Routing Indices For Peer-to-Peer Systems

Arturo Crespo, Hector Garcia-Molina
Stanford University
{crespo,hector}@db.Stanford.edu

Abstract

Finding information in a peer-to-peer system currently requires either a costly and vulnerable central index, or flooding the network with queries. In this paper we introduce the concept of Routing Indices (RIs), which allow nodes to forward queries to neighbors that are more likely to have answers. If a node cannot answer a query, it forwards the query to a subset of its neighbors, based on its local RI, rather than by selecting neighbors at random or by flooding the network by forwarding the query to all neighbors. We present three RI schemes: the compound, the hop-count, and the exponential routing indices. We evaluate their performance via simulations, and find that RIs can improve performance by one or two orders of magnitude vs. a flooding-based system, and by up to 100% vs. a random forwarding system. We also discuss the tradeoffs between the different RI schemes and highlight the effects of key design variables on system performance.

1 Introduction

Peer-to-peer systems (P2P) have grown dramatically in recent years. In a P2P system, distributed computing nodes of equal roles or capabilities exchange information directly with each other. These systems represent an incredible wealth of information allowing users to exchange documents (Freenet [7]), music files (Napster [17], Gnutella [8]), and even computer cycles (Seti-at-home [20]). A key part of a P2P system is document discovery. Our goal is to help users find documents with content of interest across potential P2P sources efficiently.

There are many mechanisms for searching in a P2P system, each with their own advantages and disadvantages. These solutions can be classified in three categories: mechanisms without an index, mechanisms with specialized index nodes (centralized search), and mechanisms with indices at each node (distributed search). Gnutella uses a mechanism where nodes do not have an index and queries are propagated from node to node until matching documents are found. This search mechanism works by flooding the network (or a subset of it) in the hope of finding a match for a query. Although this approach is simple and robust, it has the disadvantage of the enormous cost of flooding the network every time a query is generated.

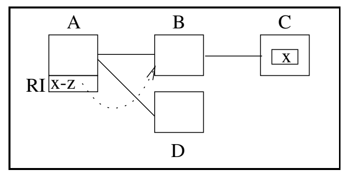


Figure 1. Routing Indices

Centralized-search systems use specialized nodes that maintain an index of the documents available in the P2P system. To find a document, the user queries an index node to identify nodes having documents with the content of interest. These central indices may be built with the cooperation of the nodes (e.g., Napster nodes provide a list of available files at sign-in time) or by crawling the P2P network (as in a web search engine). The advantages of a centralized-search mechanism is efficiency (just a single message is needed to resolve a query). However, a centralized system is vulnerable to attack (e.g., index sites can be shut down by a court order or a hacker attack) and it is difficult to keep the indices up-to-date.

A distributed-index mechanism, the option we will study in detail in this paper, maintains indices at each node. These distributed indices need to be small, so instead of using traditional "destination" indices, we use *Routing Indices* (RIs) that give a "direction" towards the document, rather than its actual location. To illustrate, consider Figure 1 which shows four nodes A, B, C, and D, connected by the solid lines. The document with content "x" is located at node C, but the RI of node A points to neighbor B instead of pointing directly to C (dotted arrow). By using "routes" rather than destinations, the index size is proportional to the number of neighbors, rather than to the number of documents. We can reduce the size of RIs even further by using approximate indices, i.e., by allowing RIs to give a hint (rather than a definite answer) about the location of a document. For example, in the same figure, an entry in the RI of node A may cover documents with contents "x," "y," or "z." A request for documents with content "x" will yield a correct hint, but one for content "y" or "z" will not.

In this paper we study options for building effective RIs, and evaluate their performance. In particular, the contributions of this paper are:

- We introduce Routing Indices, an efficient way of locating content in a P2P system (Sections 3 and 4).
- We present three RIs: the compound, the hop-count, and the exponential routing indices (Sections 5).
- We evaluate the performance of RIs via simulations, and find that RIs can improve performance by one or two orders of magnitude over a flooding-based system, and by 50-100% versus a random forwarding system (Section 7).

2 Related Work

The problem of indexing a P2P network is related to the problem of indexing a distributed database [13]. However, algorithms for indexing distributed databases make two fundamental assumptions that are not applicable to P2P systems: that nodes are stable and connected most of the time, and that the number of nodes is small.

There are several working P2P systems currently available, each with its own "indexing" approach. Napster [17] uses centralized indices, which, as stated before, are vulnerable to attack. Gnutella [8] does not build indices, instead, queries flood a significant part of the network, resulting in a simple but very costly approach as just one query can expand into hundred of thousands of requests through the Gnutella network. Freenet [7] uses an interesting approach to indexing. Each node builds an index with the location of recently requested documents, so if they are requested again, the document can be retrieved at a very low cost.

There are a number of P2P research systems (CAN [18], Oceanstore [14], CHORD [21], Pastry [19], and Tapestry [26]) that can efficiently find documents in a P2P network. The key differences between those systems and our approach is that we do not mandate a specific network structure and that queries are on the content of the documents rather than on document identifiers.

Selecting a neighbor for forwarding a query is also related to traditional routing algorithms [23] such as Bellman-Ford [1, 6]. The major difference with our algorithms is that standard routing algorithms are designed to transmit a packet between two nodes through the shortest route. In our case, we need to get a "packet" from one node to one or *more* nodes so we find the best answers to a query. Also, the destination of a packet is not pre-defined (as in IP routing), but instead it depends on the query contained by the packet. IP routing to multiple destinations (multicast algorithms) has been studied extensively (see for example [16]). However, multicast algorithms require the creation of a relatively stable multicast tree.

