Modeling and measuring scalable peer-to-peer search networks

Brian F. Cooper and Hector Garcia-Molina

Department of Computer Science
Stanford University
Stanford, CA 94305
{cooperb,hector}@db.stanford.edu

Abstract

The popularity of peer-to-peer search networks grows, even as the limitations to the scalability of existing systems become apparent. We propose a simple model for search networks, called the *search/index links* (SIL) model. The SIL model describes existing networks while also yielding organizations not previously studied. Using simulation results, we argue that a new organization, *parallel search clusters*, is superior to existing supernode networks in many cases.

Keywords: peer-to-peer search, content discovery, supernodes, network modeling, efficient networks

1 Introduction

Peer-to-peer search networks have become very popular as a way to effectively search a large number of distributed, diverse collections. These search networks take advantage of the large aggregate processing power of many hosts, while leveraging the distributed nature of the system to enhance robustness. "Peer-to-peer" (P2P) is a general term and describes a family of network organizations that combine search forwarding and content indexing to provide an information location service. The various different P2P network organizations have advantages and disadvantages in terms of scalability, efficiency and fault tolerance.

We present a simple model that allows us to represent and visualize many current P2P network organizations. We call this model the *search/index link* (SIL) model. The SIL model allows us to examine and compare the properties inherent within a particular organization, independent of the details of specific implementations of that organization. Our model also yields new organizations that have not previously been

studied. We show that these new organizations can be superior to existing P2P networks in some interesting cases.

2 The search/index link model

A *peer-to-peer search network* is a set of peers that store, search for, and transfer digital documents. We consider here "fuzzy" or content searches, such as keyword searches, metadata searches, and so on. This distinguishes a peer-to-peer search network from a distributed hash table [18, 15], where the search is for a specific document with a specific identifier. Each peer in the network maintains an index over its content (such as an inverted list of the words in each document) to assist in processing searches.

The search network forms an *overlay network* on top of a fully-connected underlying network infrastructure. Peers that are neighbors in the overlay are connected by network links that are logically persistent, although they may be implemented in a connection-oriented or connectionless way. In the search/index link (SIL) model, there are four kinds of network links, distinguished by the types of messages that are sent, and whether a peer receiving a message forwards the message after processing it:

- A *non-forwarding search link* (NSL) carries search messages from one peer A to another peer B. Peer B processes each search message and returns results to A, but does not forward the message to other nodes.

- A *forwarding search link* (FSL) carries search messages from A to B. Peer B will process each search message, return search results to A, and forward the message along any other forwarding search links originating at B. If A is not the originator of the query, it should forward any search results received from B (and any other nodes) along the FSL on which A received the query. Each search message should have a unique identifier that is retained as the message is forwarded. When a peer receives a search message with an id it has previously seen, the peer should discard the message without processing or forwarding it. This will prevent messages from circulating forever in the network if there is a cycle of FSLs.

- A *non-forwarding index link* (NIL) carries index update messages from A to B. Peer B adds A's index entries to its own index, and then effectively has a copy of A's index. Peer B does not forward the update message.

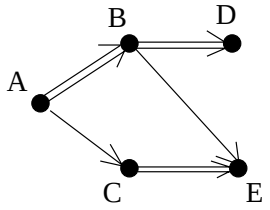


Figure 1: A network with search links.

- A *forwarding index link* (FIL) carries index update messages from A to B, as with non-forwarding index links, but then B forwards the update message along any other forwarding index links originating at B. As with FSLs, update messages should have unique ids, and a peer should discard duplicate update messages without processing or forwarding them.

Network links are directed communications channels. A link from peer A to peer B indicates that A sends messages to B, but B only sends messages to A if there is also a separate link from B to A. Modeling links as directed channels makes the model more general. An undirected channel can of course be modeled as a pair of directed links going in opposite directions. For example, the links in Gnutella [1] can be modeled as a pair of forwarding search links, one in each direction. Although forwarding links may at first glance seem more useful, we will see later how non-forwarding links can be used (Section 3).

Figure 1 shows an example network containing search links. Non-forwarding search links are represented as single arrows ($\longrightarrow$) while forwarding search links are represented as double arrows ($\Longrightarrow$). Imagine that a user submits a query to peer A. Peer A will first process the query and return any search results it finds to the user. Node A will then forward this query to both B and C, who will also process the query. Node B will forward the query to D, but not E, since the edge $B \rightarrow E$ is an NSL. Node C will not forward the query, since it received the query along an NSL. The user's query not reach E at all, and E's content will not be searched for this query.

A peer uses an *index link* to send copies of index entries to its neighbors. These neighbors incorporate these entries into their own index, and process queries over them. For example, consider a peer A that has an index link to a peer B. When B processes a query, it will return search results both for its own content as well as for the content stored at A. Peer A need not process the query at all. We say that B is *searched directly* in this case, while A is *searched indirectly*.

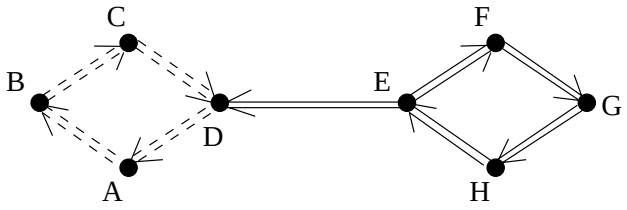


Figure 2: A network with search and index links.

Whenever a peer creates a new index entry or modifies an existing entry, it should send a message indicating the change along all of its outgoing index links. A peer might create an index over all of its locally stored documents when it first starts up, and should send all of the index entries to each of its index link neighbors. Similarly, if a node deletes a document, it would remove the corresponding entries from its own index as well as notifying its index link neighbors to do the same.

Figure 2 shows a network that contains both search and index links. Index links are represented as dashed lines, single (⇢) for non-forwarding index links and double (⇛) for forwarding index links. Nodes A, B, C and D are connected by a "ring" of FILs. An index update occurring at peer A will thus be forwarded to B, C, D and back to A (A will not forward the update again). In fact, all four of the nodes $A...D$ will have complete copies of the indexes at the other three nodes in the index "ring". Nodes E, F, G and H are connected by FSLs, and a search originating at any peer $E...H$ will reach, and be processed by, the three other nodes on the search "ring." Notice that there is also an FSL between E and D. Any query that is processed by E will be forwarded to D, who will also process the query. Since D has a copy of the indexes from $A...C$, this means that any query generated at E, F, G and H will effectively search the content of all eight nodes in the network. In contrast, a query generated at nodes $A...D$ will be processed at the node generating the query, and will only search the indexes of the nodes $A...D$.

