Huang JC, Li XQ, Wu J. A semantic searching scheme in heterogeneous unstructured P2P networks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 26(6): 925–941 Nov. 2011. DOI 10.1007/s11390-011-1190-z

A Semantic Searching Scheme in Heterogeneous Unstructured P2P Networks

Jun-Cheng Huang[1] (黄俊成), *Member*, *ACM*, *IEEE*, Xiu-Qi Li[2] (李秀琦), *Member*, *ACM*, *IEEE* and Jie Wu[3] (吴　杰), *Member*, *ACM*, *Fellow*, *IEEE*

[1] *Shanghai Hewlett-Packard Co., Ltd., No. 889 Yishan Road Xuhui, Shanghai 201206, China*

[2] *Department of Computer Science and Mathematics, University of North Carolina at Pembroke, Pembroke, U.S.A.*

[3] *Department of Computer and Information Sciences, Temple University, Philadelphia, U.S.A.*

E-mail: juncheng.huang@hp.com; xiuqi.li@uncp.edu; jiewu@temple.edu

Received May 16, 2010; revised May 12, 2011.

Abstract Semantic-based searching in peer-to-peer (P2P) networks has drawn significant attention recently. A number of semantic searching schemes, such as GES proposed by Zhu Y *et al.*, employ search models in Information Retrieval (IR). All these IR-based schemes use one vector to summarize semantic contents of all documents on a single node. For example, GES derives a *node vector* based on the IR model: VSM (Vector Space Model). A topology adaptation algorithm and a search protocol are then designed according to the similarity between node vectors of different nodes. Although the single semantic vector is suitable when the distribution of documents in each node is uniform, it may not be efficient when the distribution is diverse. When there are many categories of documents at each node, the node vector representation may be inaccurate. We extend the idea of GES and present a new class-based semantic searching scheme (CSS) specifically designed for unstructured P2P networks with heterogeneous single-node document collection. It makes use of a state-of-the-art data clustering algorithm, online spherical k-means clustering (OSKM), to cluster all documents on a node into several classes. Each class can be viewed as a virtual node. Virtual nodes are connected through virtual links. As a result, the class vector replaces the node vector and plays an important role in the class-based topology adaptation and search process. This makes CSS very efficient. Our simulation using the IR benchmark TREC collection demonstrates that CSS outperforms GES in terms of higher recall, higher precision, and lower search cost.

Keywords class-based search, GES, semantic clustering, topology adaptation, P2P networks

1 Introduction

Peer-to-peer (P2P) networks can be classified into three categories according to the control over data location and network topology: highly structured, loosely structured, and unstructured[1]. Highly structured P2P systems, such as CAN[2], Pastry[3], Chord[4], and their extensions[5-6], have tight control over data location and topology. They provide bounded data lookup efficiency and the guarantee in finding existing data. They also support exact-match lookups well. However, they suffer from high overhead caused by frequent node join/leave, known as "churn". In addition, these systems are not amenable to full-text semantic search though some recent studies[7-9] try to address this issue. In loosely structured P2P systems, such as Freenet[10] and Symphony[11], the overlay structure and data locations are not precisely determined. The overlay gradually evolves into some intended structure. In unstructured P2P systems, such as Gnutella[12], both the overlay and data locations are arbitrary. Therefore, unstructured P2P systems provide better support for complex queries like keyword/full text semantic search. In this paper, we study unstructured P2P systems.

Searching techniques in unstructured P2P networks are either blind or informed, as shown in Fig.1. Informed searches utilize hints in forwarding queries while blind ones forward queries without hints. Random walk[13], iterative deepening[14], and k-walker random walk[13] are blind. Routing indices[15] and directed BFS[14] are informed. Another taxonomy of searches in unstructured P2Ps is semantic or non-semantic. Semantic searches locate documents that have similar semantic content. For example, documents about basketball have similar semantic content. A non-semantic search does not consider semantic information. For example,

Regular Paper

This work was supported in part by the National Science Foundation of USA under Grant Nos. ANI 0073736, EIA 0130806, CCR 0329741, CNS 0422762, CNS 0434533, CNS 0531410, CNS 0626240, CCF 0830289, and CNS 0948184.

