Peer-to-Peer content search supported by a distributed index in a publication/search model

Tolosa, Gabriel H.1, Peri, Jorge A. y Bordignon, Fernando R.A. Universidad Nacional de Luján. Departamento de Ciencias Básicas Laboratorio de Redes Cruce rutas 5 y 7 – Luján – Bs. As. Argentina. 
{tolosoft, jperi, bordi}@unlu.edu.ar

ABSTRACT: Peer-to-peer networks (P2P) are considered a valid approach for the construction of distributed systems. Further research projects in the last few years have focused on using this kind of networks as an alternative for solving different situations that have traditionally required centralized servers, such as search engines. This paper deals with the problem of content search in highly distributed and dynamic environments. We propose and evaluate a distributed index model built upon a peer-to-peer network which supports complete indexing of text documents and allows searching by content. A distinctive feature of this proposal is that it requires no specific network topology or hierarchy. Evaluations with different settings were performed by simulating a 10,000-node network, where each node had the capability to share documents. With regard to the traffic generated, experiments show an improvement in efficiency of between 84% and 93% over similar systems like Gnutella. The evaluation of retrieval performance using a test collection showed that the P2P system was able to achieve the same level of performance as the centralized system. It was also found that the amount of traffic generated by this model varies between 80 and 225 Kb per set of query and answers.

Categories and Subject Descriptors
C.2.1 [Network architecture and design]: Distributed networks; H.3.4 [Systems and Software]: Information networks

General Terms
Peer to peer networks, Content search

Keywords: P2P Networks, Nodes, Distributed index model, P2P network evaluation

Received 21 Feb. 2006; Revised 29 June 2006; Accepted 12 July 2006

1. Introduction

Peer-to-peer networks (P2P) [3] are considered a valid approach for the construction of distributed systems. Further research projects in the last few years have focused on using this kind of networks as an alternative for solving different situations that have traditionally required centralized servers, such as search engines.

The first peer-to-peer systems were based on sharing multimedia files, especially music and videos. These types of digital objects have well-known identifiers: their file names and metadata descriptions (title, singer), making it possible to located them by searching for these words. P2P networks are generally composed of nodes with the same capabilities which are able to communicate in order to exchange information directly. When used as a communication infrastructure, these nodes enable the construction of distributed information systems in a decentralized way. Moreover, the systems based upon this model have interesting features such as fault tolerance, self-organization and load balancing, which turns them into a robust platform for different distributed services. For further information on P2P systems see [17] and [3].

Peer-to-peer networks are an appealing approach to search over distributed digital repositories [14]. When using a P2P network for distributed search, it becomes necessary to consider how the massive distribution of resources affects the behavior of traditional search algorithms. The study of distributed search, formally known as Distributed Information Retrieval, can be divided into three main issues [4]:

- **Resource Description**: Involves the description of the contents of each text database in order to disseminate this information over the network.
- **Resource Selection**: Consists in making decisions about the usefulness of each database description to answer a given query. In P2P networks this task is more complex due to the massive distribution of resources.
- **Result Merging**: The results provided by each database search server are combined to form only one ranked list which is then presented to the user.

Peer-to-peer networks pose a challenge to solve these tasks in an efficient way, especially those of resource selection. This is due to the fact that information can be placed (e.g. by routing algorithms) in different nodes, some of which can have special features like an index or directory service. Unlike P2P file sharing systems [2, 8, 12], where resources are identified by their names and metadata, content search systems are more complex because:

a) They require richer resource descriptions containing all the terms that will be used as keywords.
b) Search functions must make use of document contents.
c) Proper Information Retrieval algorithms must be used to include relevance scoring of documents related to the queries.

Although P2P networks have been used mainly in file sharing systems based on end-user computers, this model may also be useful for text documents, where each user could offer the others the possibility of accessing their files according to their contents. In this case, the distributed search problem extends to all the nodes which have documents to share, so it becomes fundamental to design architectures and methods for these purposes. An important disadvantage is the fact that documents in P2P networks are distributed randomly [28] as to their semantics.

This paper deals with the problem of content search in highly distributed and dynamic environments. We propose the use of a P2P network with a distributed index model which supports the mapping of terms and network nodes to make a search process possible. This index is divided according to the letters of the alphabet and each portion assigned to different index nodes. In addition, an index node selection algorithm and a query routing strategy are proposed. The main idea is to replace query flooding techniques like Gnutella’s with a distributed index, which acts as the first step for answering the queries. After the index nodes return a set of peers which are believed to have relevant documents, the query is routed to those peers, reducing network traffic.

This model is partly based on publish/subscribe systems, where resource providers publish their data and consumers do the same with their search profiles on a broker node. Then, each time a new resource matches a given profile, a filtering service sends a message to the corresponding consumer. In our new approach, there exists a publish service but instead of posting a profile, users first send a query to the index and then to the retrieved nodes. This mechanism can be considered as a publish/search system, where a proactive role must be taken by the requesting nodes.

---

A distinctive feature of our proposal is that it requires no specific network topology or hierarchy. Although there exist nodes with special functions and others which connect to them, the network topology is defined by direct interactions and the dynamic nature of a P2P environment. Evaluations of retrieval performance and traffic generation with different settings were performed by simulating a 10,000-node network, where each node had the capability to share documents. Achieving high overall search performance in distributed environments is a complicated task, affected by factors such as dynamic topology and resource distribution [14]. On the other hand, Gummadi et al. [9] found that the traffic generated in file sharing systems is greater than that of the remaining Internet services, so it becomes an important scalability factor.