For the rest of our discussion, it is useful to define the concept of a *search path*:

- A *search path* from peer X to peer Y is
  - a (possibly empty) sequence of FSLs $f_1, f_2, ...f_n$ such that f_1 originates at X, f_n terminates at Y, and f_i terminates at the same node at which f_{i+1} originates, or
  - an NSL from X to Y

A search path from X to Y indicates that queries processed at X will eventually be forwarded to Y. For

example, in Figure 2 there is a search path from F to D but not from D to F. Note also that there is (trivially) a search path from a node to itself.

Similarly, an *index path* from X to Y is a sequence of FILs from X to Y, or one NIL from X to Y. In this case, X's index updates will be forwarded to Y, and Y will have a copy of X's index.

2.1 "Good" networks

The network links we have discussed above are not by themselves new. Forwarding search links are present in Gnutella, forwarding index links are used in publish/subscribe systems, non-forwarding index links are used in supernode networks, and so on. However, different link types tend be used in isolation or for narrowly specific purposes, and are rarely combined into a single, general model. Our graphical representation allows us to consider new combinations. In fact, the number of search networks of n nodes that can be constructed under the SIL model is exponential in n^2. Only a small fraction of these networks will allow users to search the content of all the peers in the network, and an even smaller fraction will also have desirable scalability, efficiency or fault tolerance properties. We must examine how we construct "good" networks using our SIL model, and this of course requires defining what we mean by "good."

First, we observe that a search network only meets users' needs if it allows them to find content. Since content may be located anywhere in the network, a user must be able to effectively search all of the content repositories, either directly or indirectly. We can quantify this goal by defining the concept of *coverage*.

> Definition. The *coverage* of peer p in a network N is the fraction of the peers in N that can be searched, either directly or indirectly, by a query generated by p.

Ideally, networks should have *full coverage*:

> Definition. A network N has *full coverage* if every peer p in N has coverage=1.

Even a network that has full coverage may not necessarily be "good." Good networks should also be efficient, in the sense that peers are not overloaded with work answering queries. One important way to improve the efficiency of a network is to reduce or eliminate redundant work. If peers are duplicating each other's processing, then they are doing unnecessary work.

> Definition. A search network N has *redundancy* if there exists a network link in N that can be removed without reducing the coverage for any peer.

Intuitively, redundancy results in messages being sent to and processed by peers, even when such processing does not add to the network's ability to answer queries.

Redundancy can manifest in search networks in four ways. First, *search/search redundancy* occurs when the same peer P processes the same query from the same user multiple times. If there are multiple search paths from the user's peer to P, then P can receive several copies of the query. The peer P must process the query the first time in order to give the user full coverage. After that, any subsequent processing of the query is redundant and unnecessary. As mentioned earlier, to deal with this issue queries are tagged with an id number, and each peer checks the id of each query it receives. If this check can be made significantly cheaper than processing the query again, then the effects of search/search redundancy become negligible.

A second type is *update/update redundancy*, which occurs when the same peer P processes the same update multiple times. Consider a peer R that has multiple index paths to P. Peer R's update messages will be copied by the network as they are forwarded, and multiple copies will reach P. Peer P must process the update the first time, but subsequent processing of the same update is redundant. The effects of update/update redundancy is mitigated in a manner similar to search/search redundancy: by tagging update messages with ids and checking for duplicate ids.

A third type is *search/index redundancy*, where a peer A processes a query even though another peer B has a copy of A's index and processes the same query. Because B will return search results covering A's content, it is unnecessary for A to process the query directly.

Finally, *index/index redundancy* is where two different peers B and C both process a search over a copy of a third peer A's index. If A has an index path to B and C, both peers will have to process A's index updates. Yet, to achieve full coverage, only one of the copies of the index is necessary, and the processing by the second peer is redundant.

Search/index and index/index redundancy involve unnecessary work being done at two different peers, and it is difficult for those peers to coordinate and discover that their work is redundant. Therefore, in order to reduce load it is important to design networks that do not have search/index and index/index redundancies.

Note that redundancy may actually be useful to improve the fault tolerance of the system, since if one node fails another can perform its processing. Moreover, redundancy may be useful to reduce the time a user must wait for search results, if a node near the user can process the user's search even if this processing is redundant. However, fault tolerance and search latency tradeoff with efficiency, since redundancy results in

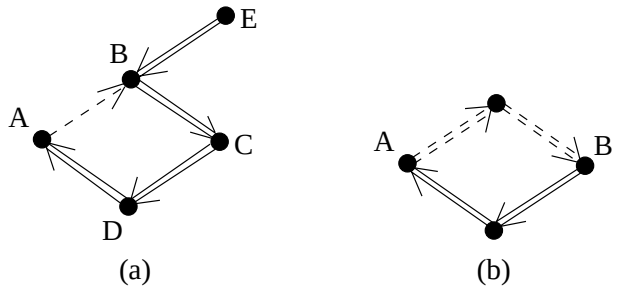


Figure 3: Networks with cycles: a. with search/index redundancy, and b. no search/index redundancy.

extra work for peers. Our goal here is to first design a non-redundant search network, which will be the most efficient. After that, redundancy can be judiciously addded to improve fault tolerance or latency, as needed.

2.2 Topological features of networks with redundancy

The concept of "redundancy" and even the subconcepts like search/index redundancy are quite general. When designing a peer-to-peer search network, it is useful to identify specific features of network topologies that lead to redundancy, and avoid those features. As noted above, we will focus on search/index and index/index redundancies.

One feature that causes search/index redundancy is a specific type of cycle called a *one-index-cycle*: a node A has an index link to another node B, and B has a search path to A. An example is shown in Figure 3a. This construct leads to redundant processing, since B will answer queries over A's index, and yet these queries will be forwarded to A who will also answer them over A's index. More formally, a one-index-cycle fits our definition of *redundancy* because at least one link in the cycle can be removed without affecting coverage: the index link from A to B. Note that not all cycles necessarily cause redundancy. Consider the cycle in Figure 3b. In this case, there is not a single index link but instead a chain of FILs forming an index path from A to B. This cycle may seem to introduce redundancy in the same way as Figure 3a, except that none of the index links can be removed without reducing coverage for some node.

