

# 6S: Distributing crawling and searching across Web peers

Filippo Menczer  
School of Informatics  
& Computer Science Dept.  
Indiana University  
Bloomington, IN 47408  
fil@indiana.edu

Ruj Akavipat  
Computer Science Dept.  
Indiana University  
Bloomington, IN 47405  
rakavipa@cs.indiana.edu

Le-Shin Wu  
Computer Science Dept.  
Indiana University  
Bloomington, IN 47405  
lewu@cs.indiana.edu

## ABSTRACT

A collaborative peer network application called 6Search (6S) is proposed to address the scalability limitations of centralized search engines. Each peer crawls the Web in a focused way, guided by the user's information context. This way better (distributed) coverage can be achieved. Each peer also acts as a search servant by submitting and responding to queries to/from its neighbors. This search process has no centralized bottleneck. A local adaptive routing algorithm is introduced to dynamically change the topology of the peer network based on a simple learning scheme driven by query response interactions among neighbors. We validate a prototype of the 6S network via simulations with 70 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 also compare the quality of the results with those obtained by centralized search engines such as Google, suggesting that 6S can draw advantages from the context and coverage of the peer collective.

## Categories and Subject Descriptors

C.2.2 [Computer-Communication Networks]: Network Protocols—Applications; C.2.4 [Computer-Communication Networks]: Distributed Systems; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval; H.3.4 [Information Storage and Retrieval]: Systems and Software—Distributed systems, information networks, performance evaluation (efficiency and effectiveness); H.4.3 [Information Systems Applications]: Communication Applications

## Keywords

Peer collaborative search, distributed topical crawlers, scalability, small world networks

## 1. INTRODUCTION AND BACKGROUND

Centralized search engines have difficulty with coverage of the Web [18] because the Web is large, fast-growing [10], and dynamic [4, 12]. Even the popular press is quickly realizing that like any

retrieval system, current search engines perform poorly for queries that require context based interpretation [14]. 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).

We identify the above limitations as problems of *scale*: in spite of enormous progress in crawling, indexing, retrieval and ranking, 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 [8, 22, 23, 7, 24]. In prior work one of the authors has investigated a tightly coupled system in which a topical crawler and search engine engage in a symbiotic relationship with the crawler feeding the search engine and the search engine helping the crawler to better its performance. Such symbiosis can allow a vertical search engine to learn about its community's interests and serve such a community with better focus [26]. Topical portals however remain prone to coverage and scalability limitations, and maintain a very coarse grained level of personalization.

There is extensive work on personalization and information customization including applications to Web searching in the AI and IR literature. (There are too many examples to list here; the reader is referred to the review in [25] as one starting point.) 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. An exception is represented by work aimed at making PageRank computations more scalable, thus enabling personalized versions [13].

It is clear that distributed systems are part of the answer to the scale problem. Even “traditional” search engines employ distributed and parallel systems to handle the massive computational and storage requirements of indexing, retrieval and ranking [5]. Grub<sup>1</sup> is a client-server distributed computing effort (a'la SETI@home [33]) to distribute the crawling task across many desktop computers in the hope of achieving high coverage. While this effort is likely useful in discovering unknown Web sites, the centralized integration of crawl index data creates a huge bottleneck for searching.

Peer network are increasingly seen as a candidate framework for distributed Web search applications. One model proposed by the YouSearch project is to maintain a centralized search registry for query routing (like Napster), but enrich the peers with the capabil-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM 2004 Washington, D.C. USA

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

<sup>1</sup><http://www.grub.org>

ity to crawl and index local portions of the Web [2]. The central control in this approach unfortunately 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 [15] 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 [29]. 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 [32]. In pSearch [34] latent semantic analysis [11, 3] is performed over such distributed hash tables to provide peers with keyword search capability. This is a promising approach, however Li *et al.* argue that full-text Web search is infeasible in both the flood model and the distributed hash table model [19].

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 [20]. In hybrid peer networks, leaf nodes provide information and use content based retrieval to decide which documents to retrieve for queries.