The problem of selecting the best database to which to send a query was studied as part of the GlOSS project [12, 9]. However, GlOSS assumes the we are selecting among

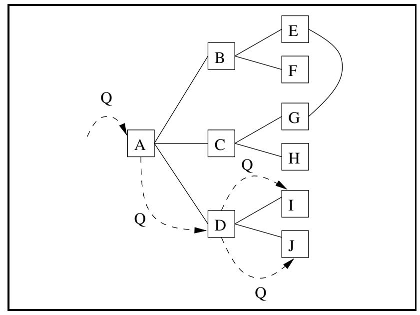


Figure 2. P2P Example

a set of databases, rather than among "paths" that lead to a set of databases.

Some recent work has empirically evaluated P2P systems. A survey and evaluation of centralized-search P2P systems can be found at [24]. An evaluation and description of the present state of Gnutella can be found at [3]. Finally, [25] focuses on search techniques that do not use indexes, although it also studies one type of "local area index." In such indices, a node indexes the content of nodes within "r" hops. However, these indices are not routing indices, they are traditional indices.

3 Peer-to-peer Systems

A P2P system is formed by a large number of *nodes* that can join or leave the system at any time and that have equal capabilities. Each node is connected to a relatively small set of neighbors which in turn is connected to more nodes. In Figure 2, the neighbors of node A are nodes B, C, and D. Note that there might be cycles in the network (such as the one caused by the link between E and G). Each node has a local document database that can be accessed through a local index. The local index receives content queries (e.g., a request for documents containing the words "database systems," a request for documents containing a picture of the sun, etc.) and returns pointers to the documents with the requested content.

3.1 Query Processing in a Distributed-Search P2P System

In a distributed-search P2P system, users submit queries to any node along with a stop condition (e.g., the desired number of results). A node receiving a query first evaluates the query against its own database, returns to the user pointers to any results, and, if the stop condition has not been reached, the node selects one or more of its neighbors and forwards the query to them (along with some state information). In turn, each of the neighbors evaluates the query in a similar fashion, returns result pointers to the user and forwards the query to neighbors.

To illustrate, consider Figure 2. Node A initially receives a query. Node A checks for local results and sends those results to the requesting node. Then, assuming that the stop

Path	# docs	Documents with topics: DB	N	T	L
B	100	20	0	10	30
C	1000	0	300	0	50
D	200	100	0	100	150

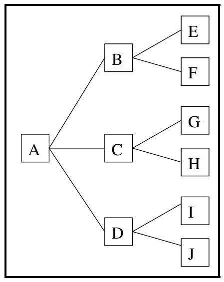


Figure 3. A Sample Compound RI

condition has not been satisfied, node A selects node D as the best neighbor to handle the query and forwards the query to it (dashed arrow). Note that for nodes to be able to verify if the stop condition has been reached, we need to include the number of results found so far as state information in each query-forwarding message. Then D processes the query and selects I as the best neighbor to continue handling the query. Let us assume now that I has processed the query, but not enough results have been found to reach the stop condition. In this case, I returns the query to D which forwards the query to the next best neighbor (J in this case).

Queries can be forwarded to the best neighbors in parallel or sequentially. A parallel approach yields better response time, but generates higher traffic and may waste resources. In this paper, we focus on a sequential forwarding of the queries.

4 Routing indices

In this section we present an example of how the *compound RI* (CRI) works. A formal definition of the algorithms and data structures for RIs can be found in [4]. Later, in Section 5 we present two other RIs: the exponential RI and the hop-count RI.

The objective of a Routing Index (RI) is to allow a node to select the "best" neighbors to send a query to. A RI is a data structure (and associated algorithms) that, given a query, returns a list of neighbors, ranked according to their *goodness* for the query. The notion of goodness may vary but in general it should reflect the number of documents in "nearby" nodes.

As a running example, we will use a P2P system for retrieval of text documents with the network depicted on the right side of Figure 3. For simplicity, this network does not have cycles (we discuss cycles in Section 6). In this system, documents are on zero or more "topics," and queries request documents on particular topics. Each node has a local index for quickly finding local documents when a query is received. Nodes also have a CRI containing (i) the number of documents along each path and (ii) the number of documents on each topic of interest In Figure 3 we show an example of a CRI for node A with three neighbors (paths): B, C, and D. For simplicity, we assume that there are only four topics of interest: databases (DB), networks (N), theory (T), and languages (L). In the figure, we can see that we can access 1000 documents through C (i.e., there are 1000 documents in C, G and H) and that of those documents, 300 are about "networks" and 50 are about "languages."

The RI may be "coarser" than the local indices maintained at nodes. For example, node A could maintain a more detailed local index in which each document is further classified into sub-categories. By keeping a *summary* of the detailed index, we achieved a more compact RI at the cost of introducing "errors" when user queries are based on the subcategories. Specifically, the summarizing of the local index may introduce overcounts or undercounts in the RI. For example, a summarization that groups several subtopics into a single topic (e.g., "indices", "recovery", and "SQL" into "databases") may introduce overcounts on the number of documents available. In fact, a query for documents on "SQL" will be converted into a query for documents on "databases," making us believe that there are many documents on "SQL" whereas in reality there may be few or even none. Summarization can also introduce undercounts. For example, if the summarization uses a frequency threshold (e.g., throws away topics with very few documents), then we may believe that there are no documents on a topic when there are in fact a few.