Another feature that causes search/index redundancy is a *search fork*: a node C has a search link to A and a search path to B that does not include A, and there is an index path from A to B. An example is shown in Figure 4a. Again, A will process any searches from C unnecessarily, since B can process the queries for A. The redundant link in this example is the link $C \Longrightarrow A$. We specify that there is a search path from C to B

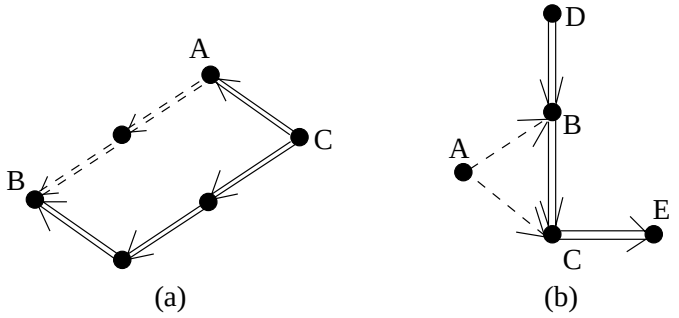


Figure 4: Networks with forks: a. a search fork, and b. an index fork.

that does not include A because if the only path from C to B included A there would be no link that could be removed without reducing coverage.

Similarly, a feature that causes index/index redundancy is an *index fork*: a node A has an index link to B and an index link to C, and there is a search path from B to C. An example is shown in Figure 4b. The index link from A to B is redundant, and will require B to process index update unnecessarily without increasing coverage.

These topological features are samples of those that cause redundancy, and hence should be avoided.

3 Fundamental network organizations

By applying the principle of "no redundancy" and avoiding features like those of Section 2.2, we can construct networks that are likely to be more efficient than networks constructed randomly. Moreover, by restricting our attention to non-redundant networks with full coverage, we can limit our examination to networks that fully process user searches without placing undue load on peers. Although the topology of individual networks differ, in this section we identify some fundamental organizations that meet these ideal conditions of low load and full coverage.

First, note that there are two basic network topologies that can meet the conditions of full coverage and no search/index or index/index redundancy:

- *Pure search networks*: strongly connected networks with only search links.
- *Pure index networks*: strongly connected networks with only index links.

In graph theory, a *strongly connected directed graph* is one in which there is a directed path from every node to every other node. Recall from Section 2 that in our SIL model, a path is either a sequence of forwarding links or a single non-forwarding link. When we say "strongly connected" in the definitions above (and below), we mean "strongly connected" using this definition of search and index paths.

In these basic topologies, there cannot be search/index or index/index redundancies since index links and search links do not exist in the same network. However, these networks are not "efficient" in the sense that nodes are lightly loaded. In a pure search network, every node processes every search, while in a pure index network, every node processes every index update. These topologies may be useful in extreme cases; for example, a pure search network is not too cumbersome if there are very few searches. A well known example of a pure search network is the Gnutella network.

However, a very useful technique is to combine search links and index links to reduce the load on nodes. We have identified four fundamental topologies that are described by the SIL model, have full coverage and no search/index or index/index redundancy:

- *Supernode networks*
- *Global index networks*
- *Parallel search cluster networks*
- *Parallel index cluster networks*

Some of these topologies are not new, and exist in networked systems today. Supernode networks are typified by the FastTrack network of Kazaa [2], while the global index network is similar to the organization of Netnews with a central indexing cluster (like DejaNews). However, the parallel search and index clusters have not been previously examined.

A *supernode network* is a network where some nodes are designated as "supernodes," and the other nodes ("normal nodes") send both their indexes and searches to supernodes. The supernodes are linked by a strongly connected pure search network. A supernode network can be represented in our SIL model by having normal nodes point to supernodes with one FSL and one NIL, while supernodes point to each other using FSLs. An example is shown in Figure 5a. Each supernode therefore has the copies of several normal nodes' indexes. Supernodes process searches before forwarding them to other supernodes. Normal nodes only have to process searches that they themselves generate. Thus, supernodes networks result in much