The rest of the paper is organized as follows. Section 2 introduces related work and section 3 describes the model proposed. Section 4 explains the evaluation methodology and experimental settings. Results are presented in section 5. Finally, section 6 concludes and suggests some future work.

2. Related work
Although the first efforts to localize resources [1, 2, 8, 19] were focused on the use of identifiers, such as a filename or part of it, newer ideas have aimed at retrieving documents by their content. Some proposals are based on query forwarding techniques like Gnutella, implementing changes which help improve efficiency. Such proposals can be seen as "query routing" strategies, also known as Resource Selection in the field of Distributed Information Retrieval.

Yang and García-Molina [36] proposed three techniques, named Directed BFS, Iterative Deepening and Local Indices, which allow the selection of a sub-set of nodes to forward the query to. Continuing the same line, Zhuang et al. [38] suggested Hybrid Periodical Flooding as a variation of the K-Walker [16] algorithm. This technique uses an adaptive search method to establish the number of nodes to which the query should be forwarded.

On the other hand, there are systems that use Distributed Hash Tables (DHT) [7, 21, 22] which allow them to map nodes and resource identifiers on an n-dimensional space. These systems establish a network structure where they can define an "overlay network" from which the mapping and searching functions are efficiently performed using resource names or associated metadata. Nevertheless, some researches in this field have recognized its limitations for information retrieval from full text-document content and have claimed to continue their research into P2P networks with the basic features of Gnutella [6].

As regards publish/subscription systems, some of them are based upon P2P networks and particularly implemented using DHTs. In [29], the authors propose a model with these features where the search service is implemented with a hash table using the Chord algorithm [7] and high expressivity filters. The authors believe that the main advantage lies in the filters’ capabilities together with the scalability and fault tolerance given by P2P systems. Other proposals like Hermes [21] which is based on Pastry [24] or Meghdoot [10] which uses a CAN [22] distributed hash table, are built this way too.

Tang et al. [28] designed an information retrieval system using P2P networks, named pSearch, which avoids using flooding techniques for query forwarding. Basically, it defines a semantic vector for each document calculated by means of Latent Semantic Indexing. As a result, similar documents are placed in nearer nodes and the search process is done by calculating a similarity coefficient between each document and the semantic vector of the query expression.

Lu and Callan [15] presented an application which made it possible to select nodes and retrieve documents using a directory service provided by special nodes in the network. End-user nodes were grouped by their content using a clustering algorithm which, based on the "neighborhood" concept, determined which nodes the query should be routed to. In other work, the same authors [14] proposed a distributed search system for digital libraries based on a hierarchical P2P network. This was formed by special nodes called hubs, which managed the resource descriptions of a set of leaf nodes. Then, hub selection and leaf node functions were implemented to determine the best nodes that could answer a query. In this work, resource description and resource selection methods are proposed too.

These last two works were conceived and evaluated as distributed information retrieval systems. Their design employs document retrieval by content together with traditional IR evaluation metrics. The research reported in this paper continues in the same line and gives an alternative approach to the same problem.

3. The aDICS distributed index
The aim of the aDICS Model [30] is to split the index of terms and nodes on a P2P network. This is done according to letters of the alphabet, that is, each node manages a part of the index with terms beginning with a particular letter. It is possible for two or more nodes to manage the same letter index space. However, they must have different contents.

The model defines two types of nodes: Index Nodes (IN) and Ordinary Nodes (ON). Any of them can have documents and publish their terms but only Index Nodes take part in the management of the distributed index by keeping a portion of it corresponding to an assigned letter. Besides, all the nodes in the network must keep a Contact Table (CT) with at least one node address per letter, thus having 26 entries (Figure 1). The creation and maintenance of Contact Tables are carefully explained in section 3.1.

Whenever a node has a document to share, the most representative terms of the document are selected and published in the index nodes which manage their leading letters. For example, to post the term "colina" a publish message must be sent to an index node in charge of letter "C". With this information, the index node maintains its portion of the Inverted Index containing the associations between a term and the list of nodes with documents that have it (Figure 2). Initially, only terms of three or more characters in length are considered, although this rule can be improved with information retrieval techniques. Such an issue is beyond the scope of this work.

<table>
<thead>
<tr>
<th>Letter</th>
<th>Node address</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>646; 125</td>
</tr>
<tr>
<td>B</td>
<td>154</td>
</tr>
<tr>
<td>Y</td>
<td>147; 287; 156</td>
</tr>
<tr>
<td>Z</td>
<td>200; 874</td>
</tr>
</tbody>
</table>

Figure 1. A node’s Contact Table

<table>
<thead>
<tr>
<th>Terms</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casa</td>
<td>154; 165; 947; 877</td>
</tr>
<tr>
<td>Colina</td>
<td>472; 554; 489</td>
</tr>
<tr>
<td>Corona</td>
<td>587; 154; 554; 145</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 2. Part of the inverted index stored inside a node

The whole process involves: a) selecting documents to share, b) selecting representative terms from the documents (local index), c) publishing each term in the corresponding index nodes and d) updating or refreshing the published information. Once this process is complete, the network is said to have enough distributed information to support the search service.

