

# Design and Implementation of a Semantic Peer-to-Peer Network

Kiyohide Nakauchi<sup>1</sup>, Hiroyuki Morikawa<sup>2</sup>, and Tomonori Aoyama<sup>3</sup>

<sup>1</sup> National Institute of Information and Communications Technology,  
4-2-1, Nukui-kitamachi, Koganei, Tokyo, 184-8795, Japan  
`nakauchi@nict.go.jp`

<sup>2</sup> School of Frontier Sciences, The University of Tokyo,  
7-3-1, Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan  
`mori@mlab.t.u-tokyo.ac.jp`

<sup>3</sup> School of Information Science and Technology, The University of Tokyo,  
7-3-1, Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan  
`aoyama@mlab.t.u-tokyo.ac.jp`

**Abstract.** Decentralized and unstructured peer-to-peer (P2P) networks such as Gnutella are attractive for large-scale information retrieval and search systems due to scalability, fault-tolerance, and self-organizing nature. This decentralized architecture, however, makes it difficult for traditional P2P networks to globally share useful semantic knowledge among nodes. As a result, traditional P2P networks cannot support semantic search (support only naive text-match search). In this paper, we describe one possible design of a semantic P2P network which enables semantic keyword search. We exploit the semantics of correlation among keywords rather than synonym. The key mechanism is *query expansion*, where a received query is expanded based on *keyword relationships*. Keyword relationships are improved through search and retrieval processes and each relationship is shared among nodes holding similar data items. Our main challenges are 1) managing keyword relationships in a fully decentralized manner and 2) maintaining the quality of search results, while suppressing result implosion. We also describe the prototype implementation and simple evaluation of the semantic P2P network.

## 1 Introduction

Peer-to-peer (P2P) networks are now one of the most prevalent Internet distributed applications because of scalability, fault-tolerance, and self-organizing nature. The primary focus of the decentralized architectures is compatibility of scalability and partial-match lookup (keyword search) capability. There are two classes of the decentralized architectures to achieve the compatibility. One class of decentralized architectures is an unstructured P2P system such as Gnutella [1], where the overlay topology is formed in accordance with some loose rules [2]. These query flooding-based P2P networks are simple and robust, but mainly considered not to be scalable. Recently some efforts [3–8] are made to improve

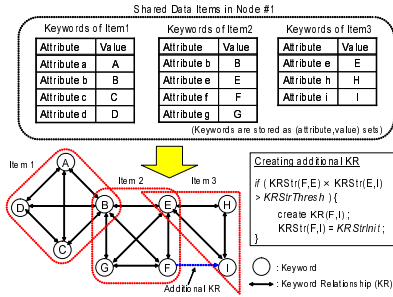
scalability. The other class of decentralized architectures is a structured P2P system commonly referred to as Distributed Hash Tables (DHTs) [9–12], where the overlay topology is tightly controlled and queries are deterministically routed. While structured P2P systems are highly scalable for exact-match lookups, inherently they cannot efficiently provide a partial-match lookup capability. Some work [13–15] have been aimed at providing a partial-match lookup capability on DHTs through a sophisticated method of generating keys (globally unique identifiers or GUIDs) corresponding to data from the attached multiple keywords.

As described above, traditional P2P networks are actively improved to achieve compatibility of scalability and partial-match lookup capability. Beyond these discussions towards developing novel classes of large-scale distributed services with more flexibility and user-friendliness, we believe semantic keyword search functionality is fundamental. However, traditional P2P networks are not capable of semantic keyword search, and consequently can find only data items with a keyword (or meta-data) exactly indicated in a query.

