UNIVERSITY OF CALIFORNIA
RIVERSIDE

Information Retrieval in Peer-to-Peer Systems

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science
in
Computer Science

by
Demetrios Zeinalipour-Yazti
June 2003

Thesis Committee:
Dr. Dimitrios Gunopulos, Chairperson
Dr. Vana Kalogeraki
Dr. Chinya V. Ravishankar

The Thesis of Demetrios Zeinalipour-Yazti is approved:

Committee Chairperson

University of California, Riverside

ACKNOWLEDGMENTS

The following faculties, student fellows, friends, and family members have contributed to this dissertation and played a significant role throughout this work. Without each of them, this work would be very difficult to accomplish.

First, I would like to thank Prof. Dimitrios Gunopulos for his support during this work, his continuous guidance and all the fruitful discussions that laid the foundation of the research presented in this thesis. I also thank Prof. Vana Kalogeraki for her encouragement, her important feedback throughout our collaboration. Finally I would like to thank Prof. Chinya Ravishankar for taking the time out of his busy schedule to be on my thesis committee.

I thank the colleagues and friends of the database lab and the department, for sharing the burden of the research through the years of my graduate study here at UCR. Theodoros Folias has made all the projects we ’ve been working on more interesting and colorful. Thanks to Anil, Dimitris, Foula, Jessica, Marios, Michalis, Mirella, Sharmila and all the other people here at UCR for the good times spent together.

I would like to thank my parents Djalal and Androula Zeinalipour, as well as my brother Constantino and his fianceé Erika. They always supported me in pursuing my objectives and I own them my beliefs in education, and my willingness to learn and go through the requirements of the Masters Degree. Last I would like to thank my fianceé Christina, which stood on my side providing me with courage and support during the two year period we were apart. I dedicate this work to her.

An earlier version of this work appeared in the proceedings of the ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, pp 300-307, Nov. 2002

DEDICATION

.

To my fianceé Christina

ABSTRACT OF THE THESIS

Information Retrieval in Peer-to-Peer Systems

by

Demetrios Zeinalipour-Yazti

Master of Science, Graduate Program in Computer Science
University of California, Riverside, June 2003
Prof. Dimitrios Gunopulos, Chairperson

Peer-to-Peer systems are application layer networks which enable networked hosts to share resources in a distributed manner. An important problem in such networks is to be able to efficiently search the contents of the other peers. Existing search techniques are inefficient because they are either based on the idea of flooding the network with queries or because they require some form of global knowledge.

We propose the Intelligent Search Mechanism (ISM) which is an efficient, scalable but yet simple mechanism for improving the information retrieval problem in Peer-to-Peer systems. Our mechanism is efficient since it is bounded by the number of neighbors and scalable because no global knowledge is required to be maintained. *ISM* consists of four components: A *Profile Mechanism* which logs query-hits coming from neighbors, a *Cosine Similarity* function which calculates the closeness of some past query to a new query, *RelevanceRank* which is an online ranking mechanism that ranks the neighbors of some node according to their potentiality of answering the new query

and a *Search Mechanism* which forwards a query to the selected neighbors.

We deploy and compare ISM with a number of other distributed search techniques. Our experiments are performed with real data over PeerWare, our middleware simulation infrastructure which is deployed on 50 workstations. Our results indicate that ISM is an attractive technique for keyword searching in Peer-to-Peer systems since it achieves in some cases 100% recall rate by using only 50% of the messages used in the flooding algorithm. Further its performance is also superior with respect to the total time for satisfying a query. Finally our algorithm improves over time as some node develops more knowledge.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Peer-to-peer (P2P) networks are increasingly becoming popular because they offer opportunities for real-time communication, ad-hoc collaboration [12] and information sharing [23, 11] in a large-scale distributed environment. Peer-to-peer computing is defined as the sharing of computer resources and information through direct exchange. The most distinct characteristic of P2P computing is that there is symmetric communication between the peers; each peer has both a client and a server role. The advantages of the P2P systems are multi-dimensional; they improve scalability by enabling direct and real-time sharing of services and information; enable knowledge sharing by aggregating information and resources from nodes that are located on geographically distributed and potentially heterogeneous platforms; and, provide high availability by eliminating the need for a single centralized component.

In this thesis we consider the *information retrieval* problem in the P2P networks. Assume that each peer has a database (or collection) of documents (see figure 1.1) which represents the

knowledge of the peer. Each peer shares its information with the rest of the network through its neighbors. A node searches for information by sending `query` messages to its peers. Without loss of generality we assume that the queries are collections of keywords and that a querying peer is interested in finding all the documents that contain a set of keywords. A peer receiving a query message evaluates the constraint locally against its collections of documents. If the evaluation is successful, the peer generates a reply message to the querying peer which includes the identifier of all the documents that correspond to the constraint. Once a querying peer receives responses from all the peers it afterwards decides which documents to download. Each document can be associated with a unique `documentId` (using for example a hash function on the contents of a document) to uniquely identify the same documents from different peers.

Note that searching based on the file contents is not possible in most current P2P systems today [4, 27]. In those systems searching is done using file identifiers instead (such as the name of the file or the documentId). Although this allows deployment of efficient search and indexing techniques it restricts the ability of P2P users to find documents based the contents of the documents.

To solve the search problem, most current systems either rely on centralized control [23] or on query message flooding mechanisms [11]. The second approach (broadcasting the query) can easily be extended to solve the problem we consider here. This can be achieved by modifying the query message to include the query terms instead of the desired file identifier. This approach is best suited for unstructured peer-to-peer networks, since all functionality (including indexing and resource sharing) is decentralized. Such systems do not use peers with special functionality. Gnutella is an example of such a system.

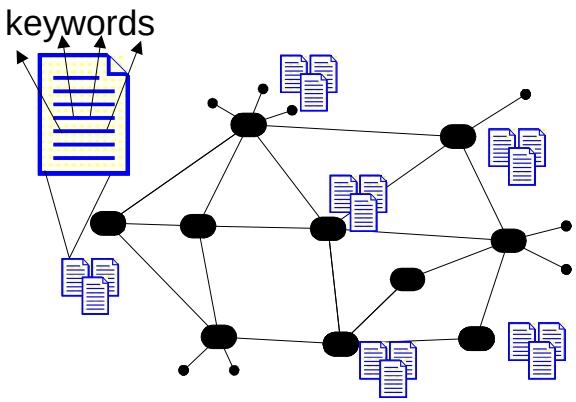

Figure 1.1: Information Retrieval in P2P systems.

In hybrid peer-to-peer networks [16, 28], one (or possibly more) peers have additional functionality in that they become partial indexes for the contents of other peers. Each peer, as it joins the network uploads a list of its files to the index server. Commercial information retrieval systems such as web search engines (e.g. Google [13]) are using a similar approach for indexing the web. These are centralized processes that exploit large databases and parallel approaches to process queries, and work extremely well. In the P2P information retrieval context however, they have several disadvantages. The biggest disadvantage is that the index needs to be an inverted index over all the documents in the network. This means that the index node has to have sufficient resources to setup and maintain such settings. Although hardware performance and costs have improved, such centralized repositories are still expensive and prohibitive in dynamic environments where nodes are joining and leaving.

Table 1.1: Top 20 Queries on Gnutella in June 2002. (inappropriate queries marked with '_'). The total set includes 15 million query messages and the last column the percentage out of all the queries.

#	Query	Occur.	%	#	Query	Occur.	%
1	divx avi	588,146	3,88%	**11**	divx	24,363	0,16%
2	spiderman avi	50,175	0,33%	**12**	spiderman	23,274	0,15%
3	p___ mpg	39,168	0,25%	**13**	xxx avi	22,408	0,14%
4	star wars avi	38,473	0,25%	**14**	capture the light	21,651	0,14%
5	avi	29,911	0,19%	**15**	buffy mpg	20,365	0,13%
6	s__ mpg	27,895	0,18%	**16**	g__ mpg	20,251	0,13%
7	Eminem	27,440	0,18%	**17**	buffy avi	19,874	0,13%
8	eminem mp3	25,693	0,16%	**18**	t___ mpg	19,492	0,12%
9	dvd avi	25,105	0,16%	**19**	seinfeld v____	18,809	0,12%
10	b_____	24,753	0,16%	**20**	xxx mpg	18,686	0,12%

1.1 Motivation

Our proposed algorithm works well in environments where there is high locality of similar queries. In order to see what the real trends are, we made an extensive analysis of the network traffic found in a real P2P network [30]. In June 2002 we crawled the Gnutella network with 17 workstations for 5 hours and gathered 15 million query messages. Table 1.1 presents the ranking of the top 10 queries. We can clearly see that most queries are submitted in large numbers and hence there exist a high locality of specific queries. This observation is exploited by our proposed Intelligent Search Mechanism. By making only local decisions we can intelligently route query messages to those neighbors that have high probability to lead us to the most relevant answers.

1.2 Our Contribution

In this thesis we consider a fully distributed technique for addressing the information retrieval problem in pure P2P networks. We propose the Intelligent Search Mechanism (ISM), an efficient, scalable but yet simple mechanism for improving the information retrieval problem in P2P systems.

Our algorithm exploits the locality of past queries by using well established techniques from the Information Retrieval field. To our knowledge no previous work has been done using similar techniques. Finally our technique is distributed and a node can make autonomous decisions without coordinating with any other peers which therefore both reduce networking and processing costs.

The thesis is organized as following: Section 2 discusses related work and a number of different techniques for Information Retrieval in P2P systems. Section 3 presents the components of the Intelligent Search mechanism. In section 4 we make an analytical study of the compared techniques. Section 5 describes PeerWare, our distributed simulation infrastructure which is deployed on 50 machines. Section 6 presents our experimental results under various scenarios and section 7 concludes this thesis.

Chapter 2

Information Retrieval in P2P Networks

In this chapter we consider a number of different techniques to search in P2P networks. The techniques presented in sections 2.1 to 2.7 are appropriate in the context of Information Retrieval since users can search based on keywords. In section 2.8 and 2.9 we present two distributed lookup protocols which allow peer-to-peer applications to efficiently locate a node that stores a particular object. These two techniques are not applicable in the context of Information Retrieval since we are searching based on keywords rather than identifiers.

For the next sections we consider a network of n nodes (peers), with average degree d (with $d << n$), that is, each peer is directly connected to around d other peers. For a given peer u, the peers of u, $N(u)$ are those nodes in the network that have a direct connection to u. Figure 2.1 shows an example of a peer-to-peer network. Each node in the figure represents a peer and an edge corresponds to a direct communication between the peers.

Each peer possesses and maintains a set of documents, which can be also made available

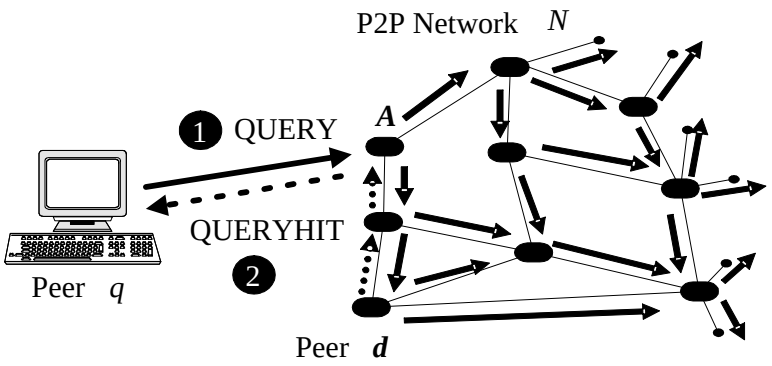

Figure 2.1: Searching in a peer-to-peer network with Breadth First Search BFS: Each peer forwards the query to all its neighbors.

to his peers. This set represents the knowledge of the peer. We assume that each document is stored in semi-structured form: for each document we have a set of attributes, such as, date, place, title as well as text. We assume that the queries are unstructured and that they are sampled from the documents collections.