3.1 Network node joining
In our model, like any other P2P system, every node must know at least one valid node address of a participant to join the network. When a new node becomes part of the network it is necessary to decide whether it will be part of the distributed index. This is done by a weighting function that determines whether the node has enough computational resources to perform that task (e.g. CPU, bandwidth). If the new node becomes an index node, an algorithm (Figure 3) will assign it a letter to manage. The next step is the update of the node’s CT, which is initially empty for every new node. This process is done by exchanging CTs with the node it is connected to and updating their information. The updated CT is stored in a local cache for future use.

Journal of Digital Information Management  Volume 4 Number 3  September 2006  176
if (ManageIndex(P(ndx)))
{
    if (IncompleteTable)
    {
        LT = SelectLetterFromEmptyEntry
        AssignLetter(LT)
    } else
    {
        LT = SelectRandomLetter(P(l))
    }
} else
{
    LT = "~"
}
return LT

Figure 3. Algorithm for assigning a letter to an index node

3.2 Term publishing

To enable search functions, the first thing to do is the local index
process and the extraction of the most relevant terms from the
documents shared by each node. As was mentioned earlier, it is
necessary to apply information retrieval techniques to select the set
of terms to be used, e.g. according to their raw or relative frequency
or their position inside the document. This is done to reduce the
number of terms to publish, thus improving efficiency. Once these
terms have been selected, they are grouped by their first letter and
a node Si,l in charge of that letter is selected from the CT. A publish
message is then sent to Si,l announcing the new terms which can be
used to solve future queries. The publishing process is shown in
Figure 4.

This example shows two nodes with terms to publish. Node 100
has the terms “casa” and “árboles” whereas node 140 has the terms
“árboles”, “colina” and “bota”. Node 100 looks inside its contact table
and finds two index node addresses for letter “C”: 102 and 165. It
takes the first and sends this node a publish message with the term
“casa” (step 1). The receiving node checks the correctness of the
term's first letter and adds an entry to its index (step 2). Next, node
100 repeats the process for the term “árboles”. In this case, the publish
message is sent to node 244 (step 3) which adds its corresponding
table entry (step 4).

This process is repeated for node 140, which must publish three
terms. For the first two, it will send publish messages to nodes 244
and 102 respectively (steps 5 through 8) whereas it will publish the
term “bota” locally given the fact that the node 140 itself is in charge
of a part of the index corresponding to letter “B”.

During the publishing process a node may detect the absence of an
index node (for example, when an acknowledgement message is
not received) and delete its entry from the CT. Every node must keep
a minimum number of nodes for each letter. Otherwise, it should
choose another node at random to exchange CTs in order to complete
its entries. This update may also occurs during query resolution.

3.3 Query Resolution

To solve a query a node uses the information stored in its own CT to
identify the index nodes responsible for the part of the distributed
index at random corresponding to the initial letter of each term in the
query. For example, suppose that node 105 needs to solve a query
with terms “árboles” and “colina”. First of all, it searches inside its CT
to get index node addresses:

IN_{árbol} = lookup(CT, “árboles”) ⇒ IN_{árbol} = {244, 342}
IN_{colina} = lookup(CT, “colina”) ⇒ IN_{colina} = {102}

Then, it asks the corresponding index nodes. First, it requests “árboles”
to node 244 and gets a list of possibly useful nodes. After that, it
does the same to node 102.

R_{árbol} = ask(244, “árboles”) ⇒ R_{árbol} = {100, 140}
R_{colina} = ask(102, “colina”) ⇒ R_{colina} = {140}

As the query is a conjunction of two terms, the final answer is
obtained by calculating the intersection set of all partial answers.

R = R_{árbol} \cap R_{colina} ⇒ R = {140}

In this example, node 140 has published both terms (“árboles” and
“colina”) in a distributed way, but this does not mean that both terms
are inside the same document. In a later step, the requesting node
will send a query to node 140 with both terms. Finally, it will evaluate
the answer.

4. Evaluation Methodology

4.1. Procedures

In order to validate our proposal a 10,000-node P2P network was
simulated in laboratory, using an alphabet of 26 letters. Two kinds of
experiments were conducted. The first (Eval-I) was carried out to
study the traffic generated for later comparison with the Gnutella
protocol. The second (Eval-II) had the purpose of evaluating the
performance of the model as an information retrieval system. The
software used in the experiments was coded in Perl v5.6.1 on
Linux. The whole simulation process was divided into the following
phases:

Figure 4. Term Publishing Process
- Random Network generation: Comprised the definition of a network and the assignment of documents to several nodes according to different settings. The nodes contact tables were filled by generating random links between nodes.

- Local Indexing and Publishing: In this stage, text documents were processed to extract representative terms, which were later published according to the information included in contact tables.

- Querying: Several sets of queries were executed on different randomly-chosen initial nodes. The queries were obtained from different sources and procedures. After an initial node selection using the aDICS index, local searching within the node was done using the Lemur Toolkit [13].

- Result Merging: After receiving answers from the nodes, a fusion algorithm was used to determine the final ranking which would be presented to the user. In this case, the well-known CombSUM [34] algorithm was used with a normalized weighting scheme.

4.2 Test Data
For Eval-I, the test dataset consisted of all the abstracts from the year 1990 to 2003 of part 1 of the National Science Foundation abstract repository [20]. A total of 49,078 documents were processed, which accounted for 30,799 different terms (the lexicon) used in the indexing stage. Different queries were generated at random with 1, 2 and 3 terms from the lexicon. 500 queries were used in each of the cases and experiments.