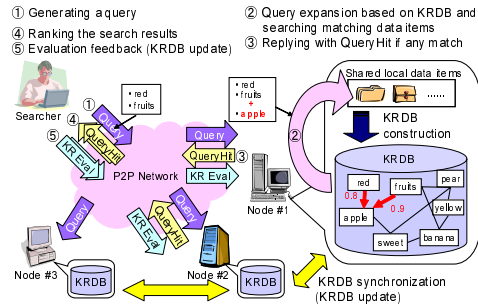
In this paper, we show one possible design of a semantic P2P network. Our goal is to build an efficient decentralized P2P network that supports semantic search, while retaining desirable properties of prevailing unstructured P2P networks such as simplicity and robustness. In order to leverage the properties of simplicity and robustness, we design a semantic search framework on the unstructured P2P networks. We exploit semantics by introducing the concept of *keyword relationships*, with which *query expansion*[16, 17], a general approach to semantic search in centralized search systems, can be applied. In a semantic P2P network, not only well-defined items, which are given keywords easy to imagine for searchers and can be located by traditional partial-match P2P search systems, but also poorly-defined items, which are given keywords generally or unexpectedly difficult to image for searchers and cannot be located by traditional ones, can be located efficiently.

We faced two main challenges when designing our semantic P2P keyword search system. The first challenge is to construct and manage the databases required for query expansion in a fully decentralized manner. In P2P networks, because of the decentralized nature, it is undesirable to have a centralized node calculate and maintain the statistics to obtain keyword relationships. The second is to maintain the quality of search results while suppressing "result implosion" in the worst case, which is the explosive increase of returned search results due to query expansion. To cope with this problem, we introduce a results ranking mechanism where data items with more keywords relevant to a query are ranked higher.

Before we proceed, we would like to emphasize the following points. First, this paper is not concerned with how to improve the quality of search results to the comparable degree of state-of-the-art IR algorithms developed for centralized search engines such as Google. Recall our concern is improvement of the possibility of locating poorly-defined data items. Second, we use the word "semantics" to define correlation rather than synonymy between any two key-



**Fig. 1.** Creating Keyword Relationships from Local Data Items



**Fig. 2.** Basic P2P Search Mechanism

words. Therefore two correlative keywords do not necessarily have synonymy. Our semantic P2P network does not cope with misspelling.

The reminder of this paper is organized as follows. In Section 2, we give an overview and describe the basic mechanisms of semantic P2P search with query expansion based on keyword relationships. Section 3 describes the distributed update mechanisms of KRDBs to enhance search performance. We describe the implementation and simple evaluation of the semantic P2P network in Section 4 and in Section 5, respectively. We conclude in Section 6.

## 2 Semantic P2P Search

In this section, we explain how to create a KRDB and then describe the basic search mechanism. We also describe the ranking algorithm.

In the semantic P2P network, as in a traditional unstructured P2P system, the overlay topology is organized in accordance with some loose rules [2]. A query includes several keywords and is flooded to be resolved.

### 2.1 KRDB

A KRDB is a thesaurus which keeps some information about the keywords relevant only to the data items stored locally in a node. That means each node may have a different and minimum KRDB. This distributed KRDB management can clearly retain the desirable properties of P2P systems.

The most important information on keywords in a KRDB is the *keyword relationship* (KR) of each pair of keywords and its strength. In this paper, we refer to the keyword relationship itself from keyword  $k_i$  to keyword  $k_j$  as  $KR(k_i, k_j)$  ( $1 \leq i, j \leq n$ , where  $n$  denotes the maximum number of keywords in a KRDB). In other words,  $KR(k_i, k_j)$  is defined as follows; when keyword  $k_i$  is given, keyword  $k_j$  is referred to as a relevant term to keyword  $k_i$ . Note that  $KR(k_i, k_j)$  and  $KR(k_j, k_i)$  should be distinguished from each other. The other variables in a KRDB are shown in Section 3.

Figure 1 shows how to initially create KRs between keywords. There are two processes to create KRs. First, when a node joins the P2P network, the node firstly extracts all the keywords for each local data item. For example, the node takes out four keywords A, B, C, D from data item 1. We consider these keywords have relationships between each other. Each  $KR(k_i, k_j)$  keeps  $KRStr(k_i, k_j)$ , which denotes the normalized variable of strength of  $KR(k_i, k_j)$  ( $0 \leq KRStr(k_i, k_j) \leq 1$ ). Larger  $KRStr(k_i, k_j)$  means  $KR(k_i, k_j)$  is stronger.  $KRStr$  is updated to reflect more accurate KR based on both evaluation feedback and KRDB synchronization described in Section 3, and is used for results ranking. The initial value of  $KRStr$  is  $KRStrInit$  (0.5 in our system).