In this paper 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 small local search engine. 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 [9, 39, 16].

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" [38]. 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" [37], we named our model *6Search* (6S).

In the remainder of the paper we illustrate the 6S model in more detail by focusing on its architecture and protocols. We also illustrate how it works by reporting on preliminary experiments using a simulated 6S network with 70 peers modeled after synthetic users. Finally we discuss the future of the project.

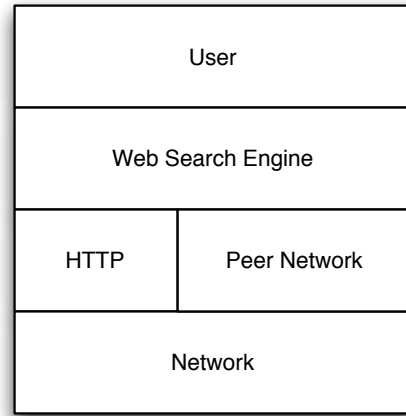


Figure 1: 6S protocol stack.

## 2. 6S PROTOCOL

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.

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 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.

### 2.1 Message primitives

We do not wish to design an overly complex protocol, which could hinder the development of improved protocols in the future. Following are the 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.

### 2.1.1 Query message

```
<Query>
:
<body></body>
<ownerid></ownerid>
</Query>
```

The query message is essential for a peer to be able 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.

### 2.1.2 Query response

```
<QueryResponse>
:
<body></body>
<ownerid></ownerid>
</QueryResponse>
```

The query response is needed for a peer to be able 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.

### 2.1.3 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.

### 2.1.4 Profile response

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

This primitive allows a peer to respond to a request for its own profile. Section 3.3 describes how a profile is generated by a peer. Such a profile initially consists of a simple list of terms. Later, as peers learn about their neighbors' expertise from the query responses they send, profiles are updated as described in Section 2.2.3.

## 2.2 Message detail and protocol algorithms

Let us briefly discuss the detail of each message primitive and algorithm in our protocol.

### 2.2.1 Query processing and response

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. Owner identification can be attached as a sign that one wants to discover new neighbors. ID and timestamp are added to help differentiate each query. The ID has to be unique only locally between two peers.

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

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:

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

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 Section 2.2.3. Then the peer will forward its neighbors' responses, if any, back to the peer who originated the query.

The purpose of TTL is to limit the spreading of a query 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.

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.

### 2.2.2 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

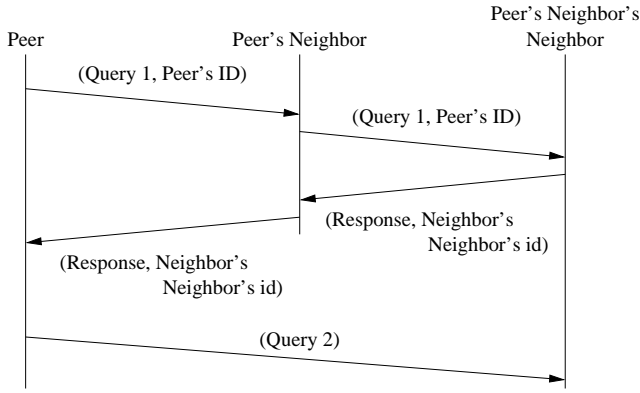


Figure 2: 6S neighbor discovery.

response with its own contact ID in the `ownerid` field of the response message. The new neighbor peers can later contact each other directly. The process is illustrated in Figure 2.

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 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.

### 2.2.3 Adaptive query routing

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 [17]. 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 vector representation for the peer profile is then initialized with the list of keywords contained in the profile response, each with unit weight.
2. Responses from neighbors (and neighbors' neighbors) are evaluated and used to update the profile of each known peer.
  - (a) The scores of the  $N_h$  hits received from each neighbor are compared with local hit scores. (We simply assume that the scoring criteria are uniform across all peers.)
  - (b) If the score of any neighbor hit is better than at least one of the top  $N_h$  local scores:
    - i. the query keywords are added to the neighbor profile vector with a weight corresponding to the score, and
    - ii. a discovery signal (`ownerid`) is sent with the next query to that neighbor.