For Eval-II, it was necessary to build a test collection that conformed to the evaluation methodology proposed in TREC conferences [32]. Such a collection should consist of: a) a set of documents, b) a set of information needs and c) relevance judgments. We used the Ohsumed test collection [11] from 87-91, which contained 348,566 documents with 106 information needs, totaling about 390 Mb of text. Documents with a minimum of three lines were selected, yielding a total of 336,942. Finally, the entire collection was divided into several parts according to different experimental settings in order to evaluate the retrieval performance of the model.

The experiments were conducted using two sets of queries. The first, named Q106, was obtained directly from the information needs of the collection and was evaluated using their corresponding relevance judgments. A second set was obtained following Liu and Callan's [15] methodology. A new set of 1,000 queries (P1000) with variable length between 1 and 5 terms was created by extracting terms from the lexicon at random. The relevance judgments were obtained by executing the queries P1000 on a centralized retrieval system and considering the first 100 answers as relevant documents. The idea behind this method was to evaluate how the P2P system could reproduce the performance of the centralized system, at least for the first n answers. In all cases, both documents and queries were processed to extract stopwords and apply stemming rules using a Porter stemmer. The retrieval tool used was the Lemur Toolkit [13] configured to use a TF/IDF weighting scheme.

4.3 Metrics
4.3.1 Traffic Measurement
The exact amount of traffic generated by the model is difficult to measure given the fact that the nodes can answer each query with an answer list of variable length. This situation is similar to Gnutella's. To overcome this problem the amount of traffic generated and the variable parameters were experimentally determined in Eval-I. The total query cost (Ctc) of 1-term queries would be as follows:

\[
\text{Ctc} = \text{Cci} + \text{Qri} 
\times \left( \text{Leti} + \text{Le} + \text{Cq} \right)
\]  

(1)

Having:

\[
\text{Cci} = 2 \times (\text{Leti} + \text{Le}) + \text{Cq} + \text{Cr}
\]  

(2)

and:

\[
\text{Cr} = \text{Qri} \times (\text{LI} + \text{Ls})
\]  

(3)

Where

- \text{Ctc} \text{ Cost of a query to an index node (including its answers)}
- \text{Cci} \text{ Average number of answers}
- \text{Leti} \text{ Length of Network and Transport protocols headers}
- \text{Le} \text{ Application protocol header length}
- \text{Cq} \text{ Cost of query expression (in bytes)}
- \text{Cr} \text{ Cost of answer (from an index node)}
- \text{LI} \text{ Node ID length}
- \text{Ls} \text{ Field separator length}

In the case of queries with two or more terms, the expression (1) is modified to reflect that there is a new different query for each term. Furthermore, the original query is re-sent only to the nodes provided in the answers which contain all the requested terms. Thus resulting:

\[
\text{Ctc} = (\text{tq} \times \text{Cci}) + \text{Qria} \times (\text{Leti} + \text{Le} + \text{Cq})
\]  

(4)

Where,

- \text{tq} \text{ Number of terms in the query}
- \text{Qria} \text{ Conjunction (AND) of the average number of answers (Qri)}

These calculations allow for the cost associated with control data included in network, transport and application protocol headers. The amount of traffic generated was also computed for Eval-II, where there was a variation in the cost owing to the distribution of documents in the P2P network nodes.

4.3.2 Measurement of Retrieval Accuracy
Traditional IR evaluation metrics were used to determine the retrieval accuracy of the proposed model. Basically, Recall and Precision were used, where Precision is the proportion of retrieved documents that are relevant, and Recall is the proportion of relevant documents that are retrieved.

\[
\text{Recall} = \frac{| \text{Relevant Retrieved} |}{| \text{Relevant} |}
\]

\[
\text{Precision} = \frac{| \text{Relevant Retrieved} |}{| \text{Retrieved} |}
\]

In addition, the different values of precision were analyzed every 5, 10, 15, 20, 30 and 100 documents in order to measure the retrieval accuracy for the first n documents, which are usually the most interesting to users [14]. Measures R_Precision and Avg_Precision were also calculated, yielding precision over all the relevant documents and average system precision respectively. All results were obtained using the widely-used trec_eval [31] application.

5. Experiments and Results
5.1 Traffic (Eval–I)
A series of experiments was conducted to study the volume of generated traffic. For this purpose, four settings were established (Table 1) which enabled the testing of the distribution of terms inside

Table 1. Experimental settings

<table>
<thead>
<tr>
<th>Exp #1</th>
<th>Exp #2</th>
<th>Exp #3</th>
<th>Exp #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network size (nodes)</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>P(n</td>
<td>\text{nd}x) = \text{Probability of a node managing a part of the index}</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>P(\text{doc}–\text{having documents to share})</td>
<td>0.50</td>
<td>0.50</td>
<td>0.75</td>
</tr>
<tr>
<td>Maximum number of documents per node qMaxDocsxNode</td>
<td>200</td>
<td>400</td>
<td>200</td>
</tr>
</tbody>
</table>

The probability of a node managing a part of the distributed index was empirically determined in [30] by comparing the amount of generated traffic and the number of index nodes for a letter. The
The probability of a node having documents to share and the maximum number of such documents (qMaxDocsxNode) were estimated from previous studies [25].