To further sophisticate a KRDB, additional KRs are created if two KRs share the same keyword. For example, as shown in Figure 1, if the value of  $KRStr(F, E) \times KRStr(E, I)$  is larger than the pre-defined threshold  $KRStrThresh$ ,  $KR(F, I)$  is newly created. However, less useful KRs are removed from KRDBs to prevent the waste of storages and computation power.

These two KR creation processes are called when local data items are modified or newly added, or when a local KRDB is improved through KRDB updates mentioned in Section 3.2.

## 2.2 Basic Search Mechanism

The key mechanism of our P2P keyword search system, which differentiates our system from traditional unstructured P2P systems, is query expansion at nodes that receive the query.

Figure 2 shows the basic search mechanism using query expansion. When a node join the P2P network, the node first constructs a KRDB in the way described in Section 2.1. The search process is as follows.

1. A searcher issues a query which indicates several keywords. This query is flooded (forwarded in a P2P manner like Gnutella) with a certain TTL (Time To Live). Note that the forwarded query is identical with the received query (query expansion affects only local search), because consecutive query expansion at different nodes leads to query explosion with less relevant keywords, so that the possibility of finding less desired data items increases. (Process ① in Fig. 2)
2. A node which receives a query performs query expansion using its local KRDB. Specifically, the original query is expanded to include several additional keywords to which there are KRs from keywords in the original query. For example, in Figure 2, Node #1 keeps  $KR(\text{red}, \text{apple})$  in its local KRDB (with  $KRStr(\text{red}, \text{apple}) = 0.6$ ) so that a keyword "red" is expanded to two keywords, "red" and "apple". In the same way, keyword "fruits" is expanded to two keywords, "fruits" and "apple". Then all the expanded keywords for each original keyword are merged. After all, the expanded query includes three keywords, "red", "fruits", and "apple", with which Node #1 searches for local data items. (Process ②)

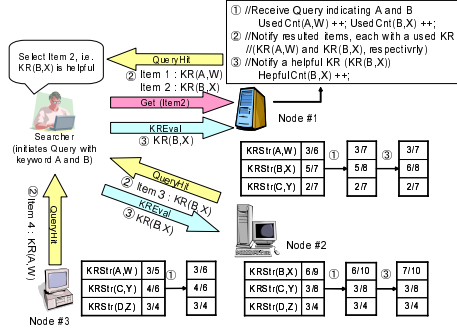
3. If a data item exists that has one or more keywords of the expanded query, Node #1 replies to the searcher with search results (a list of satisfied data items) using a QueryHit message. (Process ③)
4. The searcher gathers search results and ranks all the located data items. The ranking algorithm is described in Section 2.3. The searcher then selects one or more desirable data items, and the search itself is completed. (Process ④)
5. At the same time, the searcher feedbacks the evaluation to all the nodes which returned search results obtained using  $KR(\text{red}, \text{apple})$  or  $KR(\text{fruits}, \text{apple})$  (Node #1 and #2 in Figure 2) for the purpose of updating KRDBs in those nodes. Evaluation results indicate which KRs were used to locate the selected data items. The details of evaluation feedback are described in 3.1. (Process ⑤)

### 2.3 Results Ranking

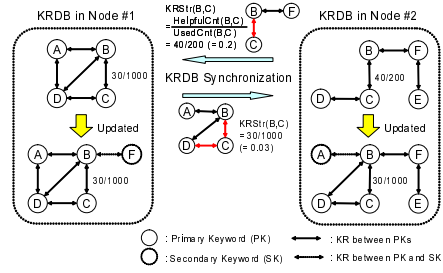
The search results are ranked based on KR strength between a keyword in an original query and that given to the located data item. The basic idea of results ranking is as follows; When an original query includes keyword  $k$ , the rank of the data item with keyword  $l$  gets higher as the value of  $KRStr(k, l)$  in the receiver's KRDB is larger. If several keywords are included in a query, or are given to data items, the above ranking algorithm is changed as follows; when an original query includes keywords  $k_i$  ( $i = 1, 2, \dots$ ), the rank of the located data item gets higher as simply  $\sum_{i, i'} KRStr(k_i, l_{i'})$  is larger, where  $l_{i'}$  ( $i' = 1, 2, \dots$ ) denote keywords given to the data item. Note that other sophisticated ranking algorithms using  $KRStr$  could be applied. We believe these ranking algorithms would also benefit traditional P2P search systems, but only for the purpose of results ranking by introducing KRDB-like databases otherwise unnoticeable for users into the search systems. Such an application, though, is beyond the scope of this paper.