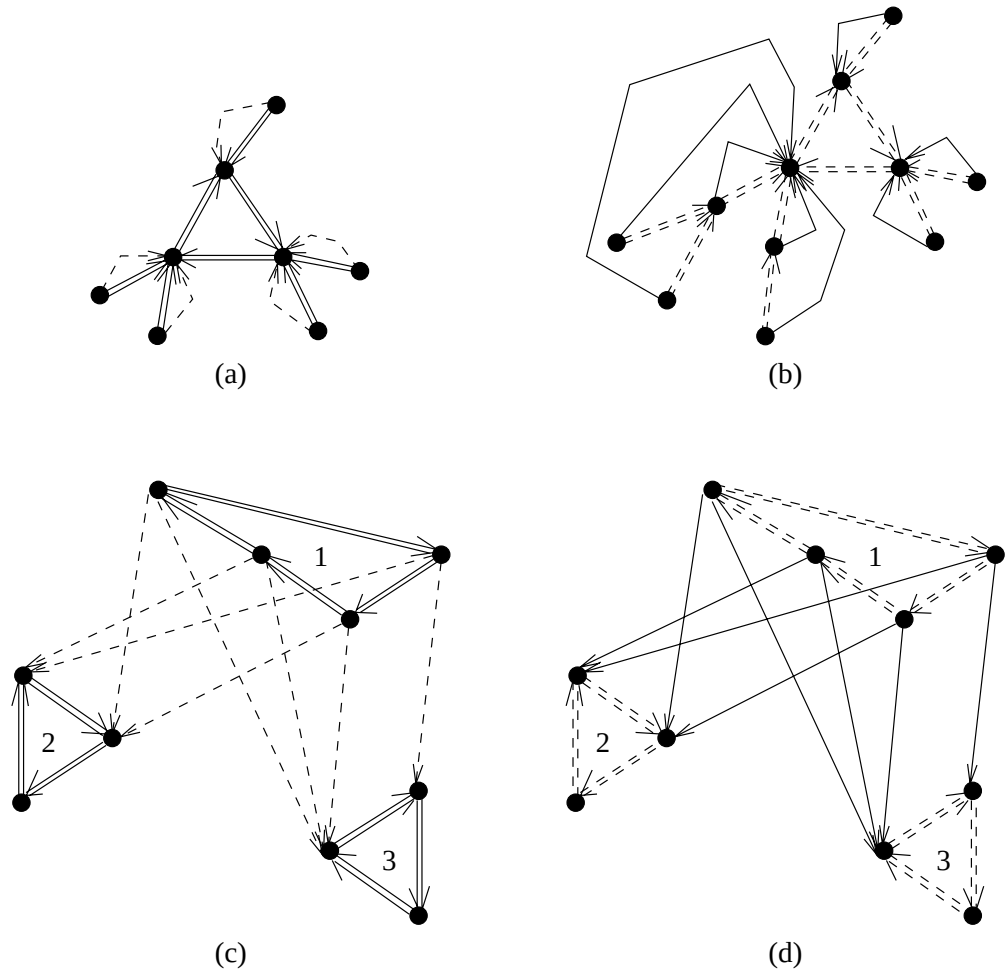


Figure 5: Fundamental topologies: a. Supernodes, b. Global index, c. Parallel search cluster, and d. Parallel index cluster. Some inter-cluster links are omitted in networks c and d for clarity.

less load on an average peer than a pure search network. A disadvantage is that as the network grows, the search load on supernodes grows as well, and ultimately scalability is limited by the processing capacity of supernodes. This disadvantage exists even though there is unused processing capacity in the network at the normal nodes. These normal nodes cannot contribute this spare capacity to reduce the search load on supernodes, because even if a normal node is promoted to a supernode, every supernode must still process all the queries in the network.

An organization similar to supernodes is a *global index network*, as illustrated in Figure 5b. In this organization, some nodes are designated as global indexing nodes, and all index updates in the system flow to these nodes. A normal node sends its queries to one of these global indexing nodes. The global indexing

nodes themselves are connected by a strongly connected pure index network. Under our model, normal nodes have a FIL to another normal node or to a global index node, and normal nodes also have NSLs to global index nodes. In this example, the normal nodes form a tree of index paths rooted at a global index node. Index updates flow from the normal nodes to form a complete set of global indexes at each of the global index nodes. Note that a similar tree-like structure could be constructed in the supernode network, where normal nodes would form a tree of search paths rooted at a supernode, while each normal node would have an index link directly to a supernode.

The advantages of global index networks are similar to those of supernode networks. Most nodes process only index updates and their own searches, while leaving the processing of all other searches to the global index nodes. Moreover, there are multiple nodes that have a complete set of indexes, so the network can recover from the failure of one node. However, the load on the global index nodes is high; each global index peer must process all the index updates in the system and a significant fraction of the searches.

A third organization is called *parallel search clusters*. In this network, nodes are organized into clusters of strongly connected pure search networks (consisting of FSLs), and clusters are connected to other clusters by NIL index links. An example is shown in Figure 5c. In this figure, the cluster "1" has outoing NILs to the other clusters "2" and "3". Clusters "2" and "3" would also have outoing NILs to the other clusters, but we have omitted them in this figure for clarity. The nodes in each cluster collectively have a copy of the indexes of every node outside the cluster, so full coverage is achieved even though queries are only forwarded within a cluster. Unlike in a supernode topology, there are no nodes that must handle all of the queries in the network. Nodes only handle queries that are generated within their own cluster. Moreover, all of the search processing resources in the system are utilized, since every node processes some queries. A disadvantage of this topology is that nodes must ship their index updates to every other cluster in the network. If the update rate is high, this will generate a large amount of update load. The network topology may also be difficult to maintain, since every node must know about all the other clusters and maintain possibly many outoing NILs.

Finally, the fourth organization is *parallel index clusters*. In this organization, clusters of strongly connected pure FIL index networks are connected by NSL search links. As a result, nodes in one cluster send their searches to one node of each of the other clusters. An example is shown in Figure 5d. (Again, some inter-cluster links are omitted in this figure.) This topology is the analog of parallel search clusters, with index links replaced by search links and vice versa. Parallel index clusters have advantages and

disadvantages similar to parallel search cluster networks: no node handles all index updates or all searches, and all resources in the system are utilized, while inter-cluster links may be cumbersome to maintain and may generate a large amount of load.

These fundamental topologies can be varied or combined in various ways. For example, a variation of the supernode topology is to allow a normal node to have an FSL pointing to one supernode and an NIL pointing to another. Another example is to vary parallel cluster search networks by allowing the search clusters to be constructed as mini-supernode networks instead of (or in addition to) clusters that are pure search networks. These and other variations are useful in certain cases. Allowing a mini-supernode network as a search cluster in a parallel search cluster network is useful if the nodes in that cluster are heterogeneous, and some nodes have much higher capacities than the others. Moreover, note that the pure index and pure search networks are special cases of our four fundamental topologies. For example, a pure search network can be viewed as a supernode network where all nodes are supernodes, or a parallel search cluster network where there is only one cluster.

Finally, as we stated in Section 2.1 our restriction of no redundancy can be relaxed to improve the fault tolerance or search latency of the system at the cost of higher load. For example, in a supernode network, a normal node could have an NIL/FSL pair to each of two different supernodes. This introduces, at the very least, an index/index redundancy, but ensures that the normal node is still fully connected to the network if one of its supernodes fails.

4 Experimental results

The SIL model allows us to understand the spectrum of options when constructing peer-to-peer search networks. The model also allows us to arrive at some organizations, like parallel clusters, that have not been studied in previous work. These “new” options have important advantages, which we discussed qualitatively in Section 3. To illustrate and evaluate these advantages, we have conducted simulation experiments to build different networks and compare them quantitatively on various metrics. We have focused on supernode networks, as implemented in popular peer-to-peer networks such as Kazaa, and parallel search cluster networks, which use the same basic building blocks (FSLs and NILs) as supernode networks. Our results, described in this section, show that the parallel search cluster organization can be superior to traditional supernode networks.

4.1 Comparing network organizations

The organizations described in Section 3 are really families of networks. Within a family there can still be widely varying topologies. For example, different supernode networks can differ in the number of supernodes, the distribution of normal nodes to supernodes, the topology of the pure search network between supernodes, and so on. Given this diversity, which supernode variant do we compare to which parallel clusters variant? Our approach to this dilemma is as follows. First, we select a scenario with a given number of nodes and a given query and update load. Then, we search for parameters that lead to "good" supernode networks, that is, networks that minimize load for this scenario while retaining full coverage. We similarly find "good" parallel cluster networks. Then, we compare good instances of supernode networks to good instances of parallel cluster networks. We iterate over different scenarios, comparing different supernode and parallel cluster instances in each case. Due to space limitations, we do not present the process of finding a good network here; see [6].