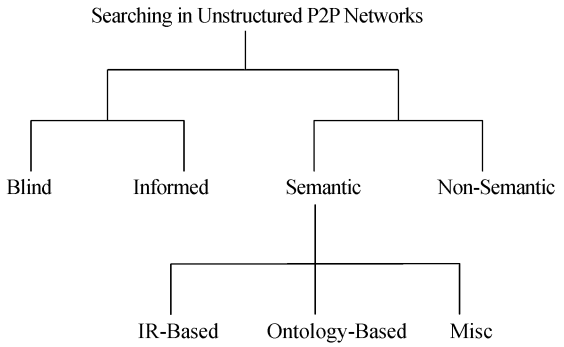

Fig.1. The classification of searching in unstructured P2P networks.

finding a file with ID 500 does not need any semantic information.

Semantic searches can be further classified as IR-based, ontology-based, and misc, as shown in Fig.1. IR-based searches adapt classical models in Information Retrieval (IR), such as VSM (vector space model) or LSI (latent semantic indexing), to P2P networks. For example, GES[16] utilizes the IR model VSM. Ontology-based searches, such as SenPeer[17], try to apply ontology techniques in Semantic Web to P2P networks. The other semantic searches are neither IR-based nor ontology-based. They do not employ common techniques. They are classified as misc (miscellaneous). For example, the work in [18] adapts techniques in content-based multimedia retrieval to P2P networks. Multimedia features such as color and texture are explored.

Ontology techniques are supposed to improve classical IR models. However, ontology-based searching schemes introduce the new challenging issue of ontology mapping. Different ontologies for the same domain may exist in distributed systems due to different perspectives of people with different cultural background, education, etc. Some schemes like [19] assume the same ontology among all peers and do not consider ontology mapping. Others like Klink+[20] and SenPeer[17] address this issue with limitation. Klink+ requires either manual or semi-automatic annotation. The semi-automatic solution still relies on classical IR approach TF/IDF. The mapping scheme in SenPeer employs the appearance similarity between words. However, words like "power" and "tower" may look similar, but do not have similar meanings.

In this work, we investigate IR-based approaches and full-text content search. In a full-text content search, queries may be in the form of a sentence, a paragraph, or even an entire document. The search techniques locate documents that have similar semantic content to queries by looking inside the document texts. IR techniques are designed for full-text content search. A number of IR-based approaches[16,21-26] have been proposed for full-text content search in unstructured P2P networks. To the best of our knowledge, no existing IR-based searches are designed specifically for the scenario where documents on a single node are heterogeneous and may be in different categories. For example, files on a node may be in the music class, literature class, and sports class. Such a node should be assigned to multiple categories. However, all existing schemes use one category vector to represent the heterogenous document collection on a single node. This one-vector representation is inaccurate.

In this paper, we present a distributed, dynamic, and IR-based semantic searching scheme named CSS, which is designed for unstructured P2P networks with heterogeneous documents on a single node. It is an extension of GES[16], a distributed IR-based semantic search. GES summarizes all the documents on each node into an average term vector (named node vector) based on VSM. Its features include: 1) semantic clusters (nodes are categorized into clusters according to their node vectors), 2) semantic or random link (physical link which connects two nodes whose node vectors are semantically relevant or irrelevant, respectively), 3) node-based topology adaptation algorithm, and 4) node-based search protocol that combines biased walk and flooding. The drawback of GES is that if there are different categories of documents on a node, the node vector representation is not accurate.

The goal of CSS is to make searching more efficient, achieving higher recall at a lower search cost in the scenario of heterogenous single-node document collection. CSS extends GES by clustering all documents on a node into different classes and using multiple class vectors to represent document classes on a single node. CSS builds *virtual short links* and *virtual long links* between document classes on different nodes. These document classes are *virtual nodes* on a physical node. The virtual nodes, virtual long and short links form a virtual semantic overlay on top of the original P2P overlay. The virtual short and long links replace physical semantic and random links in GES. With the above variation, we develop a new class-based topology adaptation algorithm, a fully class-based search protocol CSS(1) and a partially class-based search protocol CSS(2). In addition, we take advantage of an efficient and effective document clustering algorithm, particularly the online spherical k-means clustering (OSKM)[27], to cluster documents on each node efficiently and precisely.