## 3 Distributed KRDB Updates

In our P2P keyword search system, the accuracy of KRDBs significantly affects the search performance. Therefore KRDBs are required to be updated and kept as accurate as possible. In this paper, we show two distributed KRDB update mechanisms; evaluation feedback and KRDB synchronization. Evaluation feedback is aimed at improving KRDBs through a search process. In this mechanism, the subjective evaluations of searchers are directly reflected in the KRDBs, and potential statistical effects can be expected. KRDB synchronization is aimed at improving KRDBs that include inaccurate information due to special circumstances: for example, when a node has just joined the P2P network, or a node has just exposed many new data items. These two update mechanisms are complementary to each other and are essential for keeping KRDBs accurate.



**Fig. 3.** KRDB Update through Evaluation Feedback



**Fig. 4.** KRDB Synchronization

### 3.1 Evaluation Feedback

The evaluation feedback updates  $KRStr(k_i, k_j)$  in the nodes which replied to the searcher with resulted data items found using  $KR(k_i, k_j)$ . The basic idea of evaluation feedback is as follows. When a searcher initiates a query with keyword  $k_i$  and then selects a data item with keyword  $k_j$  from the resulted item list (the item was located by query expansion using  $KR(k_i, k_j)$ ),  $KR(k_i, k_j)$  is regarded as helpful and  $KRStr(k_i, k_j)$  is increased. Otherwise,  $KRStr(k_i, k_j)$  is decreased.

We use two variables,  $UsedCnt(k_i, k_j)$  and  $HelpfulCnt(k_i, k_j)$ .  $UsedCnt(k_i, k_j)$  is incremented when both of the following conditions are satisfied; 1) the original query including keyword  $k_i$  is expanded using  $KR(k_i, k_j)$  and 2) a data item with keyword  $k_j$  is located. However,  $HelpfulCnt(k_i, k_j)$  is incremented only when a third condition is also satisfied; 3) a data item with keyword  $k_j$  is selected by a searcher. We define  $KRStr(k_i, k_j)$  using these two variables;

$$KRStr(k_i, k_j) = \frac{HelpfulCnt(k_i, k_j)}{UsedCnt(k_i, k_j)}$$

This means that  $KRStr(k_i, k_j)$  increases as more searchers regards  $KR(k_i, k_j)$  as useful one ( $HelpfulCnt(k_i, k_j)$  increases). Currently, the initial values of each variable are set as follows:  $UsedCntInit = 2$ ,  $HelpfulCntInit = 1$ ,  $KRStrInit = 1/2 = 0.5$ .

Figure 3 shows the process of evaluation feedback. In this figure, a searcher initiates a query that includes keywords A and B.

1. Each node receives a query with keyword  $k_i$  from a searcher. At this time, a node which keeps  $KR(k_i, *)$  ( $*$  denotes an arbitrary keyword) increments  $UsedCnt(k_i, *)$ . In Figure 3,  $UsedCnt(A, W)$  and  $UsedCnt(B, X)$  are incremented at Nodes #1 and #3, and at Nodes #1 and #2 respectively. More specifically, for example, Node #1 initially has  $KRStr(A, W) = 3/6 (= 0.5)$ , which means  $UsedCnt(A, W) = 6$  and  $HelpfulCnt(A, W) = 3$ . At Node #1,  $UsedCnt(A, W)$  and  $UsedCnt(B, X)$  are then incremented from 6 to 7 and from 7 to 8, respectively.

2. Each node notifies the searcher of search results with  $KR(k_i, *)$  used for locating the data item. In Figure 3, for example, Node #1 notifies the searcher of Item 1 with  $KR(A, W)$  and Item 2 with  $KR(B, X)$ .
3. When a data item without keyword  $k_i$  (with keyword  $k_j$ ) is selected by the searcher among search results, the searcher notifies all the responsive nodes by unicast who keep the same  $KR$  ( $KR(k_i, k_j)$ ) used to locate the data item. At this time, the node that receive this evaluation feedback increments  $HelpfulCnt(k_i, k_j)$  of the  $KRs$  specified by the feedback.