Those works show that in Gnutella about 75% of the nodes share 100 or fewer files, with the number of documents varying widely from node to node. Since Gnutella is only a file sharing system, such a basic estimation must be adjusted for the proposed system.

The probability of a node managing a part of the distributed index was empirically determined in [30] by comparing the amount of generated traffic and the number of index nodes for a letter. The probability of a node having documents to share and the maximum number of such documents (qMaxDocsxNode) were estimated from previous studies [25]. Those works show that in Gnutella about 75% of the nodes share 100 or fewer files, with the number of documents varying widely from node to node. Since Gnutella is only a file sharing system, such a basic estimation must be adjusted for the proposed system.

For each letter, the average number of nodes which selected the same index node (Qnil) was calculated. This is equal to the proportion between the number of nodes per letter (Qn) and the number of chosen nodes per letter in the routing tables (Qnt) of the 10,000 nodes. This parameter enables the estimation of the coverage of one index node. A value of Qnil = 213 was observed [30], which means that an index node stores information about another 213 nodes in the network. Experimentally derived values where used in future calculations (Table 2).

For experiment #2, replacing in (3) yields:

\[
C_{ci} = 2 \times (40 + 20) + 8 + 154
\]

Finally, from (1):

\[
C_{tc} = 282 + 22 \times (40 + 20 + 8)
\]

\[
C_{tc} = 1778 \text{ bytes}
\]

Thereby, the cost of a query to an index node and the subsequent deliveries to candidate nodes equal 1778 bytes. According to the observed Qnil value, this means that with 1778 bytes the system is able to reach the content published by 213 nodes representing the required letter. In the case of settings for experiment #4, the calculation is as follows:

\[
C_{tc} = 394 + 38 \times (40 + 20 + 8) = 2978 \text{ bytes}
\]

5.1.1 Comparison with Gnutella

The functionality of the proposed model can be directly compared with the BFS technique, whose efficiency was studied in [36, 37]. The paradigmatic protocol that implements BFS is Gnutella, which – according to Ritter [23] – has scalability limits. However, other researchers have claimed that this formula does not model the behavior of Gnutella, arguing that it is only applicable to networks with tree topology and a balanced number of connections per node. Furthermore, they showed that the probability of establishing a tree topology decreases quickly with the number of hops [26]. In [27], Schollmeier et al. present a study in which they determine that the number of messages sent is significantly fewer than that suggested for tree topologies. Finally, they concluded that there is no exponential growth in Gnutella and that its traffic grows more slowly than expected.

With this study as a reference, a comparison with our proposed model was made. The values obtained by the analyzer using different settings of direct connections with TTL = 7 (Table 3) were directly used. It can be observed in [27] that the value obtained is noticeably lower than in theory.

The amount of generated traffic was calculated according to the estimation of the average message size in Gnutella, namely 83 bytes. Moreover, it is important to consider that this calculation does not include messages dropped by intermediate nodes, as they are impossible to quantify using this methodology. Nevertheless, other studies [35, 36] claim that 30% of the messages are dropped for being redundant, so the amount of generated traffic should increase. To reach a similar number of nodes and perform the query, the proposed model needs to consult some index nodes first and candidate nodes later. Table 4 presents the generated traffic for settings of experiments #2 and #4.

<table>
<thead>
<tr>
<th>Qri</th>
<th>Exp #1</th>
<th>20 nodes</th>
<th>Number of answers from index nodes (to one-term queries)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exp #2</td>
<td>22 nodes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exp #3</td>
<td>29 nodes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exp #4</td>
<td>38 nodes</td>
<td></td>
</tr>
<tr>
<td>Leti</td>
<td>40 bytes</td>
<td>TCP and IP Header Length</td>
<td></td>
</tr>
<tr>
<td>Le</td>
<td>20 bytes</td>
<td>Protocol Header Length</td>
<td></td>
</tr>
<tr>
<td>Cq</td>
<td>8 bytes</td>
<td>Average Term Length</td>
<td></td>
</tr>
<tr>
<td>Li</td>
<td>6 bytes</td>
<td>Node Network Address</td>
<td></td>
</tr>
<tr>
<td>Ls</td>
<td>1 bytes</td>
<td>Field Separator Length</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Parameter values

Figure 5 shows comparative results according to different settings for one-term queries.

Figure 6 shows a comparison of traffic generated for queries of 1, 2 and 3 terms in length. It is interesting how 2-term queries became the most efficient, contrary to our expectations. However, this behavior is justified by the fact that the cost of adding one or two terms is not so important as the number of new queries to be sent according to Qri. In the case of 3-term queries, the Qria value is close to that of 2-term queries so the Cci value determines the difference.
# of connections # reached nodes Generated traffic (Bytes)
1.29 224 18592
5.00 631 52373
11.90 1248 103584
17.84 1711 142017
24.29 1912 158696
40.76 2506 207998

Table 3. Reached nodes and generated traffic in [27]

<table>
<thead>
<tr>
<th>Reached nodes</th>
<th>Generated traffic (Bytes)</th>
<th>Exp.#2</th>
<th>Exp.#4</th>
</tr>
</thead>
<tbody>
<tr>
<td>213</td>
<td>1778</td>
<td>2978</td>
<td></td>
</tr>
<tr>
<td>639</td>
<td>5334</td>
<td>8934</td>
<td></td>
</tr>
<tr>
<td>1278</td>
<td>10668</td>
<td>17868</td>
<td></td>
</tr>
<tr>
<td>1704</td>
<td>14224</td>
<td>23824</td>
<td></td>
</tr>
<tr>
<td>1917</td>
<td>16002</td>
<td>26802</td>
<td></td>
</tr>
<tr>
<td>2556</td>
<td>21336</td>
<td>35736</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Reached nodes and generated traffic for Exp#2 and Exp#4

Figure 7 shows comparative results. These demonstrate that the proposed model achieves a reduction of about 90% in the case of experiment #2 and 84% in experiment #4. These values amount to 93% and 88% respectively if traffic rises by 30% owing to dropped messages (SCH+30%).