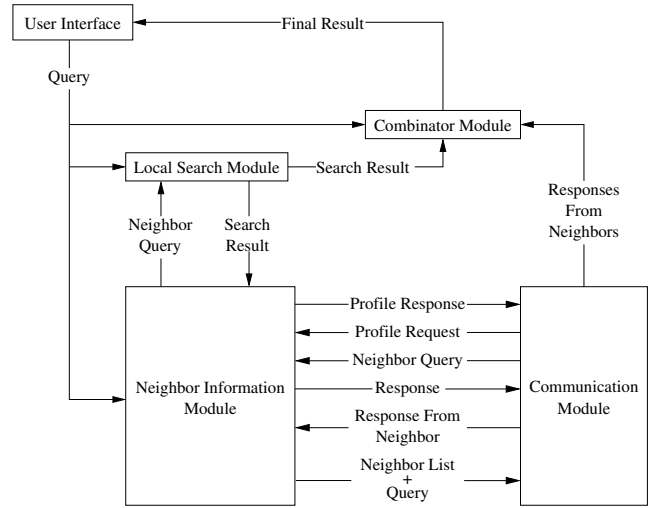


Figure 3: 6S peer architecture.

(c) New peers that respond to discovery signals are added to the list of known peers, with their profile.

3. For the next query (generated locally or forwarded), known peers are ranked by similarity between the query and the peer descriptions. A randomized ranking function is used.
4. The top  $N_n$  ranked among known peers are selected as neighbors and sent the query.
5. Goto step 2.

In the above algorithm, a profile is requested only when a peer is first discovered. However, peers are not static; they change as their index databases are updated following successive crawls. A peer may even change its focus substantially as the interests of its user base shift. This is the reason behind the randomized ranking function of peers for routing queries; a peer will be occasionally rediscovered through this mechanism, and if it yields good results, its profile can be requested again. This way we ensure probabilistic updates of peer profiles driven by query-based interactions.

## 3. 6S ARCHITECTURE

Using the protocol described in the previous section, we create the architecture of a peer as shown in the data flow of 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 seven 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.

### 3.1 User Interface module

This 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.

## 3.2 Local Search module

This 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 an open-source Web search engine, Nutch,<sup>2</sup> as the local indexing and database code for this module.

For the topical crawler we use a *best-N-first* search algorithm developed in prior work, which has proven very effective against a number of crawling algorithms in the literature [24, 27, 26, 28]. A detailed description of this crawling algorithm is outside the scope of this paper 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 idea is that 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 explorative. This crawler is also publicly available.<sup>3</sup>

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 way of the Neighbor Information and Communication modules.

## 3.3 Neighbor Information Module

This 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 in Section 2.2.3 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 in the previous section, 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.