In Figure 3, the searcher selects data item 2 among search results. Data item 2 is found using  $KR(B, X)$  at Node #1, so the searcher refers to  $KR(B, X)$  as a helpful relationship. The searcher then sends evaluation feedback (using a  $KREval$  message) to Nodes #1 and #2, who return the data item found using  $KR(B, X)$ . When Nodes #1 and #2 receive evaluation feedback, they increment  $HelpfulCnt(B, X)$  from 5 to 6, and from 6 to 7 respectively. Note that  $HelpfulCnt(A, W)$  retains the same value because  $KR(A, W)$  is useless for the searcher.

Through this evaluation feedback process of all the searchers, each  $KRStr$  in  $KRDBs$  is gradually and statistically refined.

### 3.2 KRDB Synchronization

We propose another  $KRDB$  update mechanism,  $KRDB$  synchronization, where familiar  $KRs$  and a statistically more accurate value of  $KRStr$  are shared among nodes. The basic idea of  $KRDB$  synchronization is as follows; 1)  $KRs$  relevant to a node are added to the node's  $KRDB$ , and 2) when the same  $KRs$  are shared at some nodes, the value of  $KRStr$  at each node is updated to the most accurate value.

**Which Nodes Are Selected for  $KRDB$  Synchronization ?** It is desirable for  $KRDBs$  to be synchronized with only a limited number of nodes for scalability, while keeping  $KRDBs$  as accurate as possible. Here, we refer to target nodes of synchronization as well-matched nodes. We consider well-matched nodes to possibly be nodes which hold as many identical or similar data items as possible, because consequently they are likely to keep more identical  $KRs$ .

To make it easy to find well-matched nodes, we focus on keywords as abstraction of data items. Here, the keywords that can be extracted only from local data items are called Primary Keywords (PKs) to distinguish them from additional keywords (Secondary Keywords, or SKs), which are added through  $KRDB$  synchronization. In this paper, each node selects the  $N$  best-matched nodes to synchronize with that have the largest number of the shared PKs. Every node periodically searches better-matched nodes and refines the node set for synchronization it keeps. In this paper, the way to discover well-matched nodes is simply broadcasting.

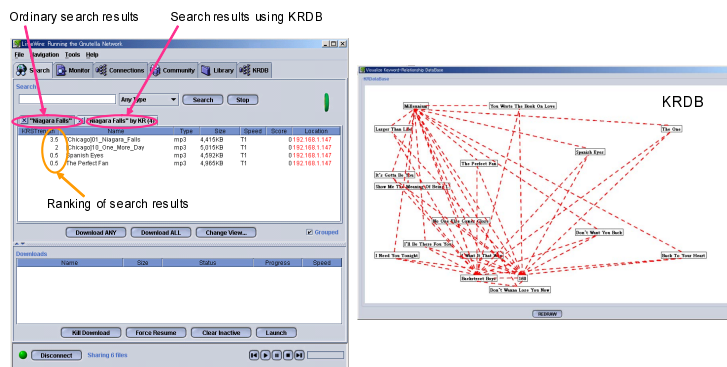


Fig. 5. Implementation on LimeWire

**Synchronization Mechanisms** Subscribing nodes periodically synchronize their KRDBs with well-matched nodes. Figure 4 shows an overview of the KRDB synchronization mechanism after a node discovers the  $N$  best-matched nodes for KRDB synchronization in the way described above. The KRDB synchronization consists of two mechanisms. The first mechanism is KR addition, where a KR between two PKs or a KR between a PK and a SK kept in one node is added to the KRDB in the other node only when one of the two PKs or the PK is shared at those two nodes respectively. For example, in Figure 4, the KRDB in Node #1 is updated by adding two KRs concerning PK B, KR(B, F) and KR(F, B), which are extracted from the KRDB in Node #2, because there exists a shared PK B.