5.2 Retrieval Performance (Eval–II)

In order to evaluate retrieval performance, the Ohsumed corpus was partitioned according to 4 settings (Table 5), changing the distribution of documents among the nodes. It should be noted that in experiment #8 all the documents were distributed among the 10,000 nodes, taking complexity to the utmost.

Each node used the Lemur Toolkit as an indexing and retrieval engine, which was configured to work with the TF/IDF weighting scheme. All the nodes were assumed to belong to the same content distribution system, which is why only one information retrieval engine has been used.

<table>
<thead>
<tr>
<th>Network size (nodes)</th>
<th>Exp #5</th>
<th>Exp #6</th>
<th>Exp #7</th>
<th>Exp #8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(ndx)</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>P(doc)</td>
<td>0.25</td>
<td>0.50</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>Nodes with documents to share</td>
<td>2452</td>
<td>4968</td>
<td>7384</td>
<td>10000</td>
</tr>
<tr>
<td>Number of documents</td>
<td>346197</td>
<td>373347</td>
<td>410411</td>
<td>459753</td>
</tr>
<tr>
<td>Duplicated documents</td>
<td>@ 3%</td>
<td>@ 11%</td>
<td>@ 22%</td>
<td>@ 36%</td>
</tr>
</tbody>
</table>

Table 5. Settings for retrieval experiments

As was previously mentioned, the retrieval output of the distributed system was compared to that of its centralized counterpart, which was taken as a baseline. In all cases, 10 index nodes per letter were consulted. After receiving their answers, node intersections were calculated according to the proposed protocol and the nodes which had all the terms (n) of the query or one less (n – 1) were selected as candidates. Finally, 10, 25, 50 and 100 candidate nodes were sent the query according to different settings.

The result lists received were merged considering only the first 100 answers to be presented to the user. As was mentioned earlier, the CombSUM [34] merging algorithm was used with a normalized weighting scheme, as expressed below:

alternative normalization schemes can be found in [18]. Although the fusion of results is an important problem in the field of distributed information retrieval, it is beyond the scope of this work, so a typical method from the literature [5, 33, 34] was used instead.

For each query, an initial node was selected at random, trying to simulate a real situation where users with different internal states in their nodes, i.e. routing tables, execute queries. The execution of the Q106 query set and its evaluation using its corresponding relevance judgments show a difference in the behavior of the different systems. Table 6 summarizes the results whereas Figure 8 represents a Recall/Precision diagram showing the system’s behavior vs. centralized retrieval (the baseline).

Looking at the Recall and Precision values it can be seen that they are lower than expected. This situation is caused by the fact that the P2P network returns a large number of documents since n nodes are consulted and those truly relevant are widely distributed. According to [15], the overall search performance is affected by individual components. In addition, factors such as dynamic topology and uncertainty about the location of relevant documents contribute to the complexity of the problem. With regard to this, the proposed metrics could be somewhat inaccurate. Hence, the analysis was extended to evaluate Precision for different quantities of documents (5, 10, 15, 20, 30 and 100) from an end-user perspective.

Figure 7. Comparison of the traffic generated by Gnutella and our proposed model. SCH: Schollmeier’s study of Gnutella’s traffic, SCH+30%: SCH including dropped messages. Exp#2 and Exp#4: Experimental settings no. 2 and 4 of our model.

Figure 8. Recall/Precision (R-P) diagrams according to different settings. nf10, nf25, nf-50 and nf-100 correspond to our system’s performance when selecting 10, 25, 50 and 100 candidate nodes in the final step of query resolution.
From table 6 we should note that the differences in performance among the experiments cannot be conclusive about the best setting. In half of the cases, increasing the number of candidate nodes consulted did not result in an improvement of precision, especially for the first documents. This suggests that the merging algorithm should be adapted to improve performance, since the users often pay more attention to the first results.

After that, the retrieval experiments were conducted with the P1000 query set, which was distributed in groups of 196, 202, 210, 200 and 192 queries comprising 1, 2, 3, 4 and 5 terms respectively. As was mentioned earlier, the first 100 answers from the centralized system were chosen as relevance judgments. As in the previous experiment, 10, 25, 50 and 100 final nodes were consulted. These results (Tables 7 and 8) show an overall increase in the system’s performance. Particularly, as the number of candidate nodes increases (from 10 to 100) precision is improved. For example, P(0.30) achieves a 12% improvement in performance when consulting 25 nodes (instead of 10) and 21% with 100. These results show the capability of the distributed system to respond in a way