## 3.4 Combinator module

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

## 3.5 Communication module

This module acts as the interface between the peer application

<sup>2</sup><http://www.nutch.org>

<sup>3</sup><http://informatics.indiana.edu/fil/IS/JavaCrawlers>

```
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
  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.

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.

## 4. EXPERIMENTAL SETUP

To analyze the behavior of 6S peer network interactions, we created a simulator that allows us to model synthetic users and run their queries over real indexes obtained from actual distributed Web crawls. The goal of the simulator in the preliminary experiments outlined below is to study the statistics of 6S's emergent peer network topology and compare the distributed peer framework with its centralized counterpart.

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 pseudocode for the simulator is shown in Figure 4.

There are  $N = 70$  peers in our simulation experiment. 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 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

**Table 1: The seven ODP topics used to simulate communities of users with shared interests.**

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

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<sup>4</sup> (ODP) to simulate the group structure, according to a simple methodology developed to evaluate topical crawlers [31]. The topics corresponding to the groups in the simulation are shown in Table 1. For each group, we extract a set of 100–150 URLs from the ODP subtree rooted at the category node corresponding to the group’s topic. The peers in the group use this set of URLs as the seeds of their crawlers but each peer is assigned a different set of 3–5 topic keywords to guide the crawl, selected at random from the URL descriptions in ODP. So the search engines within each group differ from each other according to the different sets of pages crawled using different topic keywords.

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. We crawled around 10,000 Web pages for each peer (for a total of 700,000 pages). The Nutch package was then used to index these pages and build each peer’s search engine.

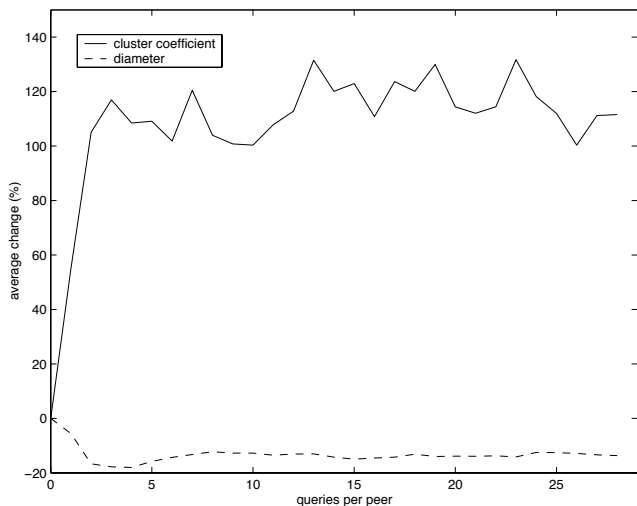
Each peer is allowed to know about all of the other peers ( $N_k = 69$ ) and to have  $N_n = 5$  neighbors. At the beginning of each experiment, the peer network is initialized as a random *Erdos-Renyi* graph, i.e., each peer is assigned 5 random neighbors drawn from a uniform distribution, irrespective 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 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. Since we have 10 peers in one group and 10 queries per peer, we have 100 queries per group and a total of 700 queries overall.

Finally we set the TTL to start at 15. This means a query can be forwarded at most 3 times from one peer (because a peer has 5 neighbors). We ran the simulator for about 10,000 time steps, corresponding to 1,000 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 this simulation the peers have static content, as only one crawl takes place per peer. Therefore it is not necessary to randomize the ranking function for query routing, nor to request a peer’s profile more than once (cf. Section 2.2.3).

Our experiments were performed on Indiana University’s AVIDD Linux cluster with 208 2.4 GHz Prestonia processors using a General Parallel File System and a gigabit Ethernet connection to Abi-

<sup>4</sup><http://dmoz.org>



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

lene 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.