Given the index, we need now to compute the "goodness" of each node for a query. For CRIs we will use the number of documents that may be found in a path as a measure of goodness. To compute the number of documents, we will use the estimators in [10, 11]. Given that our focus is not on the estimators but on the use and maintenance of RIs, throughout the paper we will use a simplified model where queries are conjunction of subject topics, documents can have more than one topic, and document topics are independent. Thus, we can estimate the number of results in a path as: $NumberOfDocuments \times \prod_i \frac{CRI(s_i)}{NumberOfDocuments}$ where $CRI(s_i)$ is the value for the cell at the column for topic s_i and at the row for a neighbor.

To illustrate, let us assume that A receives a query for documents on "databases" and "languages." We estimate the number of results as $\frac{20}{100} \times \frac{30}{100} \times 100 = 6$ at B, $\frac{0}{100} \times \frac{0}{100} \times 100 = 0$ at C, and $\frac{100}{200} \times \frac{150}{200} \times 200 = 75$ at D. Therefore, the "goodness" of path B will be 6, of path C will be 0, and of path D will be 75. Note that these numbers are just estimates and they are subject to overcounts and/or undercounts. In particular, if there is a strong correlation between the topics "databases" and "languages," then path B may have as many as 20 documents matching the query for topics "databases" and "languages" On the other hand, if there is a strong negative correlation between the topics "databases" and "languages," then there may be no documents in path B on either topic.

A limitation of using CRIs is that they do not take into account the difference in cost due to the number of "hops"

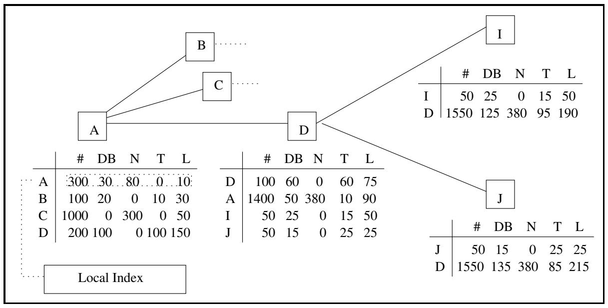


Figure 4. Routing Indices

necessary to reach a document. For example, the documents along path B may all be just one hop away, while the documents along path C may be scattered in a long chain of nodes and finding them would require many messages. Later, we will introduce more sophisticated RIs that do not have this limitation.

In the rest of the section we describe how compound RIs are used, created, and maintained.

4.1 Using Routing Indices

In this subsection we show how RIs, and in particular compound RIs, can improve the performance of query processing in a P2P system. Consider the P2P system described in Figure 3. In Figure 4 we present part of the P2P network with RIs attached to each node. For compactness, we are representing the four topics of interest: database, network, theory, and languages with the letters DB, N, T, and L respectively. In the example we are assuming that the first row of each RI contains the summary of the local index. (This summary can be obtained, e.g., by consolidating subtopics into the main topics, or perhaps by using clustering on a local keyword index to generate topics for each of its documents.) In particular, the summary of A's local index shows that A has 300 documents: 30 about databases, 80 about networks, none about theory, and 10 about languages. The rest of the rows represent a compound RI. In the example, the RI shows that node A can access 100 database documents through D (60 in D, 25 in I, and 15 in J).

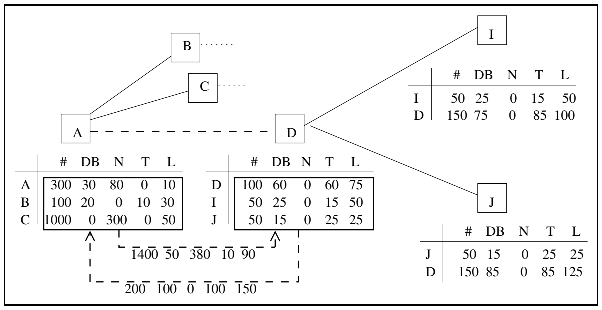
When A receives from a client a query for documents about "databases" and "languages," it first uses the local database to answer the query. If not enough answers are found, it computes the goodness of each path as explained earlier. In this case, the goodness of B, C, and D is 6, 0, and 75 respectively, so A selects D as the best neighbor to forward the query to. In turn, D returns all local results to the client of A and, if not enough results are found, computes the goodness of I and J (25 and 7.5). Since I has the highest goodness, D forwards the query to I. In turn, I returns local results, but it cannot forward the query any further, so (if more results are needed) it returns the query to D which forwards it to its best next neighbor J. Even though the network in the example is very small, a query with a stop condition of 50 documents will generate 9 messages when using flooding; but only 3 messages if we use the RI. Even if we send the query serially in a depth-first fashion to neighbors ranked randomly, we will have 3 messages in the best case and 9 messages in the worst case. The savings in the number of messages when using RIs are the result of forwarding the query only to the nodes that have a high potential of having results.

The storage space required by an RI in a node is modest as we are only storing index information for each neighbor. Furthermore, the storage space per neighbor can be adjusted by increasing or decreasing the level of summarization of the index. Specifically, if s is the counter size in bytes, c is the number of categories, N the number of nodes, and b the branching factor (i.e., number of neighbors), then a centralized index would require $c \times (t+1) \times N$ bytes, while each node of a distributed system would need $c \times (t+1) \times b$ bytes. Thus, the total for the entire distributed system is $c \times (t + 1) \times b \times N$ bytes. Although the RIs require more storage space overall than a centralized index, the cost of the storage space is shared among the network nodes.