Our contributions in this paper are summarized as follows:

• We design a new distributed and dynamic semantic search specifically for heterogenous documents on a single peer in unstructured P2P networks. To the best of our knowledge, it is the first IR-based work targeted

at the scenario of heterogenous single-node document collection. The new concepts of class vector, virtual node, and virtual link are introduced. The new class-based topology adaptation algorithm and search algorithms are proposed. A good algorithm OSKM is exploited for effective and efficient document clustering.

• We extensively evaluate our search scheme using the IR benchmark TREC collection. The experimental results show that our scheme is more efficient than GES in all cases. Both CSS(1) and CSS(2) are effective in retrieving relevant documents.

The rest of the paper is organized as follows. We survey related work in Section 2. The preliminaries about VSM and GES are introduced in Section 3. We describe the design of each component of CSS in Section 4. Simulation results are presented in Section 5. The paper is concluded and future work is discussed in Section 6.

2 Related Work

Informed Search. An informed search, such as directed BFS[14], routing indices[15], and local indices[14], achieves more efficiency than a blind search by having each node forward a query to a subset of neighbors based on previous query results or the summaries of documents stored in neighbors. Similarly, in CSS nodes replicate the information about the class vectors of each neighbor and use this to direct walks. However, they differ in the information kept for neighbors.

Most semantic searches are either for structured P2P networks or for unstructured P2P networks. Some hybrid systems[28-30] combine structured and unstructured P2P networks. Semantic searches in unstructured P2Ps are ontology-based, IR-based, or Misc (miscellaneous).

Ontology-based Semantic Search. [19] describes peers' expertise using ontology and forwards queries to peers whose expertises are closest to the query. SenPeer[17] uses super-peers to maintain a distributed expertise table, describing data at neighboring peers using ontology. Queries are forwarded to relevant peers based on the expertise table. Different ontologies are mapped based on linguistic and structural similarity. Klink+[20] defines ontologies at peer-level and peer-group-level and provides sense-based search and concept-based search. The ontology mapping SECCO considers three different matchers: syntactic, lexical, and contextual.

IR-based Semantic Search. Some[22-23] just add semantic information into the P2P overlay. Others[16,21,24-26] build a semantic overlay before the search process. In the semantic overlay, nodes in the same semantic group (cluster) are close to each other. The major difference between CSS and existing IR-based semantic searches is that existing schemes either do not consider heterogeneous document collection on a single node[16,22-26], or recognize this scenario but do not accurately represent the single-node heterogeneous document collection[21]. They all use one category vector instead of multiple vectors to represent all documents on a single node. Additional differences are included below.

The work in [22] adds a hierarchical summary structure into the superpeer P2P overlay. Documents on peers and superpeers are summarized using IR models VSM and LSI. It requires that the underlying P2P architecture is constructed with superpeers. It is a hierarchically centralized system in which a centralized index is maintained at a server in superpeers, and all queries are directed to it first. In contrast, our scheme CSS is totally decentralized. Zhou *et al.*[23] add a content-based semantic relevance mechanism into the P2P overlay. The idea is to estimate the relevance of peers locally when receiving query messages. Only those peers deemed as relevant will receive the forwarded query. The similarity between the query model and the document collection model is then measured. Our search protocol is inspired from this method and then takes a further step by directly calculating the similarity between the query vector and the class vector, which is more accurate. CSS also differs from the work in [22-23] in that CSS creates a semantic overlay.

The topology adaptation algorithm in CSS bears similarity to SETS[21] and GES[16,25]. SETS tries to reorganize the network topology so that topic-related peers are close to each other. However, in SETS, a single designated node is responsible for clustering nodes into semantic topic segments, which is actually not distributed. CSS extends the topology adaptation algorithm of GES[16]. They differ from each other as follows. First, CSS uses class-based virtual links to assist the topology adaptation algorithm whereas GES is node-based. Second, CSS reduces two host caches (semantic and random caches) in GES into one since there is no corresponding concept of semantic and random links in CSS. Finally, CSS uses a novel formula to calculate the relevance score between nodes. The score is crucial to our topology adaptation algorithm.