2.1 The ”naive” Breadth First Search (BFS) Technique

BFS is a technique widely used in P2P file sharing applications, such as Gnutella. BFS sacrifices performance and network utilization in the sake of its simplicity. The BFS search protocol in the peer-to-peer network N (see figure 2.1) works as follows. A node q issues search messages when it wants to search for data and information among its peers. The node generates a `Query` message with and propagates the message to all of his neighbors. When a peer A receives a `Query` request, it first forwards the query to all the peers, other that the sender, and then searches its local repository for relevant matches. If some node d receives the query and has a match, d generates a `QueryHit` message to transmit the result. The `QueryHit` message includes information such

as the number of corresponding documents and the network connectivity of the answering peer. If for example node q receives a `QueryHit` from more than one peer, it may choose to do the actual download from the peer with the best network connectivity. `QueryHit` messages are sent along the same path that carried the incoming `Query` messages.

The disadvantage of BFS is that a query is consuming excessive network and processing resources because a query is propagated along all links (including nodes with high latencies). Therefore the network can easily become a bottleneck. One technique to avoid flooding the whole network with messages for a single query is to associate each query with a time_to_live (TTL) field. The TTL field determines the maximum number of hops that a given query should be forwarded. In a typical Gnutella search the initial value for the TTL is usually 7, which is decremented each time the query is forwarded. When the TTL becomes 0, the message is no longer forwarded. We will show that this technique is not adequate for reducing messaging and that we can further improve on that.

2.2 The Random Breadth-First-Search (RBFS) Technique

In [20] we propose and evaluate the *Random Breadth-First-Search (RBFS)* technique that can dramatically improve over the naive BFS approach. In RBFS (see figure 2.2) each peer A forwards a search message to only a fraction of its peers. Node A randomly selects a subset of its peers to propagate the search request. The fraction of peers that are selected is a parameter [1] to the mechanism. The advantage of this technique is that it does not require any global knowledge. Every node is able to make local decisions in a fast manner since it only needs to select half of its incoming

[1] In our experiments we used a fraction of 0.5 (a peer propagates the request to half its peers, selected at random).

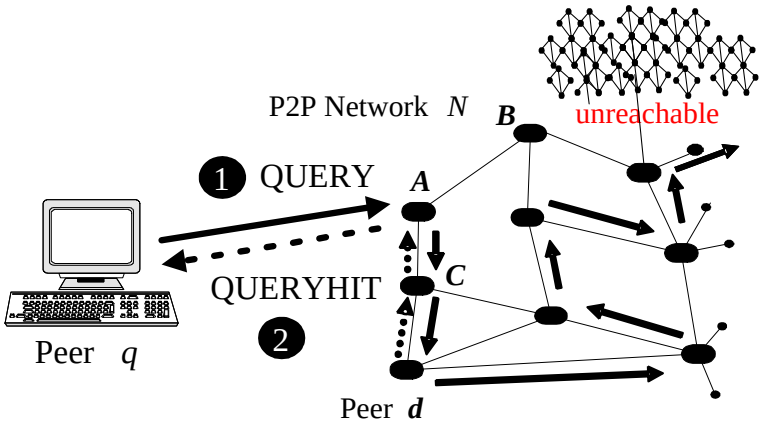

Figure 2.2: Searching in a peer-to-peer network with Random Breadth First Search RBFS: Each peer forwards the query to a subset of its neighbors.

and outgoing connections. On the other hand, this algorithm is probabilistic. Therefore some large segments of the network may become unreachable because a node was not able to understand that a particular link would lead the query to a large segment of the graph.

2.3 Searching using Random Walkers

In [19] a search technique based on random walks is presented. In the proposed algorithm each node forwards a query message by selecting a random neighbor and the query message is called a *walker*. In order to reduce the time to receive the results the idea of the *walker* is extended to a *k-walker* which after T steps is expected to reach approximately the same number of nodes as 1 walker after kT steps. In order to thwart duplicate messages each node may retain states. This algorithm resembles much the Random Breadth First Search (RBFS) Technique with the difference that in RBFS each node forwards a query message to a fraction of its neighbors and that in RBFS the incurred increase in messages is exponential while in the k-Walker model the messages used is

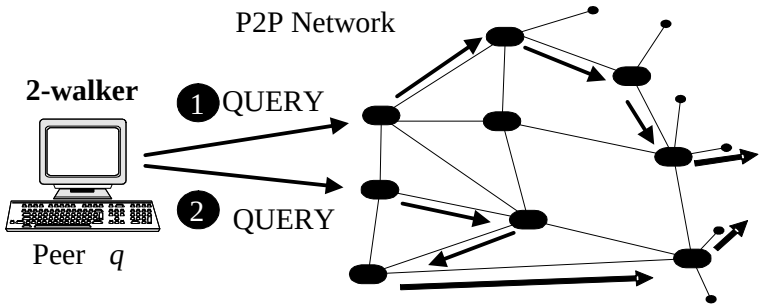

Figure 2.3: Searching using a 2-walker. Each node forwards the query to a random neighbor.

linear. Both RBFS and K-walker do not use any explicit technique to guide the search query to the most relevant content, which is a desirable property in Information Retrieval, making it therefore inappropriate in our context.

2.4 Directed BFS and the Most Results in Past (>RES) Heuristic.

The most closely related technique to our work is presented in [29]. The authors present a technique where each node forwards a query to some of its peers based on some aggregated statistics.

The authors compare a number of query routing heuristics and mention that the *The Most Results in Past (>RES)* heuristic has the best satisfaction performance. A query is defined to be *satisfied* if Z, for some constant Z, or more results are returned. In >RES a peer q forwards a search message to k the peers which returned the most results for the last 10 queries. In their experiments they chose $k = 1$ turning in that way their approach from a *Directed BFS* into a *Depth-First-Search* approach.

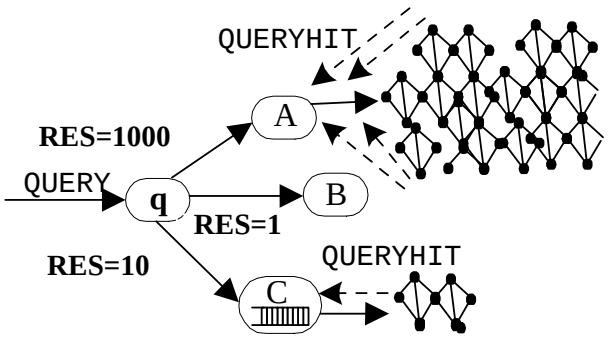

Figure 2.4: The $>$RES heuristic is able to identify stable neighbors, neighbors connected with many others as well as neighbors which are not overloaded. It however fails to explore nodes which contain content related to a query.

The technique is similar to the Intelligent Search Mechanism we propose, but uses simpler information about the peers, and is optimized to find Z documents efficiently (for a fixed Z) rather than finding all documents. Although the authors mention that $>$RES performs well because past performance is a good indicator for future performance in our work we identify some further points that justify the $>$RES's performance.

From the experimental analysis performed in section 6 we conclude that $>$RES performs well because it manages to capture one important problem in P2P systems, namely *network instability*. The $>$RES metric for a connection can be translated as a metric of stability of that particular peer or of the network segment that particular peer connects us to. Moreover $>$RES captures the network segments which are not overloaded since usually those segments return the less results. For example in figure 2.4 node C has a loaded queue and is therefore not able to promptly route and answer queries. Finally $>$RES provides an insight on the number of peers or documents which are

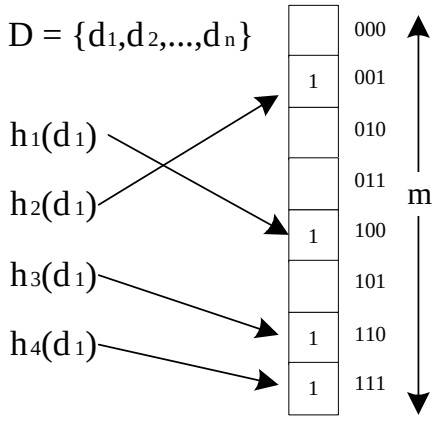

Figure 2.5: A Bloom Filter that uses 4 hash functions and has a size of m=8 bits.

hidden behind a particular node. In the same example we can see that node A may lead us a big segment with potentially many results. Although the >RES has many advantages it doesn't manage to explore the nodes which contain content related to the query. We therefore characterize >RES as a *quantitative* rather than *qualitative* approach.

2.5 Using Randomized Gossiping to Replicate Global State

In *PlanetP* [7] an approach for constructing a content addressable publish/ subscribe service that uses gossiping of global state across unstructured communities is proposed. Their approach uses *Bloom filters* to propagate global state across the community.

A *Bloom filters* [2] is a method for representing a set $D = \{d_1, d_2, ..., d_n\}$ of n elements to support membership queries. More specifically a Bloom Filter is a vector V of m bits which is able to compress the content of D by only using m bits. D can be thought as an index of all the keywords found in the repository of some node N. Intuitively propagating D is an expensive

operation making it therefore inappropriate for large communities. Therefore N uses the vector V and k independent hash functions, $h_1, h_2, ..., h_k$ each with range {1,..,m}, and hashes each keyword (i.e. $\forall d_i$) with the k hash functions. Given V somebody may query the data collection by computing the k hash functions of a particular query term and then checking if all the positions in V are set to 1. If yes then there is a high probability that the query term indeed is part of the D collection. "False positives" can be eliminated drastically, as shown in [9] by choosing the appropriate values for m and k. The reason why *PlanetP* uses Bloom Filters is that the cost of replacing a bloom vector in the global index is constant (i.e. m bits).

In PlanetP each node randomly propagates a *membership directory* and a *compact content index* which are merged to the local structure of each node. Therefore given that each node N maintains an updated list with of (IP, Bloom Filter) pairs a node can perform a local search to derive which nodes have the searching term and then forward the query to only those peers which have potentially some answer. Then each node that receives the query either performs an *exhaustive search* or performs a *selective search* using the vector space rank model. The later uses, similarly to the ISM mechanism, the cosine similarity to measure the similarity between two vectors (i.e. the query and the document). The main distinction is that they are using the cosine similarity in a different context (i.e. for giving the most related answers) while in ISM we use the cosine similarity to route the query to the hosts that have answered the most related queries in the past.

The main advantage of PlanetP, with respect to the *Distributed HashTable* approaches, is that the documents being shared by the nodes are not required to be replicated or moved, making it therefore appropriate for dynamic environments. The main disadvantage though, as with every system that uses global knowledge, is the scalability issue. Although some methods for scaling

Table 2.1: An example of a *Compound Routing Index* at node A. The first row presents the local index of A while the rest rows indicate how many documents are reachable through each neighbor.

@A	Number of Documents	Database Related	Network Related	Theory Related
A	300	20	80	0
B	100	20	0	10
C	1000	0	300	0
D	200	100	0	150

beyond communities of 10000 nodes are proposed in the paper none of them is experimentally evaluated.