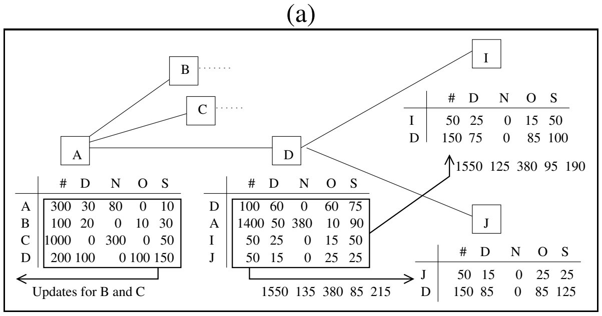
4.2 Creating Routing Indices

Let us now turn our attention to how RIs are created. Returning to our running example, let us assume that initially there is no connection between A and D. The initial state of the system is shown by the solid lines of Figure 5a. When the $A - D$ connection is established, node A informs node D of all the documents that can be accessed through node A. Specifically, node A *aggregates* its RI and sends it to D. In our example, the aggregation is done by adding all the vectors in the RI. (We describe additional aggregation procedures in [4].) Thus, A sends D a vector saying that it has access to 1400 documents ($300 + 100 + 1000$), of which 50 are on databases ($30 + 20 + 0$), 380 on networks ($80 + 0 + 300$), 10 on theory ($0 + 10 + 0$), and 90 on languages ($10 + 30 + 50$). A does not need to send more information as D does not need to know the precise location of the documents, but only that they can be accessed through A. After D receives the aggregated RI from A, it adds an additional row to its RI with A's identifier and A's aggregated RI (as shown in Figure 5b). Note that by aggregating RIs we reduce both the amount of information transmitted and the storage space used. Similarly, D, aggregates its RI (excluding the row for A if it is already in the RI) and sends its aggregated RI to A. Note that the RI creation process at A and D can be done in parallel.

After A and D update their RIs, they need to inform their other neighbors that now they have access to more documents. Thus, D sends an aggregate of its RI to I (excluding I's row) and to J (excluding J's row) as shown in Figure 5b. Then I and J update their RI by replacing the row for D with the new information (not shown in the figure). If I and



(a)



(b)

Figure 5. Creating a Routing Index

J were connected to nodes other than D, they would have to send an update to those nodes too.

4.3 Maintaining Routing Indices

The process of maintaining RIs is identical to the process used for creating them. To illustrate, let us suppose now that client I introduces two new documents about "languages" in its database. To update the RIs of its neighbors, I summarizes its new local index, aggregates all the rows of its compound RI (excluding the row for D), and sends this information to D. Then D replaces the old row for I with the received aggregated RI. In turn, D computes and sends new aggregates to A and J. When receiving the update, A and J update their RIs and compute new aggregates for their neighbors, and so on. For efficiency, we may delay exporting an update for a short time so we can batch several updates, thus, trading RI freshness for a reduced update cost. We can also choose not to send updates when the difference between the old and the new value is not significant. By not sending minor updates, we can again trade reduced update cost for accuracy of the RI.

Finally, a special but frequent update case occurs when a node disconnects from the network. To illustrate, let us suppose that I disconnects from the network. Node D detects the disconnection and updates its RI by removing the row for I. Then D informs its neighbors of the change on the number of documents it can access by sending new aggregates of its RI to them. In turn, the neighbors of D update their RIs and propagate the new information to their neighbors. Note, that we did not need I's participation (or the participation of any other neighbor) in the disconnection

Node	1 Hop					2 Hops				
	#	DB	N	T	L	#	DB	N	T	L
X	60	13	2	5	10	20	10	10	4	17
Y	30	0	3	15	12	50	31	0	15	20
Z	5	2	0	3	3	70	10	40	20	50

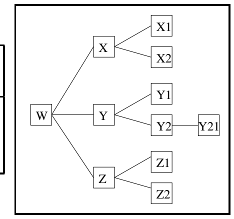


Figure 6. A sample Hop-count RI for node W

process. Not requiring the participation of a disconnecting node is an important feature in a P2P system where nodes can come and go at will.

5 Alternative Routing Indices

5.1 Hop-count Routing Indices

In this subsection, we present an alternative data structure for an RI: a hop-count RI. The main limitation of the compound RI is that it does not take into account the number of "hops" (query forwardings) required to find documents. In the hop-count RI we stored aggregated RIs for each "hop" up to a maximum number of hops. We call this number the *horizon* of the RI. We show in Figure 6 a sample hop-count RI with a horizon of 2 hops. The node with this hop-count RI has three neighbors: X, Y, and Z. With one hop via neighbor X, the node can find 60 documents, out of which 4 are about databases, 2 about networks, 5 about theory, and 10 about systems. The node can also find 20 more documents through X with 2 hops (i.e., at X's neighbors). Note, that we do not have information beyond the horizon with this kind of RI.

The estimator of a hop-count RI needs a cost model to compute the goodness of a neighbor. For example, neighbor X may be preferable over neighbor Y for a query on topic "DB," as through X we would find 13 results with one hop, while it would require two hops to find that many results through Y. On the other hand, we can find more results (31) when going through Y.

If we define cost in terms of number of messages (we will expand on the notion of cost in Section 7), then we can define the goodness of a neighbor as the ratio between the number of documents available through that neighbor and the number of messages required to get those documents. So a neighbor that allows us to find 3 documents per message is better than a neigbor that allows us to find 1 document per message.