In [24], a small world semantic overlay is constructed and maintained by using a gossiping mechanism. Each peer periodically sends out a query containing its own profile. When it receives an answer to its query, it will analyze the answer to decide whether to add the candidate node to its neighborhood or not. In some cases, neighbors may have to be replaced due to size constraints. CSS is different from this work because CSS uses random walk, not flooding, to build the semantic

overlay. The relevance calculation formula in CSS also differs from that in [24]. SEIF[26] employs the feedback from past queries to form a semantic overlay and automatically expands the current query. It depends on a query history of many similar queries. CSS does not have this limitation. DSC[31] is similar but independent work. [32] is our preliminary study about CSS.

Misc Semantic Search. These schemes are neither topology-based nor IR-based. They use various techniques to create a semantic overlay. The main difference between CSS and these approaches is that CSS employs IR models. In [33], Ng *et al.* proposed a query routing model called the firework query model on top of the clustered P2P network. It clusters P2P networks based on the characteristics of peer nodes and introduces the notion of attractive and random links, which resembles CSS. However, geographical information is used to determine the similarity between peer nodes. In [34], the semantic overlay network (SON) is built as follows. First, the documents at each node are classified and a document hierarchy is spread throughout the network. Second, the SON (clusters) is built according to that document hierarchy. Finally, each new node joins the SON in a Gnutella fashion (flooding) by finding a proper cluster. This work[34] studies searching for music song files. Queries are manually classified. CSS is concerned with text documents and does not need to classify queries.

SocioNet[35] is a semantic overlay based on users' interests in multimedia content. The overlay construction integrates the idea of social network and the resulting overlay exhibits small-world properties. Interests are expressed as weights of a small number of keywords such as genre values: rock, metal, pop, etc. It is not designed for full-text content search while CSS is good for both simple keyword search and full-text search. The studies in [18] and Agora[36] are designed for specialized domains. [18] creates a metric space overlay for locating multimedia contents based on their features, such as color and texture. Agora[36] builds a small-world semantic overlay for distributed control and automation. Nodes in Agora represent entities like transformers, generation plants, and central control rooms. CSS is designed for content-based text document retrieval.

Gia. CSS is also related to Gia[37], a non-semantic search. Gia uses a topology adaptation algorithm to balance the capacity of the network. We borrow the idea of the three-way handshake protocol for node join/leave from [37], and simplify it by removing the satisfaction level.

Document Clustering. There are many document clustering algorithms. They can be classified as k-means, fuzzy c-means, hierarchical clustering, and a mixture of Gaussian. K-means is an exclusive clustering algorithm. Online spherical k-means clustering (OSKM)[27] is an online version of the spherical k-means algorithm based on the well-known winner-take-all competitive learning. We choose it to cluster documents at each node because it achieves great clustering results and its implementation is not too complex.

3 Preliminaries

3.1 VSM

The vector space model (VSM)[38] is a way of representing documents through the words (terms) that they contain. It is a standard technique in information retrieval. In VSM, each document or query is represented using a term frequency vector. In each vector, the terms are stemmed with stop words (functional words like "is", "for", "the") removed. Suppose a collection includes two documents. The content of document 1 is "The quick fox jumped over the lazy dog's back". The content of document 2 is "Now is the time for all men to run quickly". Table 1 shows the VSM representation of the collection. In general, a collection of n documents $(D_1, D_2, \ldots, D_n)$ with t distinct terms $(T_1, T_2, \ldots, T_t)$ can be represented by a (sparse) matrix, in which w_{ij} means the weight of term i in document j.

$$\begin{bmatrix} & T_1 & T_2 & \cdots & T_t \\ D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\ D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \cdots & w_{tn} \end{bmatrix}$$

In addition, each term is assigned a weight that reflects its importance in describing the document content. Among several term weighting schemes, the "dampened" tf scheme weighs each term in the form of $1 + \log(tf)$ (tf means term frequency). It does not require global information, which fits well with the semantic search in P2P networks.