It is also important to define quantitative metrics for comparing topologies. As mentioned earlier, we restrict our attention to networks that have *full coverage*. A network can reduce coverage in order to optimize another metric, such as load. For example, load is reduced as a result of the "hop count horizon" in Gnutella, where queries are only forwarded a finite number of hops and never reach peers beyond that horizon. However, we are concerned with studying idealized network topologies in order to gain an understanding of the properties inherent in those topologies. Certainly, reducing coverage to reduce load may be a vital optimization in an implemented network, but we have chosen to simplify our analysis of idealized networks by requiring full coverage.

We can define metrics that measure the amount of load on individual peers:

> Definition. *Search load* is the load on peers from processing searches. This is measured as the number of search messages that reach peers.
>
> Definition. *Update load* is the load on peers from processing index updates sent via index links from other peers. This is measured as the number of index messages that reach peers.
>
> Definition. *Total load* is the sum of the search load and update load.

Notice that we are using a very general definition for "load" which encapsulates both the load on the network links and the load on the peers themselves. This is because we wish to define a general model that is

independent of the physical characteristics of networks or machine architectures. Moreover, we are mainly concerned with the relative measure of these metrics (e.g., network X has less load than network Y) and not their absolute values. Therefore, we define load in terms of "numbers of messages," and make the simplifying assumption that all messages are equally costly to process. This assumption may not be true, for example, if it is twice as costly to process a search as an index update. However, this situation is equivalent in our model to one where there are twice as many search messages as index messages. In fact, we study situations where there are more search messages than update messages generated in the system, more update messages than search messages, and an equivalent number of search and update messages. This allows us to model both the situation where one kind of message is produced more frequently and the case where one kind of message is more expensive to process.

A robust network must also be able to survive node failures. We can measure the resistance to failures by defining the *fault susceptibility* of a network:

> Definition. *Fault susceptibility* is the maximum decrease in network coverage caused by the failure of any one node.

We assume failures are fail-stop, so a node that fails ceases to process and forward messages. When this happens, the network coverage, measured as the fraction of the nodes in the network that are searchable, may decrease. For example, in a network with full coverage (e.g., coverage=1) a failure may cause the network to partition into three subnetworks, each unreachable from the other. If the subnetworks are of equal size, the fault susceptibility is 0.66, since each node can only search one third of the network and the coverage drops to 0.33. We have called this metric "fault susceptibility" instead of the more traditional "fault tolerance" because we are concerned with the effect of a single node failure rather than with the number of failures the network can tolerate.

Finally, we note that user satisfaction is increased if queries return results quickly. To measure this effect, we can define the *search latency* metric:

> Definition. The *search latency for a peer* p is the longest search path that a query generated by p must travel.

Intuitively, this metric represents the time that a user must wait before all of the search results are guaranteed to return. When a user submits a query to node p, it will be forwarded to every peer r_i such that there is a

Parameter	*Description*	*Base value*
n	Number of nodes	100
P_A	Average links per node for PLOD	5
P_M	Maximum links per node for PLOD	10
NS	Number of supernodes	$1...n$
NC	Number of clusters	$1...n$
SL	Average search load generated by a peer	$10...100$
UL	Average update load generated by a peer	$10...100$

Table 1: Simulation parameters

search path from p to r_i. The user may quickly get some results, but will only be assured of getting all results when that query reaches every node r_i. Since messages can travel separate paths in parallel, the maximum time for the query to reach every node r_i (and for search results to return) is proportional to the number of hops from p to the r_i that is farthest away.

Definition. The *search latency for a network* N is the the average over all peers $p_1, p_2...p_n$ of the search latency of each peer p_i.

4.2 Simulation setup

We have constructed a simulator to generate networks with a given topology, and to evaluate the metrics of load, fault susceptibility and search latency. Our simulater takes several input parameters, which are summarized in Table 1. The topologies are static in the sense that they are constructed a priori, and we calculate metrics only once they have been built. This again allows us to study the properties inherent in a specific topology. In ongoing work we are examining the properties of dynamic networks, where nodes are constantly joining and leaving (as they do in real networks.)

For our experiments, we generated pure search, supernode and parallel cluster networks. For each scenario, we generated 50 "good" instances of each topology, and the results we report represent averaging our metrics over all of these instances. In each case, our results have 95 percent confidence intervals of $\pm$ 3 percent or less of the value reported, unless otherwise noted. Each network instance had n nodes. We assume for simplicity that each node has equivalent bandwidth and processing capacity. We allowed search/search redundancy, since this redundancy reduces fault susceptibility and search latency without increasing load, and because real networks such as Gnutella and Kazaa also have search/search redundancy.

As a baseline, we constructed pure search networks using the Gnutella model, where we connect two nodes using a pair of FSLs going in opposite directions. Since other investigators have noted that Gnutella networks tend towards a power law distribution [10, 16], we have constructed pure search networks using the PLOD algorithm [14]. The PLOD algorithm takes two parameters, the average number of links per node and the maximum number of links per node, and generates a power-law network. The values we chose for these parameters (see Table 1) were determined experimentally to ensure full coverage, reduce fault-susceptibility, and reduce search latency; see [6].

We built supernode networks by designating some nodes as supernodes; the number of supernodes *NS* is specified as an input parameter to the simulation. Each normal node was assigned to a supernode randomly using a generator that produced normally distributed random values (via the polar Box-Muller algorithm [4]). The number of normal nodes assigned to a given supernode is normally distributed with a mean of n/NS and the standard deviation of one quarter of the mean. Normal nodes were connected to their supernode with one outgoing FSL and one outgoing NIL. Supernodes were connected using a Gnutella-like network of FSLs similar to the pure search network.

We made parallel search cluster networks by assigning nodes to clusters of FSLs, and connecting separate clusters with NILs. The number of clusters *NC* is specified as an input parameter to the simulation. The number of nodes in a given cluster is normally distributed with a mean of n/NC and a standard deviation of one quarter of the mean. Note that this means that the number of nodes in a cluster is similar to the number of normal nodes assigned to a supernode. Within a cluster, we built a power law search network of FSLs, similar to the pure search network above. After we had constructed clusters, every node in the network was given one NIL to one randomly chosen member of each other cluster.