The second is KRStr modification where KRStr in one node is updated to that in the other node, which would be statistically more accurate, when KRs between two PKs are shared at those two nodes. Here, we consider the accuracy of  $KRStr(k_i, k_j) (= \frac{HelpfulCnt(k_i, k_j)}{UsedCnt(k_i, k_j)})$  statistically increases as  $UsedCnt(k_i, k_j)$  increases. For example, in Figure 4,  $KRStr(B, C)$  in Node #2 is updated from  $40/200 (= 0.2)$  to  $30/1000 (= 0.03)$  because  $UsedCnt(B, C)$  in Node #1 ( $= 1000$ ) is larger than that in Node #2 ( $= 200$ ). Thus each KR radiates from the node that keeps the PK with the largest  $UsedCnt$  through KRDB synchronization.

## 4 Implementation

We are now developing a prototype semantic P2P search system with query expansion. We have implemented the search algorithms described in Sections 2 and 3 on LimeWire [18], which is a well known open-source P2P keyword search application. LimeWire is written in Java.

Figure 5 is a screenshot of the prototype (LimeWire with query expansion). The upper window shows the ranked search results. To make clear the difference between the search results of a normal search and those with query expansion,

and to help performance evaluation and further development, the prototype simultaneously executes a normal search and a search with query expansion. Note that because a common query routing algorithm is used for these two search mechanisms, no additional traffic is generated. We made it possible to switch the display to show the search results of these two mechanisms. The lower window shows the visualized KRDB. The dotted line denotes the KRs and quadrangles denote keywords.

## 5 Evaluation

In this section, we evaluate the semantic P2P network through simple analysis of experimental results.

### 5.1 Metric

We propose a novel metric *metadata correctness*  $P_i$  for content  $i$ , which shows the ratio of data items with content  $i$  which are correctly attached all the desired metadata related to content  $i$  to all data items with content  $i$ . In these experiments, we focus on music content, and we define or identify content only by both "artist name" and "song title". Accordingly we identify a data item only by its content and do not use other metadata such as file name, encoding parameters, and file size for this purpose.  $P_i$  is defined as follows:

$$P_i = \frac{N_i^{Org}}{N_i^{Sem}}$$

where  $N_i^{Org}$  and  $N_i^{Sem}$  denotes the number of data items which traditional P2P search systems and our one can absolutely find, respectively, when *any* one of desired keywords for content  $i$  is used.

We consider  $P_i$  is useful for evaluating the quantitative superiority of our system over traditional ones when a searcher desires data items with content  $i$ , because  $P_i$  affects *success rate* of keyword search for traditional systems. Here, success rate is defined as the ratio of target data items which can be found by keyword search to all the target data items actually existing in the limited area a query can reach. When one of desired keyword for content  $i$  is used as a query in traditional systems, smaller  $P_i$  results in lower success rate because the possibility that the metadata is attached to the target data items is lower. On the other hand, success rate of our semantic P2P keyword search is always 1.0, because one of desired keywords is expanded to the other desired ones.

### 5.2 Overview of Experiments

We calculate  $P_i$  by analyzing metadata information used for a real P2P file sharing application. We use a free OpenNap client software to obtain metadata

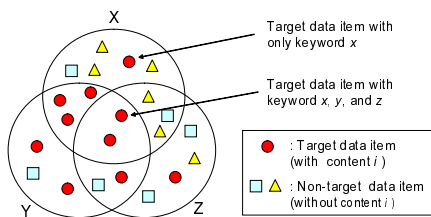


Fig. 6. An Example of Search Results

Table 1. Analysis of Experimental Search Results for Content  $i$

	Exp. #1	Exp. #2	Exp. #3
$N_i(X_i \cap Y_i \cap Z_i)$	12	11	12
$N_i(X_i \cap Y_i)$	45	38	56
$N_i(Y_i \cap Z_i)$	8	6	9
$N_i(Z_i \cap X_i)$	1	2	2
$N_i(X_i \cap \bar{Y}_i \cap \bar{Z}_i)$	0	0	0
$N_i(\bar{X}_i \cap Y_i \cap \bar{Z}_i)$	5	8	3
$N_i(\bar{X}_i \cap \bar{Y}_i \cap Z_i)$	2	6	7
$P_i$	0.16	0.16	0.14
$P_i(x)$	0.80	0.72	0.79
$P_i(y)$	0.96	0.89	0.90
$P_i(z)$	0.32	0.35	0.34

information from OpenNap servers, which have the same functionalities as Napster index servers. Though our system is designed based on Gnutella, we believe shared data items in P2P file sharing applications are similar.