Table 6. Retrieval results using the test collection

<table>
<thead>
<tr>
<th>R</th>
<th>P</th>
<th>R_Prec</th>
<th>Avg_Prec</th>
<th>P@5</th>
<th>P@10</th>
<th>P@15</th>
<th>P@20</th>
<th>P@30</th>
<th>P@100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.3789</td>
<td>0.2885</td>
<td>0.3437</td>
<td>0.2592</td>
<td>0.6579</td>
<td>0.6953</td>
<td>0.6335</td>
<td>0.5783</td>
<td>0.3942</td>
</tr>
<tr>
<td>Exp#5-nf10</td>
<td>0.0166</td>
<td>0.0275</td>
<td>0.0193</td>
<td>0.0099</td>
<td>0.1961</td>
<td>0.1873</td>
<td>0.1490</td>
<td>0.1162</td>
<td>0.0801</td>
</tr>
<tr>
<td>Exp#5-nf25</td>
<td>0.0361</td>
<td>0.0588</td>
<td>0.0397</td>
<td>0.0174</td>
<td>0.2000</td>
<td>0.1902</td>
<td>0.1954</td>
<td>0.1887</td>
<td>0.1614</td>
</tr>
<tr>
<td>Exp#5-nf50</td>
<td>0.0569</td>
<td>0.0909</td>
<td>0.0602</td>
<td>0.0244</td>
<td>0.1942</td>
<td>0.1894</td>
<td>0.1853</td>
<td>0.1817</td>
<td>0.1689</td>
</tr>
<tr>
<td>Exp#5-nf100</td>
<td>0.0673</td>
<td>0.1063</td>
<td>0.0703</td>
<td>0.0272</td>
<td>0.2098</td>
<td>0.2010</td>
<td>0.1941</td>
<td>0.1873</td>
<td>0.1732</td>
</tr>
<tr>
<td>Exp#6-nf10</td>
<td>0.0163</td>
<td>0.0296</td>
<td>0.0185</td>
<td>0.0090</td>
<td>0.1980</td>
<td>0.1939</td>
<td>0.1515</td>
<td>0.1177</td>
<td>0.0815</td>
</tr>
<tr>
<td>Exp#6-nf25</td>
<td>0.0332</td>
<td>0.0559</td>
<td>0.0361</td>
<td>0.0151</td>
<td>0.2078</td>
<td>0.1835</td>
<td>0.1793</td>
<td>0.1733</td>
<td>0.1463</td>
</tr>
<tr>
<td>Exp#6-nf50</td>
<td>0.0520</td>
<td>0.0903</td>
<td>0.0536</td>
<td>0.0214</td>
<td>0.1980</td>
<td>0.2010</td>
<td>0.1987</td>
<td>0.1861</td>
<td>0.1650</td>
</tr>
<tr>
<td>Exp#6-nf100</td>
<td>0.0618</td>
<td>0.1019</td>
<td>0.0628</td>
<td>0.0243</td>
<td>0.2137</td>
<td>0.1980</td>
<td>0.1830</td>
<td>0.1716</td>
<td>0.1526</td>
</tr>
<tr>
<td>Exp#7-nf10</td>
<td>0.0168</td>
<td>0.0323</td>
<td>0.0193</td>
<td>0.0104</td>
<td>0.2218</td>
<td>0.2020</td>
<td>0.1525</td>
<td>0.1178</td>
<td>0.0832</td>
</tr>
<tr>
<td>Exp#7-nf25</td>
<td>0.0341</td>
<td>0.0584</td>
<td>0.0374</td>
<td>0.0172</td>
<td>0.2182</td>
<td>0.2061</td>
<td>0.1926</td>
<td>0.1803</td>
<td>0.1525</td>
</tr>
<tr>
<td>Exp#7-nf50</td>
<td>0.0462</td>
<td>0.0775</td>
<td>0.0489</td>
<td>0.0199</td>
<td>0.2038</td>
<td>0.1819</td>
<td>0.1765</td>
<td>0.1648</td>
<td>0.1425</td>
</tr>
<tr>
<td>Exp#7-nf100</td>
<td>0.0599</td>
<td>0.0997</td>
<td>0.0614</td>
<td>0.0232</td>
<td>0.2040</td>
<td>0.1990</td>
<td>0.1782</td>
<td>0.1683</td>
<td>0.1475</td>
</tr>
<tr>
<td>Exp#8-nf10</td>
<td>0.0171</td>
<td>0.0332</td>
<td>0.0200</td>
<td>0.0105</td>
<td>0.2280</td>
<td>0.2010</td>
<td>0.1587</td>
<td>0.1220</td>
<td>0.0837</td>
</tr>
<tr>
<td>Exp#8-nf25</td>
<td>0.0319</td>
<td>0.0569</td>
<td>0.0345</td>
<td>0.0142</td>
<td>0.1820</td>
<td>0.1960</td>
<td>0.1747</td>
<td>0.1670</td>
<td>0.1460</td>
</tr>
<tr>
<td>Exp#8-nf50</td>
<td>0.0507</td>
<td>0.0876</td>
<td>0.0526</td>
<td>0.0227</td>
<td>0.2079</td>
<td>0.1901</td>
<td>0.1861</td>
<td>0.1743</td>
<td>0.1525</td>
</tr>
<tr>
<td>Exp#8-nf100</td>
<td>0.0553</td>
<td>0.0959</td>
<td>0.0571</td>
<td>0.0206</td>
<td>0.1941</td>
<td>0.1725</td>
<td>0.1601</td>
<td>0.1505</td>
<td>0.1373</td>
</tr>
</tbody>
</table>