2.6 Searching Using Local Routing Indices

In [6] Crespo et al, present a hybrid technique which addresses the issue of building and maintaining local indices which will contain the ”direction” towards the documents. More specifically three different techniques, namely *Compound Routing Indexes (CRI), Hop-Count Routing Index (HRI)* and *Exponentially aggregated RI (ERI)* are presented and evaluated over different topologies like tree, tree with cycles and powerlaw. The ideas deployed in the routing indexes schemes can be thought as the routing tables deployed in the Bellman Ford or Distance Vector Routing Algorithm, which is used in many practical routing protocols like BGP and the original ARPAnet [17]. More specifically a node knows which peers will lead him to the desirable amount of documents but it doesn't know the exact path to the documents. As shown in table 2.1 in CRI a node N maintains for each neighbor some statistics which indicate how many documents are reachable through each neighbor.

The main limitation of CRI is that it does not take into account the number of hops required to reach some documents. Therefore the *Hop-Count Routing Index (HRI)* is proposed which uses a different neighbor goodness model (i.e. the model that defines to which neighbor to forward the query to). and where we maintain a CRI index for k hops (i.e. $\{CRI_i |\ i = [1..k]\}$). Since this approach has a prohibitive storage cost for large values of k the *Exponentially aggregated RI (ERI)* is proposed, and which addresses this issue by aggregating HRI using a cost formula.

Their experimentation reveals that ERI and HRI offer significant improvements over not using any routing index while on the same time they keep the update costs low. Since the Local indices technique is essentially a push update, where each peer sends to its peers information about its documents (along with updates every time a local update happens), thus it is complementary to our approach where the profiles get updated when a peer answers a query.

2.7 Centralized Approaches

In centralized systems there is an inverted index over all the documents in the collection of the participating hosts. These include commercial information retrieval systems such as web search engines (e.g., Google, Yahoo) that are centralized processes, as well as P2P models that provide centralized indexes [23, 28]. Figure 2.6 illustrates the usage of centralized systems such as *Napster* [23]. In the first step node A uploads an index of all its shared documents to the centralized repository R. R then integrates the contents of A in its own index in such a way that searching for a keyword becomes efficient (i.e. an inverted index). Of course the actual index might be partitioned using some hashing scheme, along several machines in order to accommodate larger indexes. In the second step some node B can search the community by sending a query message to R. Now if we

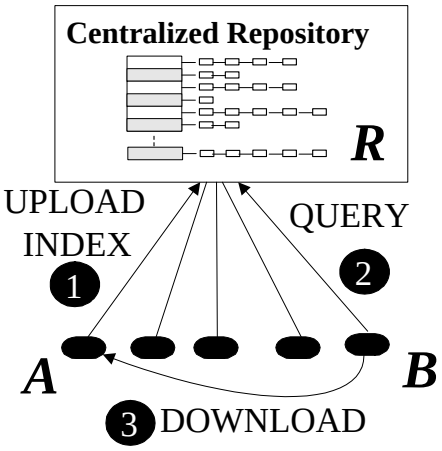

Figure 2.6: In centralized approaches there is usually an inverted index R over all the documents in the collection of the participating hosts .

suppose that A can satisfy B query criterion then R responds to B request with A's address (i.e. IP and port). In the third step node B communicates with A (using an out-of-band protocol such as HTTP) and requests the document that B found through R.

These techniques represent an altogether different philosophy, and they are not directly comparable. In general, one trades simplicity and robustness with improved search time and more expensive resources. Centralized approaches are faster and guarantee to find all results while the decentralized approaches allow always fresh contents and are less costly.

2.8 Depth-First-Search and Freenet

Freenet [4] is a distributed information storage and retrieval system designed to address the concerns of privacy and availability. The system is transparently moving, replicating and deleting files based on usage patterns and its search algorithm does not rely on flooding or centralized

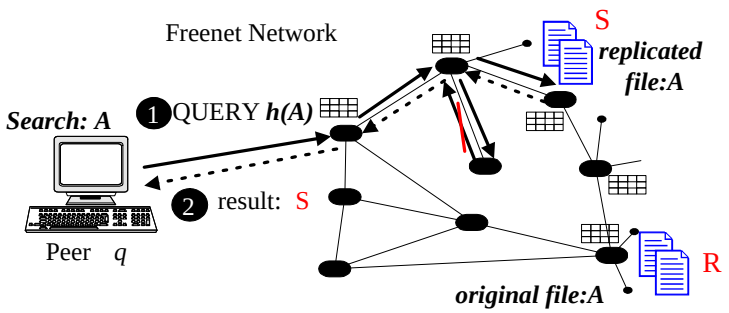

Figure 2.7: Freenet uses an intelligent Depth-First-Search mechanism along with caching of keys/objects at intermediate nodes. The intermediate caching achieves redundancy as well as anonymity.

indexes. The query model in freenet is based on an intelligent *Depth-First-Search (DFS)* mechanism which tries to find a given file. A query in Freenet is identified by a 64-bit *transaction ID* chosen randomly and locally at each peer, In order to bound the number of hops a query travels, Freenet uses the Time-To-Live (TTL) parameter which is widely used in networking applications and protocols.

In their model a user searching for file A first computes the key[2] of A (i.e. $h(A)$) checks its local key table and if it does not find the object it passes $h(A)$ to some intelligently chosen neighbor (see figure 2.7). The neighbor chosen is the neighbor that has the closest key[3] (lexicographic distance between keys).

Therefore $h(A)$ passes recursively through a chain of nodes in which each node makes a local decision about where to send the request next. Therefore their idea relies only on local

[2]Freenet uses a 160-bit SHA-1 [1] hash function

[3]The reason why they use such an approach is that newly inserted files are placed on nodes already possessing files with similar keys, which therefore yields a topology where nodes with similar keys are clustered together.

knowledge rather than any type of centralized or global knowledge. Once the object is found, either from the original publisher R or from somebody who holds a replica of it S, it is sent along the same path the query arrived. In the example of figure 2.7, S augments along with the queryhit message a notice that he is the one who holds the document. Therefore the downloader is not able to know whether he is the original publisher or not. The fact that requests pass through a chain of peers ensures the privacy of the requester and the fact that data is replicated ensures that the original publisher is never known.

Our approach is more general because Freenet allows only searching with file identifiers, instead of the file contents. In addition, we use modified versions of the Breadth-First-Search approach, where many messages are propagated in the network concurrently, rather than a Depth-First-Search approach, where each node sends a message to one peer and waits for a reply before forwarding it to another peer. The advantage of DFS search is that a small set of peers can be queried quickly and efficiently; however by its nature it can take a long time if we want to find all the results to a query, that happen to be distributed in many peers. Another drawback is that initially Freenet might perform in the worst case as bad as the flooding algorithm but it is expected to improve over time as a node develops more knowledge. Moreover Peer-to-Peer environments tend to be unstable and connections in such environments might be in the order of minutes, since nodes are joining and leaving. This inherently means that query messages may be trapped in the network because the reverse path of a query message was lost due to a broken connection.

2.9 Consistent Hashing and Chord

In this section we introduce *Chord* [27] which is a distributed lookup protocol that uses a consistent hashing scheme. Chord like other *Distributed HashTable (DHT)* algorithms allows peer-to-peer applications to efficiently locate a node that stores a particular object. In Chord one basic operation, *lookup(key)*, returns the address (i.e. IP) of the node storing the object with that key. This operation allows nodes to *put* and *get* files in the community only based on their key.

In the proposed technique an m-bit *identifier* is used to hash both *Nodes*[4] and *Objects*[5]. Although Chord is not restricted to any particular hash function the scheme deploys SHA1 which is widely used and in which collision of two keys is difficult. The next operation is to assign each *Node N* and *Key k* in a one- dimensional circular key space which has $[0..2^m - 1]$ many slots (figure 2.8). The key k is assigned its *successor*, which is the first node N that is equal or follows the value of k. In order to make lookups more efficient a node maintains the *finger table* which contains a number of successors. Therefore in the steady state a node is maintaining information about only $O(logN)$ other nodes, where N is the size of the network, and since data items are stored in specific locations object lookups involve $O(logN)$ messages. Every time a node joins or leaves the system the protocol does not require more than $O(log^2N)$ messages for updating the finger tables of the rest nodes.

In the example of figure 2.8, we are searching for `file1` from node A. From A's finger table the algorithm decides to continue the lookup through node C, which's key immediately precedes the one of `file1` (i.e. $5 \leq 7$). The same happens at nodes C which continues the lookup

[4]The hashing is based on the IP of the node

[5]The hashing is based on the object itself and the result is refereed as *key*

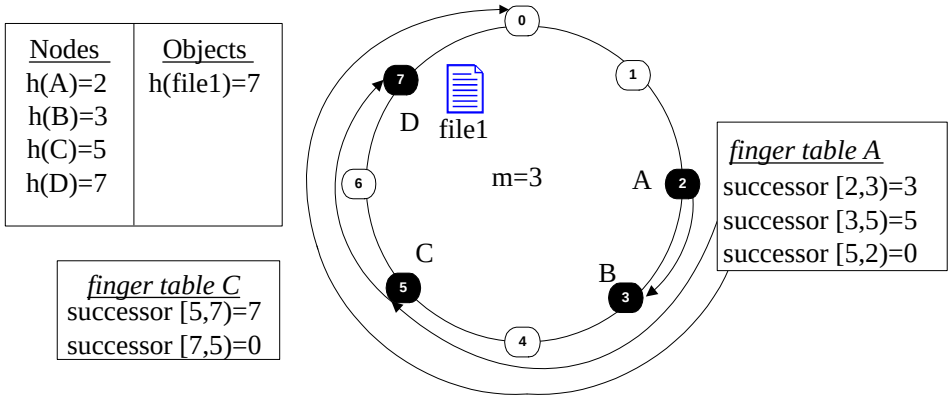

Figure 2.8: Chord uses a consistent hashing scheme to organize objects and nodes in a virtual circle. The proposed algorithm provides an efficient way for looking up objects.

through node D at which point the lookup completes successfully. The use of Consistent Hashing scheme has two major advantages: (i) Little data movement in the case nodes join or leave the network[6] and (ii) there is good load balancing since each node has approximately the same amount of keys

The main advantage of Chord over Freenet or other approaches presented in this chapter is that we can have efficient and predictable object lookup. Although Chord is appropriate for applications like distributed file systems, application layer multicast it is not suitable in the context of Information Retrieval because in the later we are searching the contents of the shared documents rather than the objects. Furthermore DHT algorithms present some drawbacks in environments were nodes join/leave at high paces since the finger tables won't be in steady state which can lead to wrong routings. Furthermore data moving may take considerably long time if objects are large.

[6] In the event of the Nth node join only O(1/N) keys/data are moved around

Chapter 3

The Intelligent Search Mechanism (ISM)

In this section we present the Intelligent Search Mechanism (ISM) which is a new mechanism for information retrieval in the P2P networks. The objective of our algorithm is to help the querying peer to find the most relevant answers to its query quickly and efficiently rather than finding the larger number of answers.

Keys to improving the speed and efficiency of the information retrieval mechanism is to minimize the communication costs, that is, the number of messages sent between the peers, and to minimize the number of peers that are queried for each search request. To achieve this, a peer estimates for each query, which of its peers are more likely to reply to this query, and propagates the query message to those peers only.

3.1 Design Issues of the ISM mechanism.

The design objectives of the Intelligent Search Mechanism were the following:

1. **Maintain Only Local Knowledge**. Approaches that maintain global state tend to have a significant communication overhead which makes them inefficient for dynamic environments were the participants are not known a' priori. ISM uses only local state information of a constant size which therefore minimizes the communication overhead.