Among metadata directly obtained from OpenNap servers, we use only "artist name" and "song title". We extract another kind of metadata except them from file name, because there is no rule to name data items. For example, we can find some data items with file name "artist name - song title - movie title.mp3". As a result, at most three keywords can be derived from each data item.

We decide a specific keyword  $x$  used for keyword search and obtain search results from an OpenNap server, from which all metadata are extracted for constructing a *global* KRDB. We refer to this global KRDB as a local KRDB in a node in our experiments. Note that even in distributed environments, each node can construct a part of the global KRDB through KRDB synchronization, so that we believe the same results can be obtained in such environments. From this global KRDB, we select another two keywords  $y$  and  $z$  that have a KR to  $x$ . We calculate  $P_i$  by investigating how many target data items keep all these three keyword.

We did this experiment three times at intervals of at least one day. The number of online users and that of data items were about 3,500 - 5,000 and 2,100,000 - 2,500,000, respectively.

### 5.3 Methodology

We describe the methodology using an example of search results shown in Figure 6. The methodology is as follows. Firstly, we decide an arbitrary keyword  $x$ , and then obtain a search result (a list of data items) from an OpenNap server for keyword  $x$ . We refer to the set of data items as  $X$ . From the data items, we extract metadata and construct a global KRDB. We select keyword  $y$  and  $z$  that have a KR to  $x$ . Then, we obtain search results from an OpenNap server for keyword  $y$  and  $z$ , respectively. (Correspondingly, the sets of data items are referred to  $Y$  and  $Z$ , respectively.) After that, we select a target data item which keeps all the keyword  $x$ ,  $y$ , and  $z$ , and define its content as  $i$ . For example, in

Figure 6, data items represented by a circle are selected because one of them exists in the area  $X \cap Y \cap Z$ .

We calculate  $P_i$  using the following formula converted from the one defined in Section 5.1

$$P_i = \frac{N_i(X_i \cap Y_i \cap Z_i)}{N_i(X_i \cup Y_i \cup Z_i)}$$

where  $X_i$ ,  $Y_i$ , and  $Z_i$  represent a set of target data items with content  $i$  in  $X$ ,  $Y$ , and  $Z$ , respectively, and  $N_i(A)$  denotes the number of target data items with content  $i$  in the set  $A$ . As  $N_i(X_i \cup Y_i \cup Z_i)$  can be converted to  $N_i(X_i \cap Y_i \cap Z_i) + N_i(X_i \cap Y_i) + N_i(Y_i \cap Z_i) + N_i(Z_i \cap X_i) + N_i(X_i \cap \bar{Y}_i \cap \bar{Z}_i) + N_i(\bar{X}_i \cap Y_i \cap \bar{Z}_i) + N_i(\bar{X}_i \cap \bar{Y}_i \cap Z_i)$ ,  $P_i$  can be calculated by counting each of these items. For example, in Figure 6, we can count each item as follows:  $N_i(X_i \cap Y_i \cap Z_i) = 2$ ,  $N_i(X_i \cap Y_i) = 3$ ,  $N_i(Y_i \cap Z_i) = 1$ ,  $N_i(Z_i \cap X_i) = 0$ ,  $N_i(X_i \cap \bar{Y}_i \cap \bar{Z}_i) = 1$ ,  $N_i(\bar{X}_i \cap Y_i \cap \bar{Z}_i) = 2$ ,  $N_i(\bar{X}_i \cap \bar{Y}_i \cap Z_i) = 1$ . As a result,  $P_i$  is calculated as follows:  $P_i = \frac{2}{2+3+1+0+1+2+1} = \frac{2}{10} = 0.2$ .