There are many different ways to measure how similar two documents are, or how similar a document is

Table 1. An Example Showing VSM Representation

Indexed Term	all	back	dog	fox	jump	lazy	men	now	over	quick	run	time
Document 1	0	1	1	1	1	1	0	0	1	1	0	0
Document 2	1	0	0	0	0	0	1	1	0	1	1	1

to a query. The cosine measure is a very common similarity measure. Given a normalized document vector D and a normalized query vector Q, the relevance score is calculated as:

$$REL(D,Q) = \sum_{t \in D,Q} f_{t,D} \times f_{t,Q} \quad (1)$$

In the above formula, t is a common term occurring in both D and Q. $f_{t,D}$ is the weight of term t in D and $f_{t,Q}$ is the weight of term t in Q.

3.2 GES Overview

GES[16] is a distributed, content-based IR system proposed by Zhu *et al*. There are four components in GES that are also in CSS. However, they are different in that CSS introduces the new concept of class vector and makes all these components class-based. In addition, a query is routed through virtual links in the search protocol CSS(1) rather than through physical links in GES. The following is a brief summary of the GES version of these components:

Node Vector and Physical Link. A node vector is a compact outline of all documents on a node, which is indeed an average normalized term vector derived from the term vectors of all documents on a node. To judge whether two nodes or a node to a query are relevant, the cosine similarity measure between node vectors and the query vector is used. A physical link is a link that connects two nodes in the P2P network. Physical links can be further classified as semantic (relevant) or random (non-relevant) links based on the relevance score of the node vectors of the two adjacent nodes.

Topology Adaptation Algorithm. The goal is to organize relevant nodes into semantic groups through semantic links. The algorithm is implemented in a distributed manner. Each node periodically issues two random walk query messages that contain its node vector. One is for nodes whose node vectors are sufficiently relevant to the node vector of the query source, the other is for non-relevant nodes. These two kinds of candidate nodes selected by the query are put into the query source's semantic and random host caches, respectively. After that, each node periodically checks these two caches for semantic or random neighbor addition/replacement based on the relevance score calculated from node vectors. When the number of neighbors reaches the maximum limit, an existing neighbor has to be dropped before a new neighbor can be added.

Selective One-hop Node Vector Replication. Each node maintains the node vectors of its random neighbors to assist the informed search process. The node vectors of semantic neighbors are not replicated.

Search Protocol. GES uses a biased walk rather than a random walk to forward a query through random links. Each node looks up its local documents satisfying the query. If at least one relevant document is found on a node, this node is called a *semantic group target node*. This target node terminates the biased walk and starts flooding the query along its semantic links. If no relevant document is found, the node forwards the query to the random neighbor whose node vector is most relevant to the query vector. This two-stage search protocol makes a query walk into a proper semantic group and then retrieves many useful responses within it. Besides, the book-keeping technique is also used in GES to sidestep redundant paths.

4 Scheme Design

4.1 Overview

In CSS, each node contains several classes of documents and has corresponding class vectors as a summary of the semantic content on each node. Each class may have two types of virtual links, short and long links, which connect with similar and dissimilar classes on its neighbor nodes, respectively. The topology adaptation algorithm reorganizes the network according to the similarity of the contents on different nodes. We design two search protocols. The class-based search protocol CSS(1) first directs a query through long links, trying to locate a class within a relevant semantic group. Once such a relevant class is found, CSS(1) then floods the query within that semantic group to retrieve more relevant documents. The partially class-based search protocol CSS(2) searches several selected classes instead of all classes on a node at a time, which greatly reduces message overhead.

4.2 Class Vector and Virtual Link

A class vector is a centroid vector of all documents in a class. We calculate class vectors on a node based on VSM, as follows. First, a term vector is derived to represent a document, in which each term's weight is assigned by its term frequency in that document. Second, we re-weigh each term using the "dampened" *tf* scheme in the form of $1 + \log(tf)$. Third, we normalize the weighed term vector to unit length. Fourth, we feed all processed term vectors (corresponding to all documents on a node) to the OSKM[27] clustering algorithm. Finally, given the number of classes you want to cluster (e.g., 6), the algorithm outputs the given number of normalized class vectors and a list showing which document belongs to which class.