For all types of networks, we assigned two parameters to each peer p_i: SL_i, the amount of search load created by the peer, and UL_i, the amount of index update load created by the peer. Then, by examining a topology and determining for each peer p which peers can send search and update traffic to p, we can calculate the total load for p.

Definition. The *load for a peer* p is the sum of:

- The SL_i for each peer r_i such that there is a search path from r_i to p, and
- The UL_i for each peer r_i such that there is an index path from r_i to p.

Definition. The *average load for a network* N is the average over all peers $p_1, p_2 ... p_n$ in N of the load for each peer p_i.

Definition. The *maximum load for a network* N is the load at the most heavily loaded peer p in N.

The average SL_i over all P_is in the network is denoted *SL*, while the average UL_i over all P_is is *UL*. The SL_i and UL_i parameters were assigned randomly to each node according to a normal distribution. The mean of the distribution was specified per experiment. For example, in order to study a situation where $SL >> UL$ we assigned 100 "load units" as the mean *SL* and 10 "load units" the mean *UL*. The SL_i and UL_i parameters were then generated for each node, distributed with the given mean, again using the Box-Muller transformation. The standard deviation for the normal distribution was one quarter of the mean. In each case, the mean *SL* and *UL* values was normalized so that $SL + UL$ was constant across all experiments.

The search latency for a peer P was calculated using two steps. First, we determined the shortest path s_i to each peer P_i where P_i had a search path to P. Then, we calculated search latency as the length of the longest s_i. The search latency for the network was the average search latency of each peer.

Recall that the fault susceptibility of the network is the maximum change in coverage that could be caused by removing one peer from the network. This was measured in a straightforward way, by removing each peer, measuring the change in coverage, and then restoring the peer and its connections to the network before removing the next peer.

4.3 Load experiments

First, we examined the load characteristics of supernodes and parallel clusters versus each other and a baseline of a pure search network. To do this, we varied the ratio between the average *SL* and *UL* from $SL = UL \times 10$ to $SL = UL/10$. For each point in this interval, we constructed "good" supernode and cluster networks as discussed in Section 4.1. For supernodes, we used:

- 5 supernodes for the interval $SL = 10 \times UL$ to $SL = 5 \times UL$, and
- 20 supernodes for the interval $SL = 4 \times UL$ to $SL = UL/10$.

For parallel clusters, we used:

- 15 clusters for the interval $SL = 10 \times UL$ to $SL = 3 \times UL$, and

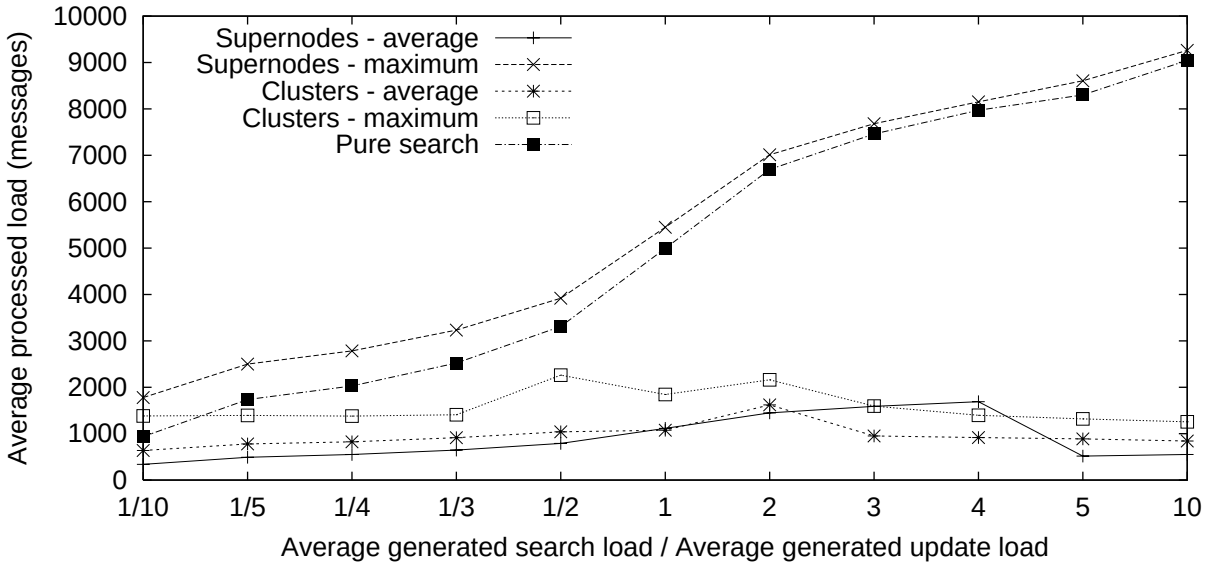


Figure 6: Load of different topologies.

- 10 clusters for the interval $SL = 2 \times UL$ to $SL = UL/2$, and
- 5 clusters for the interval $SL = UL/3$ to $SL = UL/10$.

We also constructed a Gnutella-like pure search network as a baseline comparison.

The results are shown in Figure 6. This figure shows both average and maximum load for the supernode and parallel cluster topologies, as well as the load for the pure search network (where the average and maximum are the same.) On the extreme left of Figure 6, searches are rare (for each search issued 10 index updates are generated), while the extreme right represents a mostly search scenario. In this result, the 95 percent confidence interval of the maximum load in a parallel cluster topology is as large as $\pm$ 10 percent (in the case of $SL = UL/10$). First, note that the maximum load in the supernode network is close to, though higher than, the load in the pure search network. This is because supernodes must handle both searches and updates, while pure search network peers only handle searches. Note also that the average load for supernode networks is significantly lower than that for pure search networks, which is to be expected, since most of the nodes in a supernode network are only handling their own local searches and updates.