2. **Avoid Data Replication**. Distributed HashTable approaches usually distribute resources along with the keys to nodes in a network. That approach has a significant overhead for nodes that join the network only for a short period of time. Consider for a example a node n that joins the network for a few minutes. This node is assigned by the global hashing scheme some k files which will be transferred to n. The size of the k files might be in the order of several Megabytes yielding therefore a significant communication cost. Since n decides to leave the network after a few minutes the network did not gain anything by replicating the k files. ISM on the other hand assumes that no replication takes place in a network of nodes. This is also a reasonable assumption from the social point of view (i.e. "Why would some node share resources used by somebody else?"), although we don't claim that DHTs are appropriate for file sharing applications.

3. **Reduce Messaging**. Although brute force techniques such as Breadth-First-Search (section 2.1) are simple since they don't require any form of global knowledge, they are expensive in terms of messages. ISM addresses this issue by intelligently forwarding query messages to nodes that have a high probability of answering the particular queries.

4. **Route Queries to Relevant Content**. Although approaches such as Random Breadth-First-Search (section 2.2) or the Random Walkers (section 2.3) significantly reduce messaging they

do not address the issue of finding the most relevant content. ISM on the other hand uses the $RelevanceRank$ of a peer to forward a query to the peers that have the highest potentiality of answering the particular query.

3.2 Components of the ISM mechanism.

The Intelligent Search mechanism for distributed information retrieval consists of four components:

1. A *Search Mechanism* to send the query to the peers. This is the only mechanism used by a node to communicate with its peers. It is the same mechanism employed by the Gnutella protocol for communications between peers.

2. A *Profile Mechanism*, that a peer uses to keep a profile for each of its neighbors. The profile keeps the most recent past replies coming from each neighbor.

3. *RelevanceRank*, which is a peer ranking mechanism that a peer runs locally using the profiles of its peers and the specific query. The mechanism ranks the peers in $N(u)$ in order to send the search query to the most likely peers.

4. A *Cosine Similarity* function that a peer uses locally to find the similarity between different search queries.

3.3 The Search Mechanism

Assume that a peer initiates a search to find documents about a specific topic. Since it is initiating the search, we call him the *querying_peer*. The *querying_peer* generates a `Query` message that describes his request, finds which of his peers are most likely to provide an answer (using the profile mechanism and the peer ranking mechanism) and broadcasts the `Query` message to those peers only.

If a peer receives a `query` message we call him the *receiver_peer*. If the *receiver_peer* can provide an answer, it returns an answer to the requesting *querying_peer*. It also propagates the `Query` message only to those of his peers it considers most likely to provide the answer (Figure 3.1). To provide a termination condition so that the messages are not propagated indefinitely in the network, the *querying_peer* sets a bound on the depth of the recursion. When a reply QueryHit message is sent back to the *querying_peer*, the peers in the answer path (which is the same as the query path) record the query and the name of the peer that provided the answer in a $(query, peer)$ table, illustrated in Table 3.1. Each peer sets a bound on the number of pairs to be recorded, and uses a least recently used strategy to allow space for new queries.

3.4 Peer Profiles

To decide to which peers a query will be sent, a node ranks all its peers with respect to the given query. The number of peers that a query will be sent is a parameter that is defined by the user. To rank its peers, each node maintains a profile for each of its peers. The profile contains the list of the most recent past queries, which peers provided an answer for a particular query as well as the

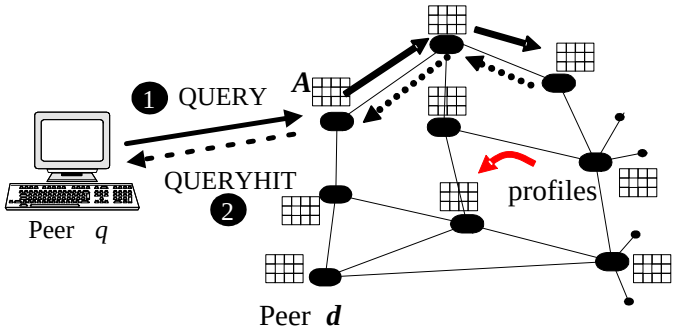

Figure 3.1: Searching in a peer-to-peer network with the Intelligent Search Mechanism ISM: Each peer uses the knowledge it obtains from monitoring the past queries to propagate the query messages only to a subset of the peers.

number of results that a particular peer returned. Although logically we consider each profile to be a distinct list of queries, in the implementation we use a single $Queries$ table (see table 3.1) which records the described information.

The node accumulates the list of past queries by continuously monitoring and recording the `Query` and the corresponding `QueryHit` messages it receives. The node keeps the list of queries in its local repository. For each node this list is incomplete, because each node can only record information about those queries that were routed through it. The node uses a size limit T that limits the number of queries in each profile. Once the repository is full, the node uses a Least Recently Used (LRU) policy to keep the most recent queries in the repository. Since the node keeps profiles for its (d) neighbors only, the total size of the repository is $O(Td)$.

Table 3.1: The Peer's Profile Mechanism snapshot. It shows from which neighbors (i.e. {P1,P2...}) each queryhit came from and on which time (timestamp).

Query Keywords	**GUID**	**Connections & Hits**	**Timestamp**
Columbia NASA Nevada	G568FS	(P_1,50),(P_4,80),...,(P_5,10)	10000000
Elections Usa Bush	OF34QA	*NULL*	10001000
...	...	....	...
Superbowl San Diego	LQI65D	(P_2,20), (P_3,30)	10012300

3.5 Peer Ranking

For each query received by a node P_l, P_l uses the profiles of its peers to find which ones are more likely to have documents that are relevant to the query. To compute the ranking, P_l compares the query to previously seen queries and finds the most similar ones. To find the similarity between the queries, it uses the Nearest Neighbor classification mechanism. The reason that we employ this technique is that it is simple, and it has shown to have good accuracy in many different settings.

Since it is likely that some peers will be associated with many similar queries and others with some, we compute the ***RelevanceRank (RR)***, which is the aggregate weighted similarity of a peer to a given query. Given the K most similar queries to q, peer P_l computes the RelevanceRank $RR_{P_l}(P_i, q)$, of peer P_i to query q as follows:

$$RelevanceRank_{P_l}(P_i, q) = \sum_{\forall q_j answered by P_i} Qsim(q_j, q)^{\alpha} * S(P_i, q_j)$$

where, q_j is one of the K most similar queries to q. This parameter limits the influence to the similarity to the most similar queries only. In addition $S(P_i, q_j)$ is the number of results returned

by P_i for query q_j. This allows us to rank higher the peers that returned more results. Finally, we use the parameter α, which allows us to add more weight to the most similar queries. For example, when α is very large, RR reduces to one-nearest neighbor. For $\alpha = 0$, RR reduces to K-nearest neighbor. If $\alpha = 1$, RR adds up the similarities of all queries that have been answered by the peer. P_l then sends the query to the m peers (for a user defined constant $m < d$) that have the higher *RelevanceRank*.

Consider the following example where we assume that $K = 5$, $\alpha = 1$ and that $\forall i, j$ $S(P_i, q_j) = 2$. Peer P_l wants to send a query q to two of its peers. Let q_1, q_2, q_3, q_4, q_5 be the most similar queries to q, among the ones P_l has information about, with $Qsim(q, q_1) = 0.8$, $Qsim(q, q_2) = 0.6$, $Qsim(q, q_3) = 0.5$, $Qsim(q, q_4) = 0.4$, and $Qsim(q, q_5) = 0.4$. If peer P_1 answered q_1, peer P_2 answered queries q_2 and q_3, and peer P_3 answered queries q_4 and q_5, then we compute the aggregate similarities of the three peers to the query q as follows: $RR_{P_l}(P_1, q) = 0.8^1 * 2 = 1.6$, $RR_{P_l}(P_2, q) = (0.6^1 + 0.5^1) * 2 = 2.2$, and $RR_{P_l}(P_3, q) = (0.4^1 + 0.3^1) * 2 = 1.4$. Therefore P_l chooses to send the query only to peers P_1 and P_2.

The Profile Mechanism of a host P_l is shown in table 3.1. As we can see, each query q_j that was routed through P_l is logged along with the peers $\{P_1, P_2, ..., P_d\}$ from where a queryhit came from. The $S(P_i, q_j)$ pair which shows the number of results that came from P_i for q_j can be found in the *Connections & Hits* column. The ranking mechanism performance is bounded by the number of entries and therefore yields good performance when the table has not an excessive amount of entries.

3.6 Distance Function: The Cosine Similarity

In order to find the most likely peers to answer a given query we need a function $Qsim$: $Q^2 \rightarrow [0,1]$ (where Q is the query space), to compute the similarity between different queries. Since the queries are sets of keywords, we can use a number of different techniques that have been used effectively in information retrieval. We make the assumption that a peer that has a document that is relevant to a given query is likely to have documents that are relevant to similar queries. This is a reasonable assumption if each peer concentrates on a set of topics.

The *cosine similarity* (formula 3.1) metric between 2 vectors ($\vec{q}$ and $\vec{q_i}$) has been used extensively in information retrieval, and we use this distance function in our setting. Let L be the set of all words that have appeared in queries. We define an $|L|$-dimensional space where each query is a vector. For example, if the set L is the words $\{A, B, C, D\}$ and we have a query A, B, then the vector that corresponds to this query is (1,1,0,0). Similarly, the vector that corresponds to query B, C is (0,1,1,0). In the cosine similarity model, the similarity sim of the two queries is simply the cosine of the angle between the two vectors.

$$sim(q, q_i) = cos(q, q_i) = \frac{\sum(\vec{q} * \vec{q_i})}{\sqrt{\sum(\vec{q})^2} * \sqrt{\sum(\vec{q_i})^2}} \tag{3.1}$$

3.7 Random Perturbation

One problem of the technique we outline above is that it is possible for search messages to get locked into a cycle. The problem is that the search will fail then to explore other parts of the peer-to-peer network and may not discover many results.

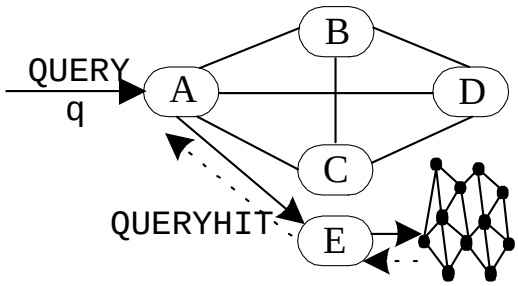

Figure 3.2: With Random Perturbation we give node A the opportunity to break the cycle (A,B,C,D) in which queries may get locked and therefore allow it to explore a larger part of the network and find the correct answers.

Consider for example figure 3.2 and the following scenario: Peer A receives a query q which has no answer from any of the displayed nodes (i.e. A,B,C or D). Further each node answers to the conjunction of the terms found in q. Suppose that A chooses to forward q to B,C and D because these nodes have successfully answered to a similar query in the past. Therefore A doesn't choose node E which this time would lead him to the correct results. Query q gets consequently locked in a cycle (i.e. A,B,C,D) and fails to explore the segments of the network which contain the correct answer.

To solve this problem, we pick a small random subset of peers[1] and add it to the set of best peers for each query. As a result, even if the best peers form a cycle, with high probability the mechanism will explore a larger part of the network and will learn about the contents of additional peers.

[1] In our experiments we additionally select 1 random peer.

Chapter 4

Performance Analysis of the Proposed Techniques

In this section we describe the characteristics of the proposed techniques, in comparison with the Gnutella protocol which is a BFS Algorithm with some TTL (Time-to-Live) parameter that limits the depth that a query travels. We concentrate on the *recall* rate, that is, the fraction of documents our search mechanism retrieves compared to the other mechanisms, and the *efficiency* of the technique, that is, the ratio of number of messages that the different techniques use for the same search.