## 5. ANALYSIS OF RESULTS

### 5.1 Emergent network topology

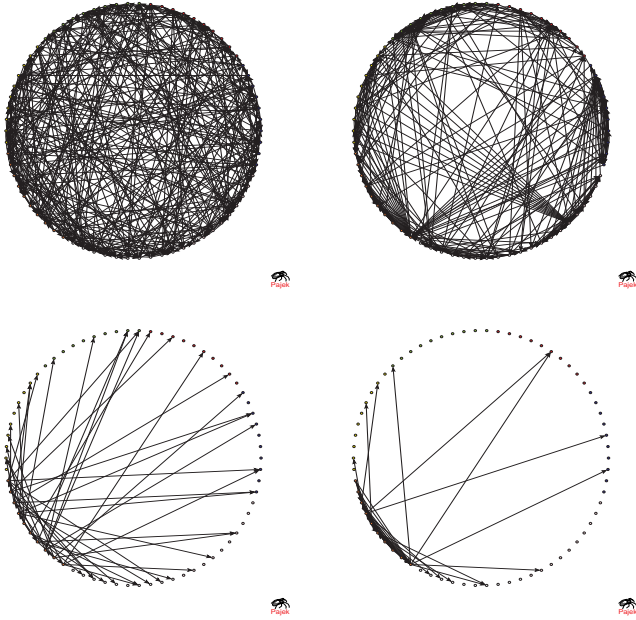
Let us define two network statistics, the *cluster coefficient* and the *diameter*. The cluster coefficient for a node is the fraction of a node’s neighbors that are also neighbors of each other. This was computed in the directed graph based on each peer’s  $N_n$  neighbors, with a total of  $N_n(N_n - 1) = 20$  possible directed links between neighbors. The overall cluster coefficient  $C$  is computed by averaging across all peer nodes. The diameter  $D$  is defined as the average shortest path length  $\ell$  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:

$$D = \left( \frac{1}{P} \sum_{p=1}^P \ell_p^{-1} \right)^{-1} \quad (1)$$

where  $p$  is a pair of nodes and  $P = N(N - 1)$  is the number of possible pairs. 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 shows that the 6S diameter remains roughly equal to the initial random graph diameter (actually there is a slight decrease), 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 [38, 37]. 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



**Figure 6: Peer network connectivity for all groups (top) and for one of the groups (bottom). Left: initial neighbor links. Right: final neighbor links. (Drawings by Pajek network visualization tool.)**

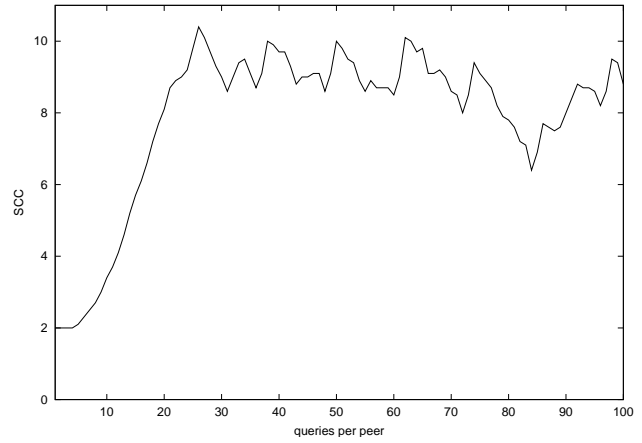
to peers in this group are placed around the 8 o'clock position. On the left we see the initial random connections, on the right we see the connections in the final network. One can observe that there are more local (within group) links and fewer long (cross-group) links on the right-hand-side, revealing the emergence of local clusters in the network topology as the semantic locality is discovered among peers.

Another important network statistics is the number of connected components, which in the case of a directed graph are called strongly connected components (SCC). This number is plotted in Figure 7, showing that the network splits into a number of SCCs usually slightly larger than the number of simulated peer groups; a few groups have more than one SCC. The network maintains a single weakly connected component based on undirected links — peers continue to have access to most peers in other groups.

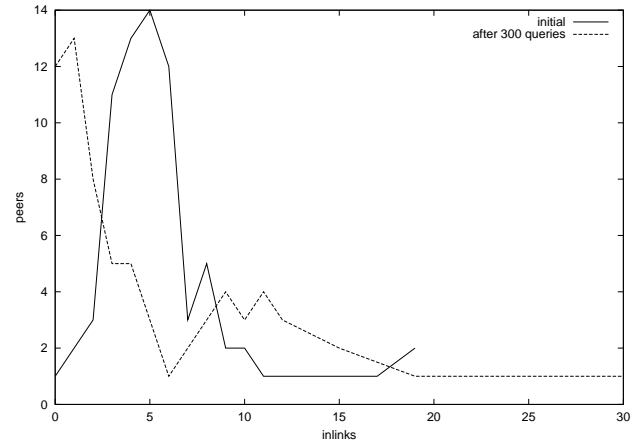
Finally, Figure 8 compares the indegree distribution for the initial random network and a later one. (Recall that outdegree is fixed at  $N_n$ .) The initial network shows a Poisson distribution centered at  $N_n$ , as expected for a random graph. After 300 queries have been submitted to each peer, the distribution has shifted its mode near zero and developed a long tail. This means that a few peers are attracting a greater fraction of the links. For example a single peer in group 7 has 30 incoming links — almost half of all the peers send their queries to this one popular node. We are currently investigating the mechanisms by which some peers achieve such authority.

## 5.2 Quality of results

In order to compare the performance of the 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.



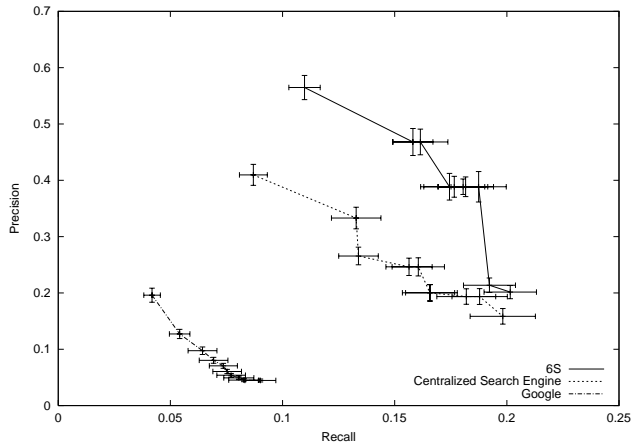
**Figure 7: 6S peer network fragmentation as illustrated by the running average of the number of strongly connected components.**



**Figure 8: 6S peer indegree distribution.**

We consider two different types of centralized search engines. The first one is *Google*, based on the Google Web API. The other is built 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. For each of these two different centralized search engines, 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. In order 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 then submitted to a centralized search engine that combines the 70 peers' search engine databases, and 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