A striking result in Figure 6 is that both the maximum and average load of parallel cluster networks are relatively low, roughly comparable to the average load in a supernode network. In fact, the average load in a parallel cluster network is always (in these experiments) within a factor of two of the average load in a

supernode network, and the maximum load in a cluster network is only as much as a factor of four larger than the average load in a supernode network. Moreover, sometimes the average and maximum load in a cluster network is less than the average load in a supernode network. This indicates that nodes in a parallel cluster network are effectively sharing the load, contributing their resources to reduce the overall load on all nodes in the network. In contrast, in a supernode network, some nodes are lightly loaded while the supernodes are heavily loaded, and the most heavily loaded supernode can be up to seven times more loaded than the most heavily loaded node in a parallel cluster network.

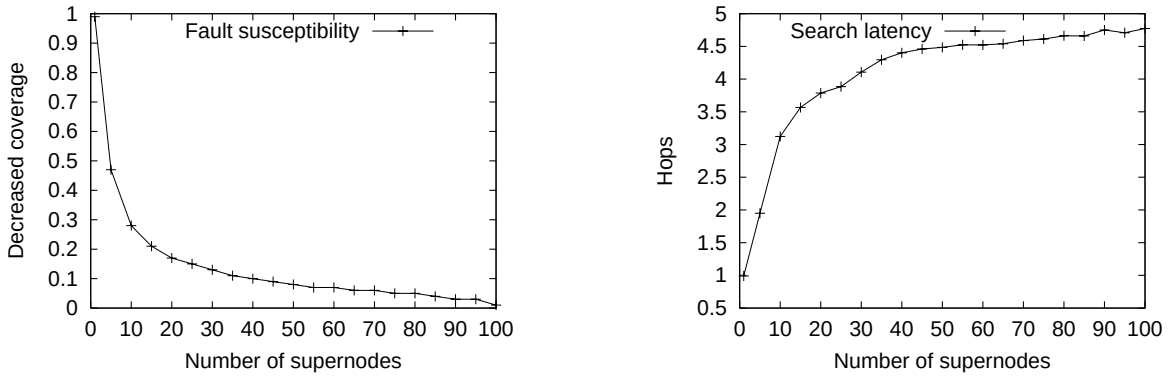
We can draw the following conclusions from these results:

- A parallel cluster network ensures that no nodes are overloaded (e.g., more than twice as loaded as the average node), while only increasing the average load on nodes by up to a factor of two over a supernode network. This is beneficial:
  - Under the assumptions of our simulation (e.g., all nodes have roughly equal capability), and
  - When a primary goal is to reduce both the maximum and average load on nodes in the system.
- A supernode network ensures that most nodes in the system are lightly loaded, at the cost of placing heavy load on supernodes. This is beneficial
  - When some nodes have higher capacities than others, so that these high capacity nodes can serve as supernodes, and
  - When a primary goal is to reduce the average load on nodes in the system.

4.4 Search latency and fault susceptibility experiments

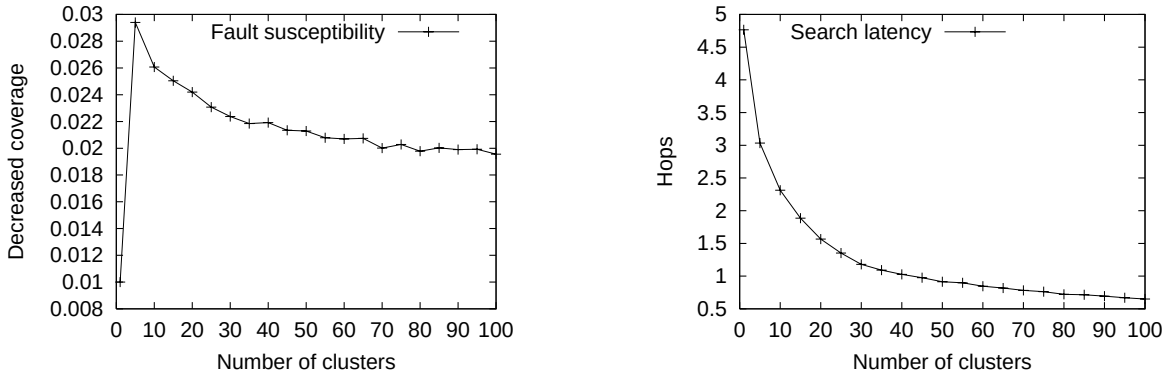
The next set of experiments we ran was to measure the search latency and fault susceptibility of the supernode and parallel cluster topologies. These metrics do not depend on the search or update rate; instead, the connectivity and topology of the network determines how quickly searches will be returned, and how vulnerable the network is to node failures. In this section, we report search latency and fault susceptibility as the number of clusters or supernodes vary.

First, Figure 7 shows the results for supernode networks. Figure 7a shows that the fault susceptibility drops as the number of supernodes increases, while simultaneously Figure 7b shows that the search latency



(a) (b)

Figure 7: Supernodes: (a) fault susceptibility and (b) search latency.



(a) (b)

Figure 8: Parallel search clusters: (a) fault susceptibility and (b) search latency.

increases with more supernodes. When there is only one supernode, the search latency is low, since all searches only have to travel one hop from any normal node to the one supernode. On the other hand, if that supernode fails, the network will become completely disconnected, so that normal nodes can only search their own content. The result is a high fault susceptibility of 0.99 (e.g., each node can only search itself after the failure). When there is more than one supernode, the search latency increases because searches must travel multiple hops to multiple supernodes. However, the additional supernodes reduce the vulnerability to failure, since a failure only disconnects the normal nodes directly connected to that supernodes leaving other nodes and supernodes in the system connected. When there are 100 supernodes, the network becomes a pure search network, with a search latency of about 4.8 hops and a fault susceptibility of 0.01 (e.g., 1 node

decreased coverage after the failure). Note that this represents near-optimal fault susceptibility. In these results, the 95 percent confidence interval of the fault susceptibility is as large as $\pm$ 7 percent, in the case of 90 supernodes.

Next, Figure 8 shows the results for parallel cluster networks. As the number of clusters increases, the search latency decreases, as shown in Figure 8b. In contrast, the fault susceptibility initially increases, and then slowly decreases (Figure 8a). Note that the scale on the vertical axis of Figure 8a is different from that of Figure 7a. Fault susceptibility is much lower for parallel search clusters. When there are more clusters, the average cluster is smaller, and thus fewer search hops are necessary to reach all of the nodes in the cluster. The result is less search latency. When there is one cluster, the network is a pure search network, and the fault susceptibility is 0.01 (e.g., 1 node decreased coverage after a failure) as with the pure search network. When there is more than one cluster, nodes have copies of other nodes' indexes. When any node F fails, the other nodes in the cluster become unable to search F, but also become unable to search the nodes whose indexes were stored at F. The result is a higher fault susceptibility. However, as the number of clusters increases, the number of nodes that depended on the failed node F decreases, since the clusters become smaller. The result is that fewer nodes are affected by F's failure, and the average fault susceptibility over the whole network again decreases.