4.1 Performance of the BFS Algorithm

Assume a graph G of n nodes and e edges where each node n_i has a degree of d_i. If each unique query q is forwarded exactly once by each node n_i, then q will be forwarded a total number of

$\sum_{i=1}^{n}(d_i - 1)$ times.

The reason for this is that when a given node n_i receives a query q from some query peer Q, n_i sends $d - 1$ messages to its neighbors (i.e. except the sender), Since each n_i forwards q exactly once, q is forwarded a total number of $\sum_{i=1}^{n}(d_i - 1)$ times.

4.2 Performance of the Random BFS Algorithm

We first consider the performance of the modified random BFS technique where each peer selects a random subset of its peers to propagate a request (that is, here a profile of its peers is not used). In a P2P network with a random graph topology, this mechanism searches the nodes of the graph more efficiently (that is, it sends fewer messages) than the BFS algorithm.

Consider a random graph G with n nodes and e edges, that has average degree d. For a given node u, let $N_k(u)$ be the set of nodes at distance at most k from u. When a node u starts a Gnutella search with a TTL $= k$ (Time To Live, as per the Gnutella search protocol), u sends approximately d messages to its neighbors, each being propagated k times. Since the BFS mechanism explores all the edges in the graph, the number of messages send by the Gnutella protocol is at least $|N_k(u)|\frac{|N_k(u)|}{n}d$.

Assume on the other hand that each node only propagates the message to a randomly chosen subset of its neighbors, of size $\frac{d}{m}$ (for a suitably chosen m). Using the same TTL (k), if $|N_k(u)|$ is smaller than $n/2$, the expected total number of messages sent is $(\frac{d}{m})^k$, and the expected number of vertices that this modified BFS process visits is at least $\frac{1}{2}(\frac{d}{m})^k$. This is because if $|N_k(u)|$ is smaller than $n/2$, then most of the nodes visited in each iteration are new nodes. Consider a node

v of distance i ($i < k$) from u. If $|N_k(u)| < n/2$, with high probability each edge of v is connected to a node not in $N_i(u)$. Setting $\frac{1}{2}(\frac{d}{m})^k = |N_k(u)|$, we have that, if $|N_k(u)| \approx n/2$, the modified BFS needs at most a fraction of $\frac{4}{d}$ of the number of messages used by the Gnutella protocol to visit approximately the same number of vertices.

4.3 Performance of the Intelligent Search Mechanism

The previous discussion indicates that propagating a query to a random subset of one's peers is more efficient when searching nodes in a P2P network with random graph topology than using the Gnutella protocol (with respect to the total number of messages). However this approach is approximate, and cannot guarantee that all nodes in $N_k(u)$ are found. Consider for example a case where two large sub-graphs are connected by one edge. If the node attached to that edge does not choose this edge, the other sub-graph will never be explored.

The Intelligent Search technique we outlined in the previous section attempts to identify edges that are likely to have good information. Nevertheless, the accuracy of the mechanism clearly depends on how accurately a peer can compute which of its peers is likely to answer a given query. Work on distributed information retrieval has shown that current techniques for database selection can give good performance. Experiments show that requesting a random set of documents from a collection is sufficient to obtain accurate estimates on the word frequencies in this collection. These results are directly applicable only for the case that each peer has full statistical information for its peers. Our setting is different because the information we collect is incomplete; these are only the queries that peers reply to, rather than all the documents in the actual replies. This is certainly very useful when very similar queries repeat.

We also note that the more efficient search allows us to use a larger TTL compared with the Gnutella protocol, while still having a smaller number of messages overall. As a result, this mechanism can visit nodes that the Gnutella protocol would not visit without sending a large number of messages. We explore this trade-off in the experimental evaluation of the technique.

In summary, the Intelligent Search mechanism for distributed information retrieval that we propose has the following characteristics:

1. The algorithm uses fewer messages compared to the standard Gnutella strategy, and scales better with respect to the size of the network (because it can search a larger network using the same number of messages)

2. The size of the profiles is proportional to the number of direct connections per peer. This is likely to remain small (constant) even for very large networks.

3. The scheme uses the combined knowledge about the peers, and adapts and modifies its behavior as each peer learns more information about its peers. On the other hand, peers do not have to export any information about their databases.

Chapter 5

PeerWare Simulation Infrastructure

In order to benchmark the applicability and efficiency of our algorithms we have implemented *PeerWare*, a distributed middleware infrastructure which allows us to benchmark different routing algorithms over large-scale P2P systems. Probing different query-routing algorithms over middleware P2P systems can be interesting from many points of views:

1. In real settings the scalability of various query-routing algorithms may be explored to the fullest extend since there are no assumptions which are typical in simulation environments.

2. Moreover many properties, such as network failures, dropped queries due to overloaded peers and others may reveal many interesting patterns.

3. Finally, in a middleware infrastructure we are also able to capture the actual time to satisfy queryhits.

Unfortunately most simulators fail to capture these properties and their results are therefore inadequate. Other approaches such as [29] tend to build simulation environments based on statistics obtained from real P2P networks. We mention that our middleware infrastructure can be adjusted to many different parameters making it appropriate for many different settings.

5.1 Simulating Peer-to-Peer Systems

The Anthill Project [3] developed at the University of Bologna uses Jtella [21] Java API as a basis for building a fully customizable API for the Gnutella network. The aim of the project is to create a simulation framework which will allow researchers to develop and validate new P2P algorithms. The system itself is inspired from the biological metaphor of Ant colonies. They mention that real Ants are known to locate the shortest path to a food source using as only information the trails of chemical substances called *pheromones* deposited by other ants. Moreover they mention that although individual ants are unintelligent and have no explicit problem solving capability, ant colonies manage to perform several complicated tasks with high degrees of consistency. Although the project doesn't emphasize particularly on P2P case studies, it is worth it to mention that they are currently using their framework to investigate the properties of the Freenet [4] algorithm by modifying its protocol and comparing the performances of different implementations.

Their framework intends to obtain:

1. Information about the queries performed by users and their distribution. More specifically

they aim to find popular queries or keywords that may be exploited to implement intelligent caching algorithms.

2. Information about the files stored in the Gnutella network, which might be obtained by logging the Gnutella QUERYHIT messages.

3. Information about the shape of the network, which might be obtained by actively probing Gnutella PING and PONG messages. They also intend to take advantage of the Gnutella PUSH messages in order to partially investigate which files are downloaded by users.

Although Anthill uses the notion of *scenarios*, which is composed of a collection of interconnected nodes and a scheduling of requests to be performed, there is no documentation on that.

5.2 Modeling Large-Scale Peer-to-Peer Networks

Jovanovic et al. study [15], of Modeling Large-scale Peer-to-Peer Networks, is to our knowledge the only comprehensive work done in the area of modeling Peer-to-Peer systems.

Their study reveals that Gnutella has some important structural properties, such as *small-world* properties and several power-law distributions of certain graph metrics. They mention the famous Milgram's experiment [22] which was conducted in the early 1960's, and in which a number of letters, addressing a person in the Boston area, were posted to a randomly selected group of people in Nebraska. Each person who received the letter forwarded it to someone that they knew, on a first name basis. As many of the letters finally reached the designated person, the average number of hops observed was between five and six. Their study reveals that a similar, small world property

existed in the Gnutella Network. More specifically in 5 different snapshots of the Gnutella Network they found that the diameter of the network ranged from 8-12.

In their work they have also discovered that the Gnutella Network obeys all four of the power-laws described in Faloutsos et al. work. [8]. More specifically they found, on a Gnutella snapshot gathered on the 28^{th} of December 2000, that the Rank Exponent R (Power-Law 1) holds with $R = -0.98$ and a correlation coefficient of $|r| = 0.94$. It is important to mention that similar results which were obtained one month earlier by an independent group at U. of Chicago [26]. The same group claims that this power law faded-out in repeated experiments in the March-June 2001 period.

Jovanovic's study on the same snapshot of data also revealed that the Outdegree Exponent O (Power-Law 2) also holds with $O = -1.4$, although this comes in disagreement with the $O = -2.3$ exponent found in the 6 month earlier study of the DSS [5] group.

Their study finally shows that the Hop-Plot Exponent H (Power-Law 3) and Eigen Exponent E (Power-Law 4) hold for four different snapshots with very high coefficients of $|h| = 0.99$, $H = 3.5$ and $|i| = 0.94$, $E = 2.83$ respectively.

The general belief is that earlier versions of the Gnutella Network were Power-Law but as the network has grown this property doesn't hold any more. One important fact is that as the Gnutella network became more mature, more intelligent clients were added to the network. Intelligent clients can affect dramatically the way a Network Crawler operates. The latter relies on the fact that clients will respond to its requests but in the case the clients do not comply with this requirement, the Network crawler will generate inaccurate data.

Table 5.1: A sample XML record from a dataPeer's XML repository.

```
< REUTERS ID = "413" >
   <DATE>2-MAR-1987 09:36:46.03< /DATE>
   <PLACE>NETHERLANDS< /PLACE>
   <TEXT>
      <TITLE>Netherlands 47 mln...< /TITLE>
      <BODY>This raised the amount of...< /BODY>
   < /TEXT>
< /REUTERS>
```

They are also presenting interesting visualizations of their gathered data, which were visualized with LEDA [18]. The main disadvantage of their study is that their experiments were performed on a small set of peers (1000), which is not representative of the today's picture of the network. Additionally, their Gnutella Crawler Implementation is in some sense static since it starts from a pre-specified seed file of peers and relies on the fact that it will discover new nodes on runtime.

In the rest of this chapter we will provide a technical description of the four components which comprise the *PeerWare* simulation testbed. These components are *dataGen* the dataset generator, *graphGen* the network topology generator, *dataPeer* a hybrid xml-p2p client which answers to queries from its local xml repository and *searchPeer* a P2P client which performs the queries in the dataPeer network.

5.3 dataGen - The Dataset Generator

dataGen is a Reuter's [25] dataset transformer which takes as input the Reuter's set of documents and transforms it into a collection of XML documents by some of the following criterions, which are found in the collection:

Table 5.2: PDOM-XQL and Retrieving an XQL ResultSet of all articles of a given country.

```
static final String query = "//REUTERS//TEXT//BODY";
public XQLResult getAllRecords() {
  XQLResult r = XQL.execute(query, doc);
  System.out.println("Row(s) found : " + r.getItem(0) + " " + r.getLength());
  return r;
}
```

$\{$*Date, Topics, Places, People, Orgs, Companies*$\}$.

For our experiments we have chosen the *Places* criterion which clusters the documents by country. There were 104 different countries with at least 5 documents, and the total number of documents for these 104 countries was $22,769$ (some documents belong to more than one country). We created a network of 104 peers. The topology was a random graph with average degree 8 (random graphs with more than $\log n$ average degree are almost certainly connected). Each peer was assigned data from exactly one country.

The objective of the Dataset Generator was to generate sets of documents about specific topics in order to represent the specialized knowledge of each peer. The dataset generator is implemented in Java and uses IBM's XML Parser [14] along with the GMD-IPSI XQL engine [10], which is a Java based storage and query application for large XML documents.