A simple model that allows us to compute this ratio is the regular-tree cost model. The model assumes that document results are uniformly distributed across the network and that the network is a regular tree with fanout F. Under these assumptions, it takes F^h messages to find all documents at hop h. Threfore, we can compute the number of documents

Path	#	DB	N	T	L
X	66.67	16.33	5.33	6.33	15.67
Y	46.67	10.33	3.00	20.00	18.67
Z	28.33	5.33	13.33	9.67	19.67

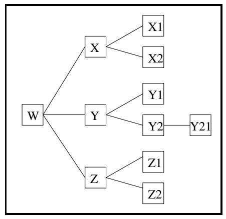


Figure 7. A sample Exponential Routing Index for Node W

per message by dividing the expected number of result documents at each hop by the number of messages needed to find them. Formally, we define the goodness ($goodness_{hc}$) of $Neighbor_i$ with respect to query Q for hop-count RI as: $goodness_{hc}(Neighbor_i, Q) = \sum_{j=0..h} \frac{goodness(N_i[j],Q)}{F^j}$, where h is the horizon of the hop-count RI, $goodness()$ is the estimator for CRI, and $N_i[j]$ is the RI entry for j hops through $Neighbor_i$. In our example, if we assume $F = 3$, the goodness of X for a query about "DB" documents would be $13 + 10/3 = 16.33$ and for Y would be $0 + 31/3 = 10.33$, so we would prefer X over Y.

Creation and update mechanisms for hop-count RIs are described in the extended version of this paper [4].

5.2 Exponentially aggregated RI

The hop-count RI is effective in taking into account the number of hops. However, this benefit comes at a higher storage and transmission cost than the compound RI. Moreover, in Section 7.2 we will see that the hop-count RI performance is negatevely affected by the lack of information beyond the horizon (a hybrid CRI-HRI overcomes this disadvantage, but it still does not solve the storage and transmission cost problem). In this subsection we present an alternative index structure, the exponential aggregated RI, that overcomes these shortcomings at the cost of some *potential* loss in accuracy.

The exponentially aggregated RI stores the result of applying the regular-tree cost formula to a hop-count RI. Specifically, each entry of the ERI for node N contains a value computed as: $\sum_{j=1..th} \frac{goodness(N[j],T)}{F^{j-1}}$, where th is the height and F the fanout of the assumed regular tree, $goodness()$ is the Compound RI estimator, $N[j]$ is the summary of the local index of neighbor j of N, and T is the topic of interest of the entry.

We show in Figure 7 an exponentially aggregated RI computed from our sample network of Figure 6. In the figure, we assume that the neighbors of X, Y, and Z are leaf nodes and that the fan out of the tree is 3. The entries for topic "DB" for X and Y have the values $13 + 10/3 = 16.33$ and $0 + 31/3 = 10.33$.

The exponential RI makes the same assumptions as the regular-tree cost model and may not be realistic in some configurations, but it can still be used as an approximate

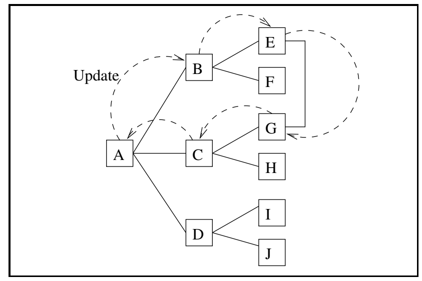


Figure 8. Cycles and Routing Indices

index. There is a fundamental difference between the exponential RI and the hop-count RI. While the hop-count RI does not have any information beyond the horizon, with the exponential RI we can keep information for all nodes accessible from each neighbor in the RI. In fact, we will see in Section 7.2 that the exponential RI outperforms the hop-count RI in most cases.

Creation and update mechanisms for exponential RIs are described in the extended version of this paper [4].

6 Cycles in the P2P Network

In this section we analyze how cycles affect the process of creating and updating RIs as well as strategies to minimize those effects. To illustrate the effect of cycles, we will use the initial setup of Figure 3, but with the network depicted in Figure 8 (with the cycle $A - B - E - G - C - A$). Let us assume that node A adds to its database two new "theory" documents and sends a new aggregate of its RI to B. Node B sends the update to its E and F neighbors, prompting E to send an update to G, which sends an update to C, which finally sends an update to A. When A receives this update, it will mistakingly assume that more "theory" documents are available via node C, but those extra documents are its own. Worse than that, the update from C will prompt A to send an update to its neighbors, informing them that they can access two more theory documents, creating an infinite loop. There are three general approaches for dealing with cycles:

No-op solution: No changes are made to the algorithms; this solution only works with the hop-count and the exponential RI schemes. In the case of the hop-count RI, cycles longer than the horizon will not affect the RI. However, shorter cycles will affect the hop-count RI but their effect will be limited if we are using the regular-tree cost model. However, the cycle increases the cost of creating/updating the hop-count RI as updates sent by a node return to it (via the cycle), causing the node to send a new update to all its neighbors (which in turn send the update back to the node again). The cycle is broken when the update reaches the horizon of the hop-count RI. Similarly, in the case of the exponential RI, updates are sent back to the originator. However, the effect of the cycle will be smaller and smaller every time the update is sent back (due to the exponential decay), until the difference between the old update and the new update is small enough and the algorithm stops propagating the update. As in the hop-count RI, the main effect of the cycle is the increase in cost of creating/updating the RI.

Cycle avoidance solution: In this solution we do not allow nodes to create an "update" connection to other nodes if such connection would create a cycle. The techniques for cycle avoidance have been extensively studied (see [22] for a survey) and we do not cover it further in this paper. The main disadvantage of this approach is that in the absence of global information we may end with a suboptimal solution.

Cycle detection and recovery: This solution detects cycles sometime *after* they are formed and, after that, takes recovery actions to eliminate (or neutralize) the effect of the cycles. In the example of Figure 8, cycles can be detected by having the originating node of a query or an update, let us say A, include a unique message identifier in the message. Any update (or query forwarding) that any other node sends as a consequence of this message will have the original message identifier. If a message with the same identifier returns to A (let us say from C), then A knows that there is a cycle and that a recovery procedure should be started.