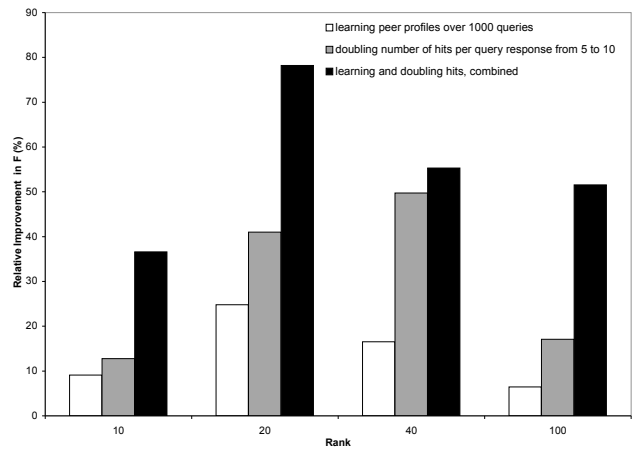


**Figure 9: Precision-recall plots for 6S and three centralized search engines. Recall is plotted on a log scale for visibility. Precision is interpolated (cf. [6], p. 55) and error bars correspond to standard errors of precision and recall averaged across queries.**

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 9 shows the precision-recall plots comparing quality of results by 6S and the two centralized search engines. 6S significantly outperforms both centralized search engines. In particular, 6S achieves higher precision than its centralized counterpart 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. The comparison with Google appears to be even more favorable, but it must be interpreted carefully. One possibility would be to attribute the advantage to the better coverage that 6S peers could achieve by focusing their crawls on pages of interest to their users. This is possible in general, however the seeds used in this particular experiment (ODP pages) are well known to Google and thus it would be surprising if Google did not have good coverage of their neighborhoods. A limited amount of manual inspection suggests that most of the relevant pages are in fact indexed by Google. A more plausible interpretation of the result is in the ranking of relevant pages; 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. Another factor to be considered is that Google’s coverage is larger than the combined 6S coverage by well over three orders of magnitude. On one hand this makes 6S’s favorable performance even more impressive. On the other hand, it is possible that Google contains other relevant pages which are ranked higher than those in our relevant sets; these would unfairly penalize Google’s measured performance. Despite this caveat, we believe the comparative result is worth reporting.

Figure 10 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 learning suggests that most of the advantage enjoyed by this version of 6S (cf. Figure 9) is due to focused coverage rather than to the rudimentary learning algorithm implemented here. The effect of  $N_h$  is larger; more communication can only



**Figure 10: Relative improvement in F-measure due to learning of peer profiles and to increasing numbers of hits per query response,  $N_h$ , plotted versus the total number of top hits considered.**

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 scalability of 6S with network traffic at a finer granularity.

## 6. CONCLUSION

In this paper we have 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 an adaptive routing algorithm 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 preliminary 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. Our experiments also suggest that 6Search can significantly outperform centralized search engines, which cannot take advantage of user context in their crawling and searching processes.

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).

As a project in its infancy stage, 6S has many directions for further development. In the immediate future we plan to extend the present experiments to more peers, more queries, more hits, and longer simulations to better characterize speed of convergence and scalability. Such an evaluation approach has been used success-