The GMD-IPSI XQL engine offers the Persistent Document Object Model (PDOM) class which implements the full W3C-DOM API on indexed, binary XML files. Therefore documents are parsed only once and stored in binary form for future usage. The cache architecture of the

Table 5.3: The `country-hosts.graph` file for ”australia” shows the outgoing connections that will be established during initialization.

```
#UCR Random Graph generator

country=australia
ip=283-20.cs.ucr.edu
port=10002

#Peers that ”australia” should connect to
china=283-20.cs.ucr.edu,10013
india=283-25.cs.ucr.edu,10008
vietnam=283-28.cs.ucr.edu,10016
lebanon=283-25.cs.ucr.edu,10021
mexico=283-26.cs.ucr.edu,10000
spain=283-26.cs.ucr.edu,10024
chile=283-20.cs.ucr.edu,10012
```

engine makes the engine a Memory-Disk structure highly appropriate for large XML documents which can’t physically fit in main memory. PDOM has increased performance as compared to the Document Object Model (PDOM) implemented by most of the XML Parsers and which tries to build an in-memory data structure of the XML documents and which usually has a great performance penalty. The XQL processor is used to query PDOM files by providing both the PDOM Document and the query to be processed (Table 5.2).

5.4 graphGen - Network Graph Generator

graphGen, is the component responsible for generating the simulation topology. More specifically, it generates a set of configuration files which can be read by the various nodes that comprise the simulation network topology. *graphGen* starts out by reading `graph.conf`, which contains among others the following parameters:

Table 5.4: Distribution of Gnutella IP Addresses to Domains.

#	Country	Domain	IP Addresses	Total Percentage%
1	Network	.net	$94,456$	$38,88\%$
2	US Commercial	.com	$81,943$	$33,73\%$
3	Canada	.ca	$8,039$	$3,31\%$
4	France	.fra	$5,565$	$2,29\%$
5	US Educational	.edu	$5,102$	$2,10\%$
6	England	.uk	$4,118$	$1,69\%$
7	Germany	.de	$3,693$	$1,52\%$
8	Australia	.au	$3,663$	$1,51\%$
9	Austria	.at	$2,962$	$1,22\%$
10	Netherlands	.nl	$2,625$	$1,08\%$

1. Outdegree of a node, which is used in the case a random graph.

2. Topology of the P2P network (e.g. random graph).

3. IP List of hosts that will participate in a simulation. This allows us to map a logical topology (e.g. Node1 -> Node10) to many different IP topologies

The output of graphgen is a directory of several `country-hosts.graph` files (see table 5.3). Each file contains the IP and port address of hosts to which a particular country must connect to. We are currently working on incorporating various other topology alternatives, such as tree, tree-with added cycles, or power-law.

On table 5.4 we can see the distribution of domains from a set of 300,000 IP addresses found in the Gnutella Network. These results, which we gathered in [30], will be used in our future work, to build a more realistic network topology that will again facilitate the evaluation of our query routing techniques.

These alternatives can easily be embodied in our system since it requires only changes on

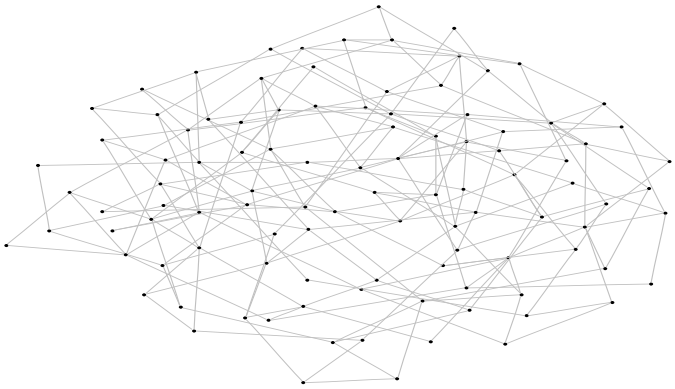

Figure 5.1: Visualization of a random graph with 104 peers & degree=4, with graphGen.

graphGen rather than the whole system. grahGen, also generates output which can be piped into Visualization Tools, such as GraphViz [24], and generate graphical representations (i.e. directed or undirected graphs) of the network topology.

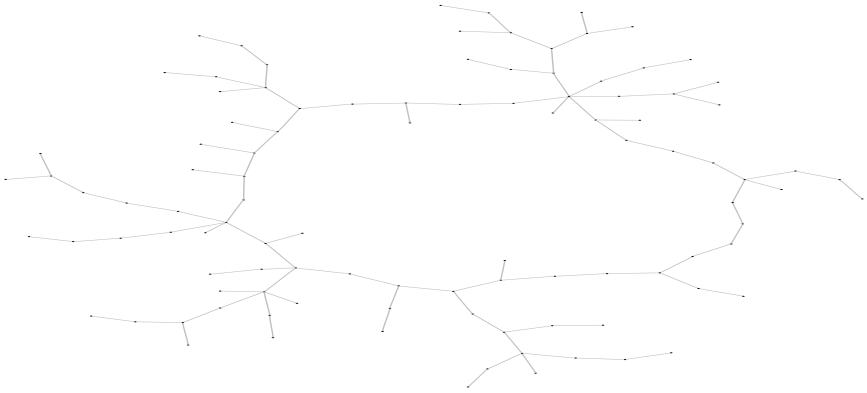

Figure 5.2: Visualization of a random graph with 104 peers & degree=2, with graphGen. The graph is connected which is not typical for graphs with a small degree.

Figures 5.1 and 5.2, present a random graph generated with graphGen using Graphviz's dot 2D undirected graph layout. It is important to mention that generating visualizations for huge graphs can take a considerably large amount of time and may finally not provide the adequate help in under-

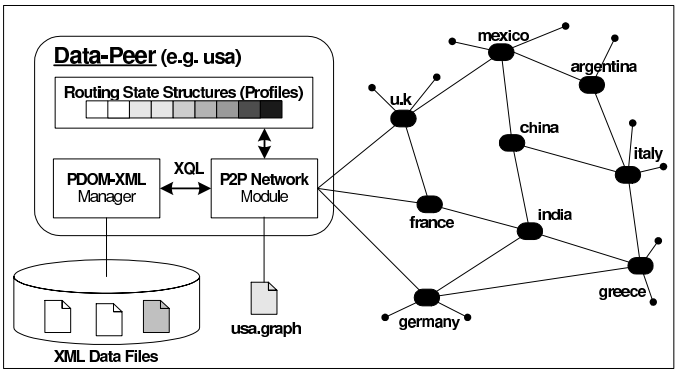

Figure 5.3: The Components of a *dataPeer* Brick.

standing how the network looks like. Visualizing P2P network graphs is described in some extend in [15]. They try to visualize the Gnutella backbone (i.e. interconnected nodes with $degree > 10$) rather than the whole network in order to obtain some more understandable results.

5.5 dataPeer - The Data Node

dataPeer, is a P2P client which maintains a local repository of XML documents. Typical P2P clients answer to queries only based on filename descriptions. *dataPeer* on the other hand queries its repository each time a query arrives.

dataPeer consists of three sub-components : 1) The *PDOM-XML Manager*, which queries efficiently the local xml repository with XQL, 2) A *P2P Network module*, which provides an interface to the rest of the network and which implements the various query-routing algorithms, and 3) *Routing State Structures*, which capture the implementation specific logic of the proposed query-routing algorithms.

Each dataPeer d reads upon initialization a `country-hosts.txt` file which contains

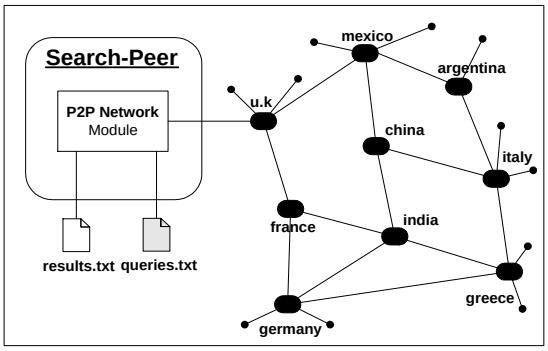

Figure 5.4: The Components of *searchPeer*.

the IP and $ports$ of other dataPeer's to which d must connect. Each dataPeer tries continuously to establish and maintain its outgoing connections. Therefore we are not required to incorporate any topological sort algorithm. Connections among dataPeers are achieved by the use of TCP Sockets and are persistent (they remain open until d shuts down). If a TCP connection goes down because of an overloaded peer then a node automatically re-establishes the connection after some small interval.

Obviously launching a large number of dataPeers on many different machines is a tedious procedure. We have therefore constructed a set of UNIX shell scripts which automatically (by the use of *ssh* and public/private keys) connect to any number of machines and launch the dataPeers. Bringing up a PeerWare Network of 104 dataPeers, on 20 machines, including latencies such as xml repository manager initialization and others takes about a minute.

5.6 searchPeer - The Search-Node

searchPeer (Figure 5.4), is a P2P client which submits a number of queries in a PeerWare network and harvests the returned results. In contrast with *dataPeer*, searchPeer consists only of a

Table 5.5: Sample set of unstructured queries posted by searchPeer. Each keyword consists of at least 3 characters and the keywords are sampled from the dataset.

#	Query
1	AUSTRIA INTERVENE DOES DOLLAR
2	APPROVES MEDITERRANEAN FINANCIAL PACKAGES
3	ABANDONS DEFEATS STRONGHOLD AFTER
4	AGREES PEACE NEW MOVES
5	AID KENYA DEBT MOI
6	AND CALLS FORCE NATO
7	BUDGET BIG RULING JAPAN
8	BANGLADESH PROPOSALS TAX GOVERNMENT
9	BANGLADESH FOR GRANT BRIDGE
10	BANS ZIMBABWE FOR OILSEEDS

Network Module and a Result Logging Mechanism. Besides logging the number of results it also gathers a number of other statistics such as the number of nodes answered to a particular query and the time to receive the results.

searchPeer reads upon initialization the `keywords.txt` file(see table 5.5), which contains a set of unstructured queries sampled randomly from the dataset. These queries are submitted sequentially with a small sleep interval between consecutive queries. The reason for that choice was twofold; firstly it would allow each host to have enough time to process a query request (e.g. a query message wouldn't stuck in some queue and suffer from long queuing delays) and secondly it would make sure that the query response time would have some meaningful value and that it would not be a function of how much traffic is currently in the network. The searchPeer maintains a different session for each query it sent out. In that way it is able to listen to queryhits that were delayed and which arrive after some new query was already send out.

5.7 Implementation

The PeerWare infrastructure is implemented entirely in Java. Its implementation consists of approximately 10000 lines of code, 6200 of which correspond to the `ucr.core.*` which contains code related to the P2P Network Module as well as the different query routing algorithms, 1000 to `ucr.graphgen.*` which contains the graph generator, 1300 to `ucr.nodes.*` which contains the implementation of the dataPeer and searchPeer, 450 to `ucr.datagen.*` which includes the code of the dataset generator and the rest 1050 to common I/O libraries. `ucr.core.*` initial codebase was based on the Jtella [21] which is an API for the Gnutella protocol. A shorten version of the *dataPeer* implementation is shown in the Appendix.

Java was chosen for a variety of reasons. Its object-oriented design enhances the software development process, supports rapid prototyping and enables the re-use and easy integration of existing components. Java class libraries provide support for key features of PeerWare: platform independence, multithreading, network programming, high-level programming of distributed applications, string processing, code mobility, compression, etc. Other Java features, such as automatic garbage collection, persistence and exception handling, are crucial in making our system more tolerant to run-time faults.

The choice of Java, however, comes with a certain risk-factor that arises from known performance problems of this programming language and its run-time environment. Notably, performance and robustness are issues of critical importance for a distributed system like PeerWare, which is expected to run on several machines and to sustain high-loads at short periods of time. In our experiments, we found the performance of Java SDK 1.3 satisfactory.