4.5 Discussion

Supernode networks are used as an alternative to pure search networks like Gnutella because they reduce the load on normal nodes and thus increase the scalability of the system. However, our results confirm our qualitative observation that the scalability of the system is limited by the capacity of supernodes. In a network with full coverage, the supernodes handle all of the searches, and therefore must have very high capacity. If we are interested in constructing a network that bounds the maximum load, parallel search clusters is a more attractive option. This is useful if there are not many super-high-capacity nodes that can acts as supernodes, because we can better utilize the aggregate resources of the system.

Even if a network is heterogeneous and some nodes are high capacity, we still may choose a parallel cluster network. Consider a situation where we want the system to be robust in the face of failures. We may specify that we want a failure to result in a small decrease in coverage, say, no more than five percent of the nodes become unreachable. This is barely feasible in a supernode network; as shown in Figure 7a, we

would need more than 80 percent of the nodes to be supernodes to achieve such a low fault susceptibility. Then, the network would resemble a pure search network and would lose the scalability advantages of the supernode network. The other alternative, having normal nodes connect to more than one supernode, results in index/index redundancy with an attendant increase in load on the supernodes. On the other hand, a parallel cluster network easily achieves low fault susceptibility without increasing load. As shown in Figure 8a, even the most fault susceptible network of five clusters experiences no more than three percent unavailable nodes after a failure.

5 Related work

Several researchers have examined special algorithms for performing efficient search in peer-to-peer search networks. For pure search networks, these techniques include parallel random walk searches [12, 3], flow control and topology adaptation [13], and iterative deepening search [21]. For networks with indexing, techniques include routing indices [7] and local indices [21]. Each of these approaches is useful for "fixing-up" an existing, inefficient network. Because these techniques can be used to improve the efficiency of the networks described by the SIL model once the networks are built, they are complementary to our own work. Some research has begun into constructing efficient networks a priori; see for example [20].

Others have suggested that the content can be placed in the network to ensure efficiency [5] or that the network topology be reorganized based on the location of the content [8]. Content-based techniques are useful if the content in the network can be appropriately analyzed. Our SIL model is content-agnostic, and describes the inherent properties of the topology of the network. This is useful in general, but especially when the network content is not easily analyzed.

Some work has also focused on constructing P2P networks for end goals other than efficiency. Free-Haven [9] is a peer-to-peer search network that seeks to provide anonymity for content authors, while SOS constructs a P2P overlay to avoid denial-of-service attacks [11].

Moreover, a large amount of attention recently has been given to distributed hash tables (DHTs) such as CHORD [18] and CAN [15]. DHTs focus on finding a document given a unique identifier by routing a query in a peer-to-peer manner. The emphasis on efficient routing of a location query, as opposed to efficient broadcast of a content-discovery query, means that a DHT necessarily has different requirements and topologies. Our model could be extended to model DHTs, although we have not yet done so.

Other studies have performed measurements of deployed peer-to-peer systems. The nature of Gnutella and Napster peers and the connections between them were characterized in [17], and the nature of the Gnutella topology was studied in [16]. In addition, a large amount of work has been done to measure other network topologies, especially the Internet topology as a whole. For example, Tangmunarunkit et al examined the structure of the Internet and idealized topologies that accurately model this structure [19].

6 Conclusion

We have introduced a search/index link model of P2P search networks that allows us to study networks that reduce the load on peers while retaining effective searching and other benefits of P2P architectures. With only four basic link types, our SIL model can represent a wide range of search and indexing structures. This simple yet powerful model also allows us to generate new and interesting variations. In particular, in addition to the supernode and pure search topologies, our SIL model describes topologies such as *parallel search clusters* and *parallel index clusters*. Experimental results from our topology simulator indicate that a parallel search cluster network reduces load by allowing peers to fairly share the burden of answering queries, rather than placing the burden entirely on supernodes. This topology makes better use of the aggregate resource of the system, and is useful in situations where placing an extremely high load on any one peer is unfeasible. Moreover, our results show that other considerations, such as fault susceptibility, may also point to parallel search clusters as an attractive topology.

References

[1] Gnutella. http://gnutella.wego.com, 2002.

[2] Kazaa. http://www.kazaa.com, 2002.

[3] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Phys. Rev. E*, 64:46135–46143, 2001.

[4] G.E.P. Box and M.E. Muller. A note on the generation of random normal deviates. *Annals Math. Stat.*, 29:610–611, 1958.

[5] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. SIGCOMM*, August 2002.

[6] B. F. Cooper and H. Garcia-Molina. Modeling and measuring scalable peer-to-peer search networks (extended version). http://dbpubs.stanford.edu/pub/2002-43, 2002. Technical report.

[7] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, July 2002.

[8] A. Crespo and H. Garcia-Molina. Semantic overlay networks, 2002. Technical Report.

[9] R. Dingledine, M.J. Freedman, and D. Molnar. The FreeHaven Project: Distributed anonymous storage service. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[10] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. SIGCOMM*, 1999.

[11] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *Proc. SIGCOMM*, Aug. 2002.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of ACM International Conference on Supercomputing (ICS'02)*, June 2002.

[13] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Proc. of the 1st Int'l Workshop on Peer to Peer Systems (IPTPS)*, March 2002.

[14] C. Palmer and J. Steffan. Generating network topologies that obey power laws. In *Proc. of GLOBECOM 2000*, Nov. 2000.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. SIGCOMM*, Aug. 2001.

[16] M. Ripeanu and I. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *Proc. of the 1st Int'l Workshop on Peer to Peer Systems (IPTPS)*, March 2002.

[17] S. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. Multimedia Conferencing and Networking*, Jan. 2002.

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM*, Aug. 2001.

[19] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs. structural. In *Proc. SIGCOMM*, Aug. 2002.

[20] B. Yang and H. Garcia-Molina. Designing a super-peer network. http://dbpubs.stanford.edu/pub/2002-13, 2002. Technical Report.

[21] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, July 2002.