fully to analyze the scalability of an adaptive query routing algorithm similar to the one proposed here, in a semantic Web setting where peers query for RDF data [35]. One technique proposed in [35] 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 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 [11, 3]). 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 [21]. 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. Alternative learning algorithms will be analyzed for adaptive query routing. The present results (cf. Figure 10) suggest that there is ample room for improving on the rudimentary scheme outlined in this paper. 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 [1]. 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 [9, 39, 16] will be explored. Referral should also be investigated as an alternative mechanism for adapting the network topology based on local reinforcement interactions [30]. Finally, we need to study how to bootstrap a peer into the network. Simple mechanisms such as those employed by Gnutella and other file sharing peer networks rely on posting the addresses of a few peers and may be adequate since adaptive query routing should rapidly adjust the connections of the new peer.

In parallel with the above algorithmic extensions, implementation of a working 6S server application is under way. We are developing a prototype based on the JXTA framework [36], which will be released to the open-source community. The peer communication protocols will be tested in real environments over TCP/IP. This will also 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.

## 7. ACKNOWLEDGMENTS

We thank Shannon Bradshaw for helpful discussions and Gautam Pant for the topical crawler libraries. We are also grateful to the Nutch Organization for its open source search engine code and to

the Open Directory Project for the data used to model our simulated users. This work was supported in part by NSF Career Grant IIS-0348940. The AVIDD cluster used in the experiments was funded in part by NSF Grant CDA-9601632. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## 8. REFERENCES

- [1] A. Asvanund, S. Bagala, M. Kapadia, R. Krishnan, M. Smith, and R. Telang. Intelligent club management in P2P networks. In *Proc. Workshop on P2P systems*, 2003.
- [2] M. Bawa, R. Bayardo Jr, S. Rajagoplan, and E. Shekita. Make it fresh, make it quick — searching a network of personal webservers. In *Proc. 12th International World Wide Web Conference*, 2003.
- [3] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.
- [4] B. E. Brewington and G. Cybenko. How dynamic is the Web? In *Proc. 9th International World-Wide Web Conference*, 2000.
- [5] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks*, 30(1–7):107–117, 1998.
- [6] S. Chakrabarti. *Mining the Web: Discovering knowledge from hypertext data*. Morgan Kaufmann, San Francisco, 2003.
- [7] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In D. Lassner, D. De Roure, and A. Iyengar, editors, *Proc. 11th International World Wide Web Conference*, pages 148–159, New York, NY, 2002. ACM Press.
- [8] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.
- [9] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. 22nd Intl. Conf. on Distributed Computing Systems (ICDCS'02)*, pages 23–32. IEEE Computer Society, 2002.
- [10] Cyveillance. Sizing the internet. White paper, July 2000. <http://www.cyveillance.com/>.
- [11] S. Deerwester, S. Dumais, F. GW, T. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.
- [12] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of Web pages. In *Proc. 12th International World Wide Web Conference*, 2003.
- [13] G. Jeh and J. Widom. Scaling personalized Web search. In *Proc. 12th International World Wide Web Conference*, 2003.
- [14] S. Johnson. Digging for googleholes. <http://slate.msn.com/id/2085668/>, July 16 2003.