Given two classes of documents (class X and Y), their relevance score is the cosine similarity of their

normalized class vectors, as listed below:

$$REL(X,Y) = \sum_{t \in X,Y} w_{t,X} \times w_{t,Y} \quad (2)$$

In this formula, t is a common term occurring in both class vector X and class vector Y. $w_{t,X}$ is the weight of term t in X, and $w_{t,Y}$ is the weight of term t in Y. If the relevance score is no less than a certain threshold, these two classes are considered relevant. Otherwise, they are not.

Sometimes we need to define the relevance between a normalized class X and a normalized query Q. The following formula applies:

$$REL(X,Q) = \sum_{t \in X,Q} w_{t,X} \times w_{t,Q} \quad (3)$$

In CSS, we build virtual links on top of physical links. Physical links are the P2P overlay links that connect peers. Virtual links connect two classes on different nodes. Formally, let E be the set of physical links and E' be the set of virtual links. Thus, CSS makes a many-to-one mapping from E' to E. It means that many virtual links can be mapped to one underlying physical link.

The goal of conceptual virtual links is to connect classes of documents on different nodes virtually. If the relevance score between two classes is no less than *short_rel_thres*, we build a virtual link between them and call it *short link*. If the relevance score between two classes is no more than *long_rel_thres*, we build a virtual link between them and call it *long link*.

Note that it is necessary for each document class on a node to have at least one long link to each of its neighbors because the directed walk in the search protocol forwards queries through long links. If the relevance scores of all virtual links coming from one class are higher than *long_rel_thres*, we just pick the link with the lowest score as the long link.

We do not classify physical links, as done in GES. Instead, we classify virtual links as short and long links in CSS. The short and long links can be considered as the extension of semantic and random links in GES. Note that we use two thresholds to classify virtual links instead of the *node_rel_threshold* in GES. The reason is that we do not want to build a virtual link between classes with relevance scores that are not high or low enough (e.g., 0.5). Using two thresholds yields better classification.

Fig.2 shows an example of a physical link and three virtual links. The relevance score between class 1 on node X, whose content is about baseball, and class 1 on node Y, whose content is about football, is higher than *short_rel_thres*, so a short link is built since these two classes both belong to sports. On the contrary, the relevance score between class 1 on node X and class 2 on node Y, whose content is about cooking, is lower than *long_rel_thres*. So, a long link is built between them because the two classes are not relevant. There is no virtual link built between class 1 on node X and class 3 on node Y because they have medium relevance.

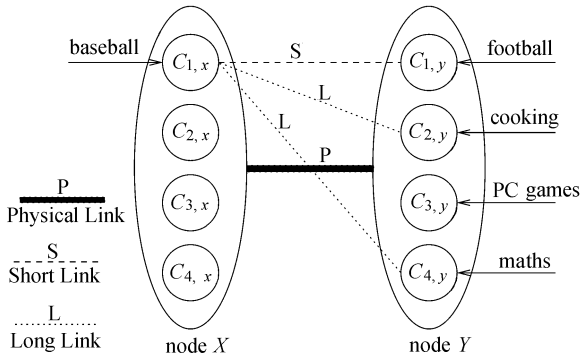

Fig.2. Class vector and virtual link. ($C_{1,x}$ means class 1 on node X).

4.3 Topology Adaptation Algorithm

4.3.1 The Goal

The topology adaptation algorithm is an important part of our search scheme. It aims not only to maintain node connectivity, but also to achieve a refined network topology for better search performance. Simply speaking, it aims to find "good" neighbors for each node in a distributed manner. The main goal of the topology adaptation in GES[16] is to ensure that relevant nodes are organized into semantic groups which may be relevant to the same queries. Our topology adaptation algorithm has to consider more factors since class vectors on a node play an important role. Our goal is to ensure that 1) relevant classes on different nodes are connected through virtual similar links (namely short links), and 2) each class should have enough virtual dissimilar links (namely long links) in order to discover proper virtual semantic groups.