We can derive another metric *metadata correctness for x*, which represents the ratio of data items with content  $i$  which are attached keyword  $x$  to all data items with content  $i$ .  $P_i(x)$  can be calculated as follows:  $P_i(x) = \frac{N_i(X_i)}{N_i(X_i \cup Y_i \cup Z_i)} = \frac{6}{10} = 0.6$ . In the same way, we can calculate  $P_i(y)$  and  $P_i(z)$  as follows:  $P_i(y) = 0.8$ ,  $P_i(z) = 0.4$ . These results show that our semantic P2P search system performs best in comparison with traditional ones when a searcher issues a query with keyword  $z$ .

#### 5.4 Analysis

Table 5.3 shows the analytic results of individual experiments. In our experiments, we select the keywords "Celine Dion", "My Heart Will Go On", and "Titanic" as  $x$ ,  $y$ , and  $z$ , respectively. In Table 5.3,  $P_i$  is at most 0.16. This means that only 16% of overall target data items are given all the three desired keywords ( $x$ ,  $y$ , and  $z$ ). In other words, the percentage of target data items that can be found in traditional P2P search systems when any one of the three keywords is used is only 16%. On the other hand, the semantic P2P search system can find all the target data items if they have at least one desired keyword. Table 5.3 also shows that  $P_i(z)$  is smallest among  $P_i(x)$ ,  $P_i(y)$ , and  $P_i(z)$ , and our system performs best in comparison with traditional ones when keyword  $z$  is used as a query.

Note that metadata correctness ( $P_i$ ,  $P_i(x)$ ,  $P_i(y)$ , and  $P_i(z)$ ) depends on target content, and the analytic results in this section do not show quantitative characteristics but show only qualitative ones. In our assumed environments where a specific naming standard for data item itself and for its metadata is not defined, the above qualitative characteristics would hold true for other content.

## 6 Conclusion

In this paper, we have described a basic concept and a design of an efficient decentralized P2P search system that supports semantic search through query

expansion while retaining the desirable properties of traditional unstructured P2P networks (e.g. simplicity and robustness). In the semantic P2P network, queries are expanded based on KRDBs to improve the possibility of locating a poorly-defined desired data item. We proposed a results ranking mechanism to cope with any consequent results implosion. To improve the KRDBs and enhance search performance, we proposed two KRDB update mechanisms: evaluation feedback and KRDB synchronization. Then, we analyzed experimental search results to evaluate our system, and showed qualitative superiority of the semantic P2P network over traditional one.

## References

1. Gnutella. <http://gnutella.wego.com/>.
2. Clip2 Distributed Search Services. The Gnutella Protocol Specification v0.4, 2000. [http://www9.limewire.com/developer/gnutella\\_protocol\\_0.4.pdf](http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf).
3. FastTrack. <http://www.fasttrack.nu/>.
4. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. Proc. ACM ICS 2002, June 2002.
5. Edith Cohen, Amos Fiat, and Haim Kaplan. Associative Search in Peer to Peer Networks: Harnessing Latent Semantics. Proc. IEEE INFOCOM 2003, Apr. 2003.
6. K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. Proc. IEEE INFOCOM 2003, Apr. 2003.
7. P. Ganesan, Q. Sun, and H. Garcia-Molina. YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology. Proc. IEEE INFOCOM 2003, Apr. 2003.
8. Yatin Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. Proc. ACM SIGCOMM 2003, Aug. 2003.
9. B. Zhao, J. Kubiawicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report, UCB/CSD-01-1141, April 2000.
10. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. Proc. ACM SIGCOMM 2001, Aug. 2001.
11. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. Proc. ACM SIGCOMM 2001, Aug. 2001.
12. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Proc. Middleware 2001, Nov. 2001.
13. M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. Proc. IPTPS 2002, Mar. 2002.
14. C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. Proc. ACM SIGCOMM 2003, Aug. 2003.
15. P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. Proc. Middleware 2003, June 2003.
16. M. Mitra, A. Singhal, and C. Buckley. Improving Automatic Query Expansion. Proc. ACM SIGIR'98, Aug. 1998.
17. W. Hersh, S. Price, and L. Donohoe. Assessing thesaurus-based query expansion using the UMLS Metathesaurus. Proc. the 2000 Annual AMIA Fall Symposium, 2000.
18. LimeWire. <http://www.limewire.com/>.