7 Experimental Results

In this section, we evaluate search mechanisms for P2P systems. First, we present our model of a P2P system. Then we introduce a simulation tool that allows us to evaluate different search mechanisms efficiently. We then use our tool to study the performance of different mechanisms as well as the factors that affect their performance. We close the section with an analysis of the cases where RIs can be used effectively.

7.1 Modeling search mechanisms in a P2P system

Our goal in this subsection is to identify the elements of a typical search mechanism in a P2P system, so we can model each element and study its impact on performance. A typical P2P system is a network of nodes T where each node contains a set of documents. Users send requests consisting of a query q and a $stopCondition$ to a node of the P2P system. The objective of the search mechanism is to answer those requests by obtaining a set of documents of size $stopCondition$ that matches the query q. In addition, search mechanisms allow for updates such as the addition of nodes or new documents.

To process queries and updates we use the mechanisms described in the previous sections (CRIs, HRIs, ERIs). For comparison purposes, we add an additional mechanism: *No-RI*. Instead of using an RI to choose the best neighbor to forward a query, this search mechanism simply chooses a random neighbor.

To further model the elements of a search mechanism, we need to define the topology of the network, the location of document results, how costs are measured, and cycle policies (described in detail in [4]).

Parameter Name	Description	Value
Network Configuration Parameters		
$NumNodes$	Number of nodes in the network	60000
T	Topology of the P2P network	tree
F	Branching factor for tree topology	4
EL	Extra links for tree+cycle topology	10
o	Outdegree exponent for power law	-2.2088
Document Distribution Parameters		
QR	Total Number of Query Results	3125
D	Document distribution	80/20
RI Parameters		
$Creation_{size}$	Avg. size of creation/update message	1000 b
$Query_{size}$	Avg. size of query message	250 b
$StopCondition$	Number of documents requested	10
H	Horizon for HRIs	5
A	Decay for ERs	4
c	RI Compression	0%
$minUpdate$	min % diff for update propagation	1%

Figure 9. Simulation Parameters

The topology of the network defines the number of nodes and how they are connected. In our model, we consider three kinds of network topologies: a tree, a tree with added cycles, and a power-law graph [15]. The first topology, a tree, is of interest because it does not have cycles (a good base case for our algorithms). For the second topology, we start with a tree and we add extra vertices at random (creating cycles) so we can measure the impact of cycles on the search mechanisms. The third topology, a power-law graph, is considered a good model for P2P systems and allows us to test our algorithms against a "realistic" topology [5].

We model the location of document results using two distributions: uniform and an 80/20 biased distribution. Under the uniform distribution all nodes have the same probability of having each document result (nodes can have more than one document result). The second distribution assigns uniformly 80% of the document results to 20% of the nodes (and the remaining 20% of the documents to the remaining 80% of the nodes).

Modeling the cost of a search mechanism is a complex task. We can model the cost based on the resources used in the P2P system (e.g., network, storage space, or processing power) or based on the user experience (e.g., mean query response time, query throughput, or query turnaround time). In current P2P systems, the critical resource is the network [3] as many of the nodes are connected through links with limited bandwidth (e.g., dial-up connections, DSL, and cable). Therefore, in this paper we focus on the network and we use the number of messages generated by each algorithm as a measure of cost. This is not to say that user-based factors (such as response time) are not important, but by focusing on the network we are also improving those factors.

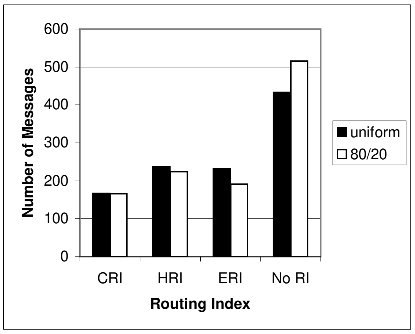


Figure 10. Comparison of CRI, HRI, and ERI

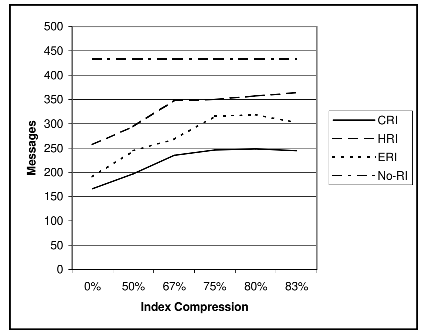


Figure 11. Effect of Overcounts

7.2 Evaluating P2P Search Mechanism

In this section we experimentally compare the three proposed RIs: compound RI (CRI), hop-count RI (HRI), and exponential RI (ERI) against each other and against the No-RI search mechanism. We also explore how the performance of the RIs are affected by approximate indices, different stop conditions, document result distribution, number of document results, and network topology.

To evaluate search mechanisms we built a simulator. The simulator receives as input a P2P model and an operation that can be an update or a query. The simulator iterates over different network topologies and document result locations, and outputs the average number of messages necessary to perform the operation plus a confidence interval. All results were computed with at least a 95% confidence interval of having a relative error of 10% or less. Parameters are set to the base values presented in Figure 9 unless stated otherwise. In the extended version of this paper [4], we explain in detail the operation of the simulator as well as the choice of the base values for the parameters.