Chapter 6

Experiments

In order to compare our intelligent search mechanism with the other Query-routing algorithms described in section 2, we have built a decentralized newspaper network. The newspaper is organized as a network of dataPeers; each dataPeer maintains a set of articles related to a particular country. In that way each dataPeer shares some specialized knowledge (i.e. documents related to the country). dataPeers are connected among them using a pre-specified random topology. We then use a searchPeer to connect to a single point in the network and perform a number of queries.

Our experiments were deployed with 104 dataPeers running on a network of 20-50 workstations, each of which has an AMD Athlon4 1.4 GHz processor with 1GB RAM running Mandrake Linux 8.0 (kernel 2.4.3-20) all interconnected with a 10/100 LAN connection.

Our evaluation metrics were: (i) the *recall* rate, that is, the fraction of documents our search mechanism retrieves compared to the other mechanisms, and (ii) the *efficiency* of the technique, that is, the ratio of number of messages used to find the results.

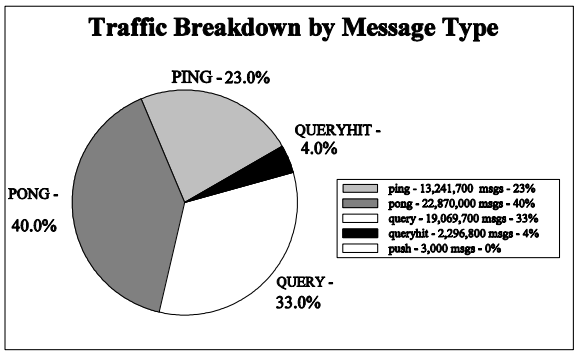

Figure 6.1: Analysis of Gnutella Network Traffic: Message breakdown by Message Type.

6.1 Reducing Query Messages

Our prior analysis on the Gnutella Network Traffic (figure 6.1) revealed that 37% of the network messages where spent on query/queryhits pairs. The ultimate goal of a P2P network is the resource discovery part. We can see from the pie-chart that the Ping/Pong messages, which are used for the host discovery part, occupy a significant share of the total network traffic. This is attributed to weak network connections in the Gnutella Network. In this work we don't consider host discovery related issues. Our goal is to decrease the number of Query/QueryHit messages while being able to discover the same resources.

In our first experiment we performed 10 queries, each of which was run 10 consecutive times over a PeerWare of 104 nodes where each host has a degree of 8. We allow a 5 second interval between each query in order to avoid congesting the network. The queries are keywords randomly sampled from the dataset.

In this set of experiments we measured the number of messages used and the percentage of

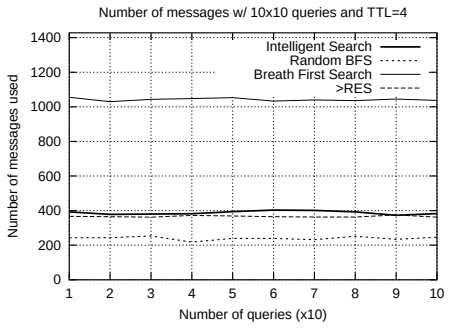

Figure 6.2: **Messages** used by the 4 Algorithms for 10x10 queries **(TTL=4)**

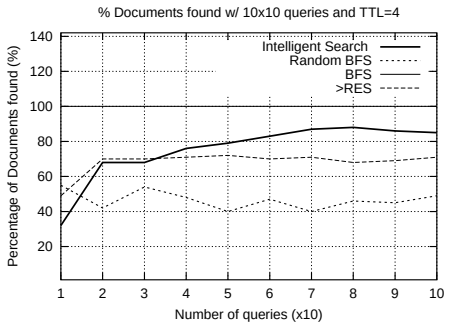

Figure 6.3: **Recall Rate** by the 4 Algorithms for 10x10 queries **(TTL=4)**

documents found in the case where the query messages has TTL=4. Figure 6.2 shows the number of messages required by the 4 query routing techniques. The figure indicates that *Breadth-First-Search (BFS)* requires almost 2,5 times as many messages as its competitors with around 1050 messages per query. BFS's recall rate is used as the basis for comparing the recall rate of the other techniques and is therefore set to 100%. *Random Breadth-First-Search (RBFS), the Intelligent Search Mechanism (ISM)* and the *Most Results in the Past (>RES)* on the other hand use all significantly less messages but ISM is the one that finds the most documents. That is attributed to the fact that ISM improves its knowledge over time. More specifically ISM achieves almost 90% recall rate while using only 38% of the BFS's messages. From figure 6.3 we can see that both >RES and ISM start out with a low recall rate (i.e. 40-50%) because they are initially both choosing their neighbors at random. Therefore their recall rate performance is comparable to that of RBFS. In both figures 6.2 and 6.3, the values shown are the averages of 10 consecutive requests. The above experiments justify our hypothesis that a large number of peers receive unnecessary messages.

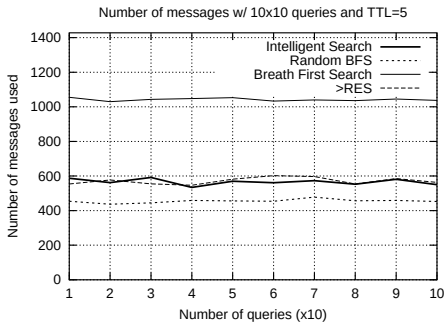

Figure 6.4: **Messages** used by the 4 Algorithms for 10x10 queries **(TTL=5)**

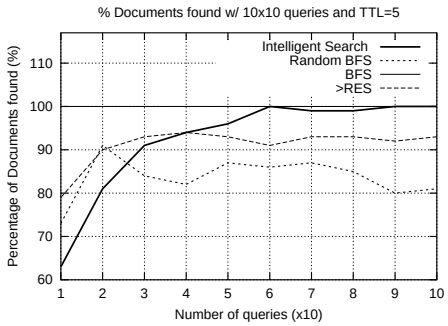

Figure 6.5: **Recall Rate** by the 4 Algorithms for 10x10 queries **(TTL=5)**

6.2 Digging Deeper by Increasing the TTL

In the second experiment we investigated what is the effect of increasing the TTL parameter to our results. Figure 6.5 shows that by increasing the value of the time_to_live field of the search requests (TTL = 5) the Intelligent search mechanism discovers almost the same documents with what BFS finds for TTL = 4. More specifically, our experimental results show that our mechanism achieves 99% recall rate (figure 6.5) while using only 54% (figure 6.4) of the number of messages used in BFS. Again, the recall rate increases as the number of queries increases over time. Another important observation is that the results for both RBFS and ISM are consistent with our analysis, and show that it is possible to search the majority of the P2P network with significantly fewer messages than the brute force algorithm.

6.3 The Discarded Message Problem

We define the *Discarded Message Problem (DMP)* (see figure 6.6) in the following way: A node P_k receives some query q with some TTL_1 at some time t_1. P_k first checks if it has for-

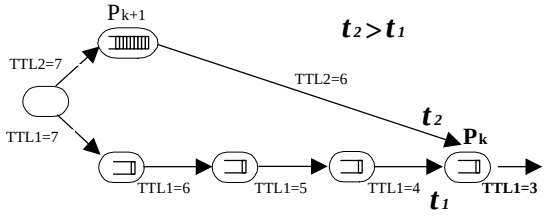

Figure 6.6: The Discarded Message Problem.

warded the same query (identified by GUID) in the past. If yes it will immediately discard the message in order to avoid forwarding the message several times. If not, it will decrease TTL_1=TTL_1-1 and forward q to some of P_k's peers.

Now what happens if node P_k receives q with some TTL_2, where $TTL_2 > TTL_1$ at some time t_2, where $t_2 > t_1$. Most of the commercial P2P clients will again discard q. The result of the DMP problem is that a query reaches less nodes that estimated. We fix the DMP problem by allowing the TTL_2 message to proceed, since this may allow q to reach more peers that its predecessor TTL_1. Of course there is some redundancy which will add up in the "number of messages" graph. Unfortunately without this fix the BFS behavior is not predictable and therefore is not able to find the nodes that we were supposed to find.

Our experiments revealed that almost 30% of the forwarded queries were discarded because of DMP. The experimental results presented in sections 6.1 and 6.2 are not suffering from DMP. This is the reason why the number of messages is slightly higher ($\approx 30\%$) than the expected number of messages. For example in the BFS case our analysis (section 4.1) shows that the total number of messages should be $msgs = \sum_{i=1}^{n}(d_i - 1)$ for n nodes each of which with a degree d_i.

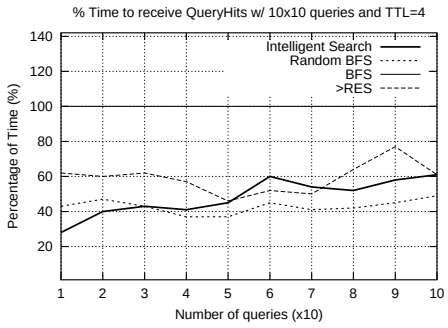

Figure 6.7: **Time** used as a fraction of the time used in BFS for the 4 Algorithms in the 10x10 experiment **(TTL=4)**.

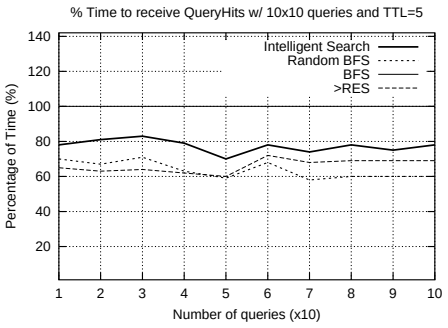

Figure 6.8: **Time** used as a fraction of the time used in BFS for the 4 Algorithms in the 10x10 experiment **(TTL=5)**.

For n=104 nodes and each of degree $d_i = 8$ we would expect msgs=104*(8-1)=728 messages. By re-running the experiments without fixing DMP we get averagely 763 messages per query which is close to the estimated amount.

6.4 Reducing the Query Response Time

Since a query may get an arbitrary large number of query results we define the *Query Response Time (QRT)* as the interval which elapses between t_1 which is the time a node q sends out a query, until t_2 which is the time that q receives the last result from the network. This result is again taken from the experiment where we perform 10 queries 10 consecutive times. Figure 6.7 shows the Query Response Time (QRT), as a percentage of the BFS algorithm, for the three algorithms ISM, $>$RES and RBFS. BFS's QRT is in the order of 6 seconds while the others use only $\approx$30-60% for TTL=4 and $\approx$ 60-80% for TTL=5 of that time. This happens because BFS uses more messages which are consequently congesting the network and therefore increasing the average QRT. The

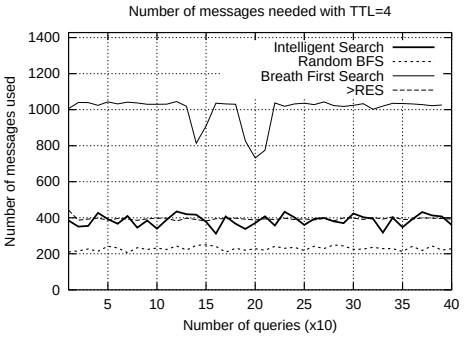

Figure 6.9: **Messages** used by the 4 Algorithms for 400 queries **(TTL=4)**

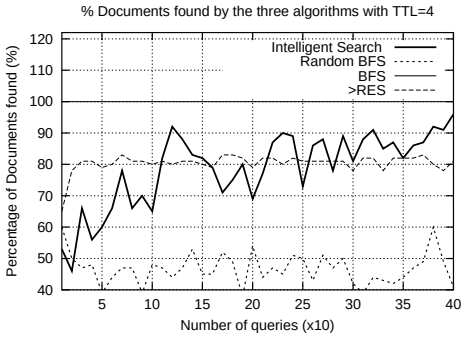

Figure 6.10: **Recall Rate** by the 4 Algorithms for 400 queries **(TTL=4)**

reason why ISM requires slightly more time than >RES and RBFS is that ISM decision involves more manipulation over the past queries.