Table 7. System performance for P1000 query set

<table>
<thead>
<tr>
<th>R</th>
<th>P</th>
<th>R_Prec</th>
<th>Avg_Prec</th>
<th>P@5</th>
<th>P@10</th>
<th>P@15</th>
<th>P@20</th>
<th>P@30</th>
<th>P@100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp#7-nf10</td>
<td>0.2235</td>
<td>0.9213</td>
<td>0.5110</td>
<td>0.5099</td>
<td>0.6833</td>
<td>0.4861</td>
<td>0.3251</td>
<td>0.2439</td>
<td>0.1627</td>
</tr>
<tr>
<td>Exp#7-nf25</td>
<td>0.3571</td>
<td>0.8723</td>
<td>0.5934</td>
<td>0.5902</td>
<td>0.6842</td>
<td>0.5298</td>
<td>0.4387</td>
<td>0.3727</td>
<td>0.2634</td>
</tr>
<tr>
<td>Exp#7-nf50</td>
<td>0.4761</td>
<td>0.8477</td>
<td>0.6450</td>
<td>0.6393</td>
<td>0.6915</td>
<td>0.5322</td>
<td>0.4369</td>
<td>0.3717</td>
<td>0.2906</td>
</tr>
<tr>
<td>Exp#7-nf100</td>
<td>0.5651</td>
<td>0.7729</td>
<td>0.6682</td>
<td>0.6566</td>
<td>0.6966</td>
<td>0.5393</td>
<td>0.4442</td>
<td>0.3771</td>
<td>0.2949</td>
</tr>
</tbody>
</table>

Table 8. Precision averages at 11 standard recall levels
that is similar to that of the centralized system. Additionally, a traffic study was conducted for each experiment. This time, the formula used in (4) was modified according to the number of index nodes contacted, yielding:

These results (Tables 7 and 8) show an overall increase in the system's performance. Particularly, as the number of candidate nodes increases (from 10 to 100) precision is improved. For example, $P(0.30)$ achieves a 12% improvement in performance when consulting 25 nodes (instead of 10) and 21% with 100. These results show the capability of the distributed system to respond in a way that is similar to that of the centralized system. Additionally, a traffic study was conducted for each experiment. This time, the formula used in (4) was modified according to the number of index nodes contacted, yielding:

$$\text{Ctc} = (tq \cdot \text{Tcci}) + \text{Oria} \cdot (\text{Leti} + \text{Le} + Cq)$$  \hspace{1cm} (5)$$

Where

<table>
<thead>
<tr>
<th></th>
<th>nf10</th>
<th>nf25</th>
<th>nf50</th>
<th>nf100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp#5</td>
<td>80.210</td>
<td>89.245</td>
<td>102.524</td>
<td>122.157</td>
</tr>
<tr>
<td>Exp#6</td>
<td>125.671</td>
<td>132.013</td>
<td>139.397</td>
<td>152.048</td>
</tr>
<tr>
<td>Exp#7</td>
<td>163.770</td>
<td>168.925</td>
<td>175.931</td>
<td>185.661</td>
</tr>
<tr>
<td>Exp#8</td>
<td>206.153</td>
<td>210.428</td>
<td>216.240</td>
<td>224.251</td>
</tr>
</tbody>
</table>

Table 9 summarizes the results of this study. Here, it is easy to see that the model generates an acceptable amount of traffic per query. This is directly compared with a query to a search engine like Google whose answer including only 100 results consumes about 140 kilobytes.

6. Conclusions and Future Work

This work presents a distributed index model built upon a peer-to-peer network which supports complete indexing of text documents and allows searching by content. Its behavior was evaluated by conducting a set of experiments on a simulated 10,000-node network which resembles a mid-sized enterprise network or a community of users with common interests. The results show an improvement in efficiency of between 84% and 93% over similar systems like Gnutella.

The proposed model requires no specific network topology or hierarchy, which is a distinctive feature with regard to other P2P systems for content search. The network topology is defined by direct interactions and the dynamic nature of a P2P environment. As to query expressions, the model supports queries of variable length and the use of incomplete terms. This feature gives a more expressive and flexible search mechanism, suitable to operate on the contents of text documents. Results suggest that the model is a valid alternative to Distributed Hash Tables and other forwarding-based strategies in their attempt to solve the problem of distributed search on P2P systems.

The evaluation of retrieval performance using a test collection showed poor behavior in comparison with a centralized system. However, the experiments with the pseudo-queries set and relevance judgments created from the answers of the centralized system showed acceptable performance. Considering the complexity of the task, the experiments showed how the P2P system was able to reproduce the performance of the centralized system.

Although the initial results allow validating the model, it is necessary to perform tests with larger networks and different settings of nodes, documents and their distribution. Moreover, it is important to continue research in order to improve the understanding of P2P networks. This will encourage the development of new models and algorithms that offer improved functions and performance, in addition to finding optimal parameters for special applications. Different index node selection schemes should be studied as well in order to obtain better results in terms of both traffic and response time.

Acknowledgments

The work presented in this paper was developed as part of a research fellowship granted by Secretaría de Investigación y Postgrado at Universidad Nacional de Luján, Argentina.

We would like to give special thanks to Mariano Felice who helped us to translate and check the manuscripts of this article.

References