Figure 10 shows the number of messages needed to process a query when using each kind of RI for two document distributions. The advantage of using RIs is obvious, we are able to reduce the number of messages by half when compared to not using RI. Among the RIs, CRI had the best performance, followed by the ERI and HRI. This difference in performance is a function of the number of nodes used to generate the index. In particular, CRI uses all nodes in the network, HRI uses nodes within a predefined a horizon, and ERI uses nodes until the exponentially decayed value of an index entry reaches a minimum value (resulting in using more nodes than HRI, but fewer than CRI). This result shows that the more nodes an RI uses to compute the goodness of a path, the better the RI is. However, we will see that a larger number of nodes implies a higher update cost.

In Figure 10 we also present the effect of using two document distributions, an 80/20 biased and a uniform document distribution. Surprisingly, a 80/20 biased distribution does not improve the performance of RIs much, but it degrades the performance of a No-RI search mechanism. To understand this result, we analyzed traces of our simulations. In the case of RIs under a 80/20 document distribution, the algorithm directed the queries to nodes with a high number of document results, but to reach those content-loaded nodes the queries needed to travel through several nodes that had very few or no document results. On the other hand, under a uniform document placement, the algorithms followed good paths where at each node it obtained a few results. In summary, in one case, we collected results by traveling over an almost empty path to a full node, while in the other case, we collected results by traveling through a path of a similar length where each node contributes a few documents. The overall result was that the number of messages per document result was about the same in both cases. In the case of the No-RI approach, an 80/20 document distribution penalizes performance as the search mechanism needs to visit a number of nodes until it finds a content-loaded node (generating a large number of messages in the process).

We also compared RIs against non-index/flooding solutions such as Gnutella. In that case, RIs reduce the number of messages by two orders of magnitude (graph not shown). However, this comparison is not completely fair as non-index systems find all results (versus only a user-defined number of results when using RIs) and they potentially have a better response time (as queries are processed in parallel, rather than sequentially). However, finding *all* results may be an overkill for most applications as users rarely examine more than the first 10 top results returned by a search engine [2]. In addition, low response times may be hard to achieve in Gnutella-like systems because of network contention created by tens of thousands of messages generated by each query.

We studied how increases in the requested number of documents affects RIs (graph not shown). As expected, the higher the number of requested documents, the more messages are generated. However, the increase on the number of messages is linear for all RIs, showing that they scale well on this parameter. We also analyzed the effect of a decrease on the number of document results available (graph not shown). In that case, we obtained a very similar graph

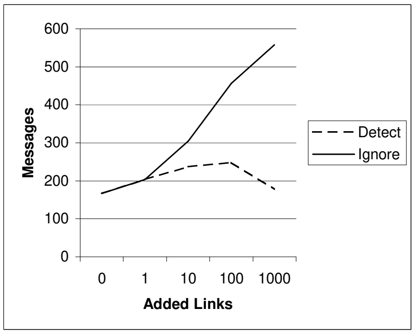


Figure 12. Effect of Cycles

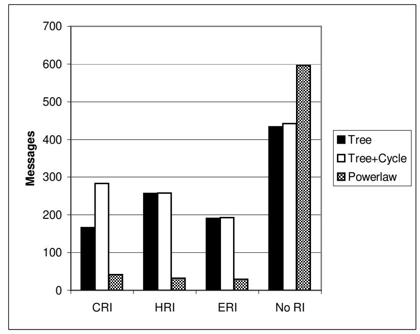


Figure 13. Network topology

where the number of messages grows linearly with a reduction of document results.

We now investigate how errors in RIs, and particularly overcounts, affect RI performance. As discussed in Section 4, errors can occur in a variety of ways; here we select one scenario to illustrate. We assume that documents are organized into categories, and the index is a hash table of categories. Several categories may hash to the same bucket, so the count in a bucket represents the aggregate number of documents in those categories. For example, suppose there are 3 "database" documents, and 2 "network" ones. If "database" and "network" hash to the same bucket, the consolidated bucket will have a count of 5 documents. If a query is looking for "database" documents, when using the RI, we will believe that they are 5 of them, when in reality there are only 3 (an overcount). (Instead of adding the original document counts, we could have also chosen to take the minimum of them, generating undercounts; or we could have averaged them, generating mixed errors.)

As the table size is reduced, more and more overcounts occur. In Figure 11, we show the performance of CRI, HRI and ERI, as a function of the "index compression." For example, a 50% value means that the number of hash table buckets is half the number of categories, while 83% represents a table with one-sixth the categories. (Note that the scale is not linear.) From the graph, we can see that even though there is a loss of performance because of overcounts, this loss is modest even in the case of significant reductions on the size of the index. Moreover, query processing when using RIs is still far cheaper than No-RI even if we use the highest compression levels. We conducted additional experiments for undercounts and mixed errors as well as for other error models. Those experiments had similar results to the one presented here and are omitted for brevity.

In Figure 12 we study how ERIs perform when cycles are added to a tree network. Cycles are created by adding random links to a tree network with $NumNodes - 1$ links. As expected, the number of messages increases as we add more links and cycles are created. The increase in the traffic is the result of two factors. First, there is a loss of accuracy of the RI. In the case of the "detect and recover" policy, this loss is the result of missing the best route to the results (as explained in Section 6), and in the case of the "no-op" policy the accuracy suffers because overcounts introduced in the generation of the RI. Second, during query processing the number of messages increases. In the case of the "detect and recover" policy, those extra messages are the result of return-queries messages sent by a node that detects a cycle. In the case of the no-op policy, extra messages are generated when we traverse a cycle more than once, finding document results that were already found in a previous iteration. In the figure, we observe that the increase in the number of messages is small if we use the "detect and recover" policy, but it can be significant if we choose to ignore cycles. An unexpected result is that the number of messages drops if we add a large number of links. This drop is the result of the added connectivity that additional links create, which allows shorter routes to document results. Similar performance to the one presented for ERI is shown by HRI and CRI (when using the ignore-detect policy, as CRI is not guaranteed to terminate when using the no-op policy).

