menczer, F., Pant, G., & Srinivasan, P. (2004) Topical Web Crawlers: Evaluating Adaptive Algorithms. *ACM Transactions on Internet Technology* 4(4), 378-419.
- Najork, M., & Wiener, J. L. (2001) *Breadth-first search crawling yields high-quality pages*. Paper presented at the 10th International World Wide Web Conference, Hong Kong.
- Ntoulas, A., Cho, J., & Olston, C. (2004) *What's new on the Web?: The evolution of the Web from a search engine perspective*. Paper presented at the 13th International World Wide Web Conference, New York, NY, USA.
- Pant, G., Bradshaw, S., & Menczer, F. (2003) *Search engine - crawler symbiosis*. Paper presented at the 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Berlin.
- Pant, G., Srinivasan, P., & Menczer, F. *Crawling the Web*. *Web Dynamics* (2003).
- Pujol, J., Sanguesa, R., & Bermudez, J. (2003) *Porqpine: A Distributed and Collaborative Search Engine*. Paper presented at the 12th International World Wide Web Conference, Budapest, Hungary.
- Risson, J., & Moors, T. (2004) *Survey of Research towards Robust Peer-to-Peer Networks: Search Methods*: University of New South Wales, Australia.
- Singh, M., Yu, B., & Venkatraman, M. (2000) Community-Based Service Location. *Communications of the ACM* 44(4), 49-54.
- Srinivasan, P., Pant, G., & Menczer, F. (2005) A General Evaluation Framework for Topical Crawlers. *Information Retrieval* 8(3), 417-447.
- Suel, C., Mathur, T., Wu, J.-W., Zhang, J., Delis, A., Kharrazi, M., et al. (2003) *ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval*. Paper presented at the International Workshop on the Web and Databases (WebDB), San Diego, CA, USA.
- Tang, C., Xu, Z., & Dwarkadas, S. (2003) *Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks*. Paper presented at the ACM SIGCOMM '03, Karlsruhe, Germany.
- Tempich, C., Staab, S., & Wranik, A. (2004) *REMINDIN': Semantic query routing in peer-to-peer networks based on social metaphors*. Paper presented at the 13th International World Wide Web Conference, New York, NY, USA.
- Waterhouse, S. (2001) *JXTA Search: Distributed search for distributed networks*: Sun Microsystems Inc.
- Watts, D., & Strogatz, S. (1998) Collective dynamics of "small-world" networks. *Nature* 393, 440-442.
- Yang, B., & Garcia-Molina, H. (2002) *Improving Search in Peer-to-Peer Networks*. Paper presented at the 22nd Intl. Conf. on Distributed Computing Systems (ICDCS'02), Vienna, Austria.