- [15] S. Joseph. Neurogrid: Semantically routing queries in Peer-to-Peer networks. In *Proc. Intl. Workshop on Peer-to-Peer Computing*, 2002.
- [16] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proc. 11th Intl. Conf. on Information and Knowledge Management (CIKM'02)*, 2002.

- [17] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proc. 12th International World Wide Web Conference*, 2003.
- [18] S. Lawrence and C. Giles. Accessibility of information on the Web. *Nature*, 400:107–109, 1999.
- [19] J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *In 2nd International Workshop on Peer-to-Peer Systems*, 2003.
- [20] J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proc. 12th Intl. Conf. on Information and Knowledge Management (CIKM'03)*, 2003.
- [21] J. Lu and J. Callan. Merging retrieval results in hierarchical peer-to-peer networks. In *Proc. 27th Annual Intl. ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004.
- [22] F. Menczer and R. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning*, 39(2–3):203–242, 2000.
- [23] F. Menczer, G. Pant, M. Ruiz, and P. Srinivasan. Evaluating topic-driven Web crawlers. In D. H. Kraft, W. B. Croft, D. J. Harper, and J. Zobel, editors, *Proc. 24th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 241–249, New York, NY, 2001. ACM Press.
- [24] F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *ACM Transactions on Internet Technology*, 4(4), 2004. Forthcoming.
- [25] J. Mostafa. Guest editor's introduction: Information customization. *IEEE Intelligent Systems*, 17(6):8–11, 2002.
- [26] G. Pant, S. Bradshaw, and F. Menczer. Search engine - crawler symbiosis. In *Proc. 7th European Conference on Digital Libraries (ECDL)*, 2003.
- [27] G. Pant, P. Srinivasan, and F. Menczer. Exploration versus exploitation in topic driven crawlers. In *Proc. WWW-02 Workshop on Web Dynamics*, 2002.
- [28] G. Pant, P. Srinivasan, and F. Menczer. Crawling the Web. In M. Levene and A. Poulouvassilis, editors, *Web Dynamics*. Springer, 2003.
- [29] J. Pujol, R. Sangüesa, and J. Bermúdez. Porqpine: A distributed and collaborative search engine. In *Proc. 12th Intl. World Wide Web Conference*, 2003.
- [30] M. Singh, B. Yu, and M. Venkatraman. Community-based service location. *Communications of the ACM*, 44(4):49–54, 2000.
- [31] P. Srinivasan, G. Pant, and F. Menczer. A general evaluation framework for topical crawlers. *Information Retrieval*, 2004. Forthcoming.
- [32] C. Suel, T. amd Mathur, J.-W. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. ODISSEA: A Peer-to-Peer architecture for scalable Web search and information retrieval. In *International Workshop on the Web and Databases (WebDB)*, 2003.
- [33] W. Sullivan III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major SETI project based on Project Serendip data and 100,000 personal computers. In *Proc. 5th Intl. Conf. on Bioastronomy*, 1997.
- [34] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. ACM SIGCOMM '03*, 2003.
- [35] C. Tempich, S. Staab, and A. Wranik. REMINDIN': Semantic query routing in peer-to-peer networks based on social metaphors. In *Proc. 13th conference on World Wide Web*, pages 640–649. ACM Press, 2004.
- [36] S. Waterhouse. JXTA Search: Distributed search for distributed networks. Technical report, Sun Microsystems Inc., 2001.
- [37] D. Watts. *Six degrees: The Science of a Connected Age*. Norton, 2003.
- [38] D. Watts and S. Strogatz. Collective dynamics of “small-world” networks. *Nature*, 393:440–442, 1998.
- [39] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proc. 22nd Intl. Conf. on Distributed Computing Systems (ICDCS'02)*, pages 5–14. IEEE Computer Society, 2002.