In Figure 13 we study how RIs perform in different network topologies. The result of our analysis is surprising at first glance: RIs perform better in a power-law network than in a tree network. There are two reasons for this result. First, while in a tree-like network the connectivity of every node (except leaf nodes) is the same, in a power-law network a few nodes have a significantly higher connectivity than the rest. By analyzing the traces of our simulation, we found that the query algorithms actually direct the queries towards those well-connected nodes. After getting to these highly connected nodes, a large number of results is collected without having to issue many messages. The second reason for this performance improvement is that power-law distributions generate network topologies where the average path length between two nodes is lower than in tree topologies. Lower path length improves performance as we need less messages to go from node to node. On the other hand, these same two factors hinder the performance of the No-RI approach. In a power-law network, there are very few

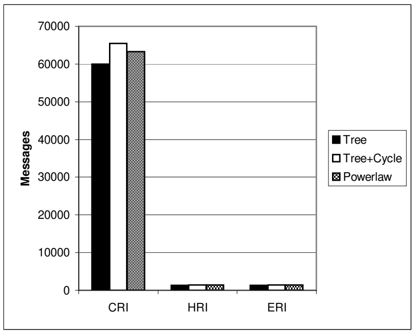


Figure 14. Updates and Network Topology

highly connected nodes and it is not easy to find them if we just move randomly as No-RI does. As a result, the No-RI approach visits a significant number of nodes until it finally stumbles onto a highly connected node (generating a large number of messages in the process). Shorter path length also hinders No-RI as bad decisions about which neighbor to contact often result in return-query messages.

Figure 14 shows the number of messages needed to update each kind of RI for each network topology. The graph shows the cost of one batch of updates, propagated throughout the network. In the graph we can see that the cost of CRI is much higher when compared with HRI and ERI. This is the result of CRI propagating the update to all nodes, while HRI and ERI only propagate the update to a subset of the network. This result confirms that the additional information and better query performance of CRIs come with a high price tag. On the other hand, HRIs and ERIs have very low update costs and their query processing performance is very close to the one of CRIs, making them an excellent choice as the search mechanism of a P2P system. In the graph we can also see that network topology has little impact on the update performance of RIs, as there is low or no correlation between the network topology and the number of nodes that needs to be updated.

We also studied the tradeoff between query and update costs for RIs (figure not shown). For a system processing 1032 queries per minute (the average query load observed on a section of the Gnutella network [24]), the point where the total cost of using ERIs is the same as the cost of a system without RIs was at an update load of 36 updates per minute. In practice we would expect the number of updates to be way below 36 per minute, especially since it is not that critical to keep indexes up to date and updates can be batched together. Thus, the search improvements afforded by RIs are seldom outweighed by the cost of updating them.

8 Conclusions

In this paper we studied how to improve the efficiency of content search in a peer-to-peer system. We achieve greater efficiency by placing Routing Indices in each node. Three possible RIs: compound RIs, hop-count RIs, and exponential RIs were proposed and experimentally evaluated using simulations. From our experiments we conclude that ERIs and HRI offer significant improvements versus not using an RI, while keeping update costs low. We belive that routing indices, and in particular ERIs and HRIs, can help improve the search performance of current and future P2P systems.

References

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
[2] S. Brin. The anatomy of a large-scale hypertextual web search engine. In *7th WWW Conference*, 1998.
[3] Clip2.com, at: http://dss.clip2.com/gnutella.html. *Gnutella: To the Bandwidth and Beyond*.
[4] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. Technical report, Computer Science Department, Stanford University, March 2002. At http://dbpubs.stanford.edu/pub/2001-48.
[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, 1999.
[6] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.
[7] Freenet. *At http://freenet.sourceforge.com.*
[8] Gnutella. *At http://gnutella.wego.com.*
[9] L. Gravano and H. Garcia-Molina. Generalizing gloss for vector-space databases and broker hierarchies. In *Proceedings of VLDB*, 1995.
[10] L. Gravano, H. Garcia-Molina, and A. Tomasic. The effectiveness of gloss for the text-database discovery problem. In *Proceedings of SIGMOD*, 1994.
[11] L. Gravano, H. Garcia-Molina, and A. Tomasic. Precision and recall of gloss estimators for database discovery. In *Proceedings of PDIS*, 1994.
[12] L. Gravano, H. Garcia-Molina, and A. Tomasic. Gloss: Text-source discovery over the internet. *TODS*, 2000.
[13] D. Kossman. The state of the art in distributed queyr processing. *ACM Computing Survey*, September 2000.
[14] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS*, 2000.
[15] B. H. L.A. Adamic A.R. Puniyani, R. M. Lukose. Search in power-law networks. Technical report, HP Labs, 2001. Available at http://www.hpl.hp.com/shl/papers/plsearch/.
[16] C. K. Miller. *Multicast Networking and Applications*. Addison Wesley, 1998.
[17] Napster. *At http://www.napster.com.*
[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, August 2001.
[19] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.
[20] Seti At Home. *At http://setiathome.ssl.berkeley.edu.*
[21] 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. ACM SIGCOMM*, 2001.
[22] A. Tanenbaum and A. Woodhull. *Operating Systems Design and Implementation*. Prentice-Hall, Inc., 1999.
[23] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 1996.
[24] B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *VLDB*, 2001.
[25] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *ICDCS*, 2002.
[26] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.