6.5 Improving the Recall Rate over Time

In the previous experiments we used 10 queries which are performed 10 consecutive times. This scenario suits well the ISM algorithm since the queries are correlated. In this experiment we use 400 queries which are randomly sampled from the dataset. The sampling is biased since we choose to pick approximately 4 queries per country (which consequently also means per dataPeer), for 104 dataPeers. With this assumption we make sure that the queries will refer to all the dataPeers rather than only a subset of them. Each query q consists of 4 keywords $q = \{A, B, C, D\}$ and a dataPeer answers to q by evaluating the union $A \cup B \cup C \cup D$.

On figure 6.9 we can see that during queries 150-200 two major outbreaks occur in BFS. This is basically an indication that some connections (i.e. sockets) broke down and that some query messages were not able to go through. This network instability is incurred by the overwhelming

amount of messages propagated by the BFS algorithm. This might also has to do something with the 63% of Ping/Pong messages (figure 6.1) which are found in the Gnutella Network where weak network connections are translated into an endless effort of peers to discover new hosts.

In this set of experimental results we can see that our ISM mechanism improves its recall rate (figure 6.10) over time approaching nearly 95% recall rate while using again $\approx 38\%$ of BFS's messages. The reason for this is that, as the nodes accumulate more knowledge about their peers, peers that provided answers in the past are still queried in subsequent queries. As more peers become likely to be queried, they themselves continue to explore the network by propagating the requests to their peers.

Chapter 7

Conclusions & Future Work

Peer-to-Peer systems are application layer networks which enable networked hosts to share resources in a distributed manner. Such systems offer several advantages in simplicity of use, robustness and scalability. In this thesis we address the problem of efficient Information Retrieval in such systems. We analyze a number of different techniques and then present the Intelligent Search Mechanism.

The Intelligent Search mechanism (ISM) uses the knowledge that each peer collects about its peers to improve the efficiency of the search. The scheme is fully distributed and scales well with the size of the network. *ISM* consists of four components: A *Profile Mechanism* which logs query-hits coming from neighbors, a *Cosine Similarity* function which calculates the closeness of some past query to a new query, *RelevanceRank* which is an online ranking mechanism that ranks the neighbors of some node according to their potentiality of answering the new query and a *Search Mechanism* which forwards a query to the selected neighbors.

We deploy and compare ISM with a number of other distributed search techniques. Our experiments were performed with real data over PeerWare, our middleware simulation infrastructure which is deployed on 50 workstations.

Our results indicate that ISM is an attractive technique for keyword searching in Peer-to-Peer systems since it achieves in some cases 100% recall rate by using only 54% of the messages used in the flooding algorithm. The results further show that ISM improves over time because nodes learn more information about their neighbors as time elapses. ISM achieves therefore a better recall rate than its competitors, although its initial performance is similar to them. Lastly we have shown that ISM requires approximately the same Query Response Time (QRT) with its two main competitors RBFS and >RES and far less QRT than BFS.

For future work we plan to probe our algorithms over new network topologies such as powerlaw and tree. Stemming and Stop Word Lists are another improvement that can be incorporated in our algorithm. In that way we will be able to route queries to the appropriate hosts even if the past queries contain different query terms which have the same semantics. We believe that the Peerware simulation infrastructure is an invaluable tool in evaluating the applicability and performance of different routing techniques. We further plan to introduce different degrees of data sources replication and see how do these affect the performance of our search techniques. We are also interested in deploying PeerWare over a Wide Area Network including hosts which are geographically distributed.

Bibliography

[1] American National Standards Institute American National Standard X9.30.21997: Public Key Cryptography for the Financial Services Industry - Part 2: The Secure Hash Algorithm (SHA-1) (1997)

[2] B. H. Bloom. ”Space/Time Trade-Offs in Hash Coding with Allowable Errors”, Communication of the ACM, 13(7):422-426, 1970

[3] Ozalp Babaoglu, Hein Meling and Alberto Montresor ”The Anthill Project” In Proceedings of the 22th International Conference on Distributed Computing Systems (ICDCS ’02), Vienna, Austria, July 2002.

[4] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, ”Freenet: A Distributed Anonymous Information Storage and Retrieval System” in Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, Springer: New York (2001).

[5] Clip2, ”Gnutella: To the Bandwidth Barrier and Beyond”, November 6, 2000, http://www.clip2.com/gnutella.html

[6] A. Crespo, H. Garcia-Molina. Routing Indices For Peer-to-Peer Systems. *Proc. of Int. Conf. on Distributed Computing Systems*, Vienna, Austria, 2002.

[7] Francisco Matias Cuenca-Acuna and Thu D. Nguyen. ”Text-Based Content Search and Retrieval in ad hoc P2P Communities”, International Workshop on Peer-to-Peer Computing, Springer-Verlag Vol:2376, May 2002

[8] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In SIGCOMM, pages 251-262, 1999.

[9] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, ”Summary cache: A scalable wide-area web cache sharing protocol,” IEEE/ACM Transactions on Networking, vol.8 number 3”, pages 281–293, 2000.

[10] Peter Fankhauser, Gerald Huck and Ingo Macherius ”Components for Data Intensive XML Applications” ERCIM News No.41 - April 2000

[11] Gnutella, `http://gnutella.wego.com`.

[12] Groove Networks `http://www.groove.net/`.

[13] Google `http://www.google.com/`.

[14] IBM alphaWorks, XML Parser for Java, `http://www.alphaworks.ibm.com/tech/xml4j/`

[15] M. Jovanovic, ”Modeling Large-scale Peer-to-Peer Networks and a case study of Gnutella”, Master’s Thesis, University of Cincinati, April 2001.

[16] Kazaa, `http://www.kazaa.com`

[17] James F. Kurose and Keith W. Ross ”Computer Networking: A Top-Down Approach Featuring the Internet” pages 286–289, Addison Wesley Longman 1999

[18] Leda, Algorithmic Solutions Software GmbH, `http://www.mpi-sb.mpg.de/LEDA`.

[19] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. ”Search and replication in unstructured peer-to-peer networks”, ICS02, New York, USA, June 2002.

[20] V. Kalogeraki, D. Gunopulos and D. Zeinalipour-Yazti *Proceedings of the ACM CIKM International Conference on Information and Knowledge Management (CIKM)*, McLean, VA, USA, pages: 300–307, November 2002

[21] Ken Mccrary, ”The JTella Java API for the Gnutella network”, October 2000, `http://www.kenmccrary.com/jtella/`.

[22] Stanley Milgram Milgram’s experiment description, available at: `http://smallworld.sociology.columbia.edu/description.html`.

[23] Napster, http://www.napster.com.

[24] Stephen North, Emden Gansner, John Ellson, ”Graphviz - open source graph drawing software”, http://www.research.att.com/sw/tools/graphviz/

[25] REUTERS-21578 dataset. http://www.research.att.com/ lewis

[26] M.Ripeanu, ”Peer-to-peer Architecture Case Study: Gnutella Network”, Technical Report, University of Chicago, 2001.

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. *Proc. of ACM SIGCOMM, pages 149–160, August 2001.*

[28] B. Yang and H. Garcia-Molina, Comparing hybrid peer-to-peer systems. *Proc. 27th Int. Conf. on Very Large Data Bases (Rome),* pages 561-570, September 2001.

[29] B. Yang, H. Garcia-Molina, Efficient Search in Peer-to-Peer Networks. *Proc. Int. Conf. on Distributed Computing Systems,* 2002.

[30] D. Zeinalipour-Yazti and T. Folias, ”Quantitative Analysis of the Gnutella Network Traffic”, Dept. of Computer Science, University of California, Riverside, June 2000

Appendix

JAVA source code for *dataPeer*

```
/*
 * (C) Copyright University of California Riverside. 2001–2002.
 *
 * dataNode     – A reuters XML News servent
 *
 * Version                  : 1.0
 * Document author          : Demetris Zeinalipour (csyiazti@cs.ucr.edu)
 * Supervisor               : Dimitris Gunopulos (dg@cs.ucr.edu)
 * Computer Science Department , University of California, Riverside
 *
 * Our code makes use of some classes from the GMD–IPSI XQL Engine developed
 * at the German National Research Center or Information Technology.
 * The initial version of ucr.core was based on the JTella API
 *
 * Notice:      This file is not self–containing since it refers to many classes
 * that could not be presented due to space limitations. The purpose of the file
 * is to provide the general ouline of the structure of dataPeer.
 */

package ucr.gnudb;

// P2P classes
import ucr.core.NetConnection;
import ucr.core.ConnectionData;
import ucr.core.FileServerSession;
import ucr.core.Host;

// PDOM–XQL classes
import de.gmd.ipsi.xql.*;
import org.w3c.dom.*;
```

```
import java.util.*;
import java.net.InetAddress;


/**
 * A dataNode Node.
 *
 */
public class dataNode
{
   // The interface to the P2P network.
   private NetConnection conn;

   // A PDOM Manager that parses and handles the XML repository.
   private static PDOMmanager pdom;

   // An XQL resultset for queries.
   private static XQLResult rxql;

   /**
    * The main loop
    */
   public static void main(String[] args)
   {
    // get my id a command line argument e.g. australia
    CONFIG._COUNTRY = args[0];

    // initialize parameters such as path,ttl,socket timeouts, log dirs, etc.
    CONFIG.initGlobalConfig("config.txt");

     // get details about the hosts that this node should connect to
    CONFIG.initLocalConfig(CONFIG._CONF_BASE + CONFIG._COUNTRY + ".txt");

    System.out.print("UCR " + CONFIG._COUNTRY + " dataNode starting ...\n");

    System.out.println("\n\nTTL = " + CONFIG._TTL);
    System.out.println("SOCKET_TIMEOUT = " + CONFIG._SOCKET_TIMEOUT);


     // Create PDOM Object
     pdom = new PDOMmanager(CONFIG._COUNTRY + ".xml");
     //rxql = pdom.getALLCriterionNodes();
```

```
        try
        {
          // create a ConnectionData object that sets connection preferences
          ConnectionData connectionData = new ConnectionData();
          connectionData.setIncomingPort(CONFIG._MYPORT);

          // create a NetConnection that launches a server on this IP and port
          // and accepts incoming messages
          NetConnection c = new NetConnection(connectionData);

          // populate the HostCache Object so that we start outgoing connections
          for (int i=0; i<CONFIG.myPeers.size(); i++)
          {
                Host host = new Host( (String) CONFIG.myPeers.elementAt(i),
                                      (Integer)CONFIG.myPeers.elementAt(i) );
                c.getHostCache().addHost( host );
          }

          // start the outgoing connections
          c.start(CONFIG.ALGORITHM, CONFIG._TTL, CONFIG._SOCKET_TIMEOUT);

          // listen for any incoming queries and reply with QueryHits
          QueryReceiver receiver = new QueryReceiver(pdom);
          FileServerSession s = c.createFileServerSession(receiver);
          receiver.setFileServerSession(s);

          // run forever – endless loop
          while (true)
          {
              Thread.sleep(30000);
          }

        }
        catch (Exception e)
        {
          // catch and print errors
          e.printStackTrace();
        }
      }
}
```