4.3.2 Calculating Overall_Score

Short links and long links are both valuable because during the search process, a query is flooded through short links and directedly routed through long links. The simulation result shows that they both affect the performance greatly. Therefore, we need a criteria to judge whether a node is a good candidate to be a neighbor or not. A formula is designed to calculate the *overall_score* between two nodes. We name it *overall_score* since it considers both relevance factor (short links) and

non-relevance factor (long links). Given two nodes, the *overall_score* is defined as follows:

1) Find all short and long links by calculating the relevance scores between all class vectors of the two nodes using (2). The relevance scores of a short link and a long link are denoted by *rel_score_short* and *rel_score_long*, respectively.

2) Define a medium value.

$$rel\_med = (short\_rel\_thres + long\_rel\_thres)/2 \quad (4)$$

3) Sum up the differences between *rel_score_short* and *rel_med* for all short links. Similarly, the differences between *rel_med* and *rel_score_long* for all long links are summed up.

$$sum\_diff\_short = \sum_{all\ short\ links} (rel\_score\_short - rel\_med), \quad (5)$$

$$sum\_diff\_long = \sum_{all\ long\ links} (rel\_med - rel\_score\_long). \quad (6)$$

4) Add a weight factor w.

$$overall\_score = sum\_diff\_long + w \cdot sum\_diff\_short. \quad (7)$$

4.3.3 Topology Query Process

When a node joins the network, it first randomly connects to other nodes using a bootstrapping mechanism, as in Gnutella. At this time, it is not aware of any content (e.g., class vectors) of other nodes and its classes do not belong to any virtual semantic group. This node then periodically issues a topology query message for such information. The query is routed throughout the network using random walk bounded by a TTL (time to live) until sufficient responses are obtained. This kind of query is different from the search query. We name it *probe query*.

A probe query message contains all class vectors of the node, the maximum number of responses, and the TTL. The query returns some qualified nodes ranked in decreasing order of *overall_score*, which will be added to the query initiator's host cache. Each entry in the host cache consists of a node's IP address, node degree, class vectors, and *overall_score*. The cache is continuously updated during the lifetime of a node using the candidates discovered by the periodic topology query process. Fig.3 illustrates the topology query process and the corresponding list of host cache entries. The topology query starts with node 1. It randomly walks to node 6, then 22, and etc. Finally it ends with node 15. This query finds five candidate nodes: 22, 2, 8, 4, and 19.

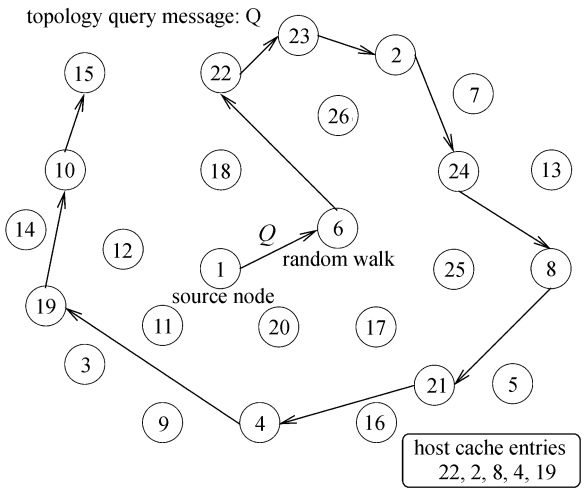

Fig.3. Topology query process.

4.3.4 Topology Modification Process

With candidate nodes stored in host cache, each node periodically performs neighbor addition and replacement in a similar way to Gia[37]. After a neighbor is added, short and long links for that neighbor will be established immediately. Similarly, before a neighbor is dropped, all short and long links associated with that neighbor have to be removed. A node, say X, chooses a candidate node with the highest *overall_score* from its host cache and verifies that the candidate is still active and not an existing neighbor. X then uses a three-way handshake protocol to communicate with the selected neighbor candidate, say Y. It is a distributed handshake protocol, which means that each node decides independently whether to accept the other node as a new neighbor or not. These nodes make a decision according to their own *max_links* (the maximum number of neighbors allowed), current degree, and *overall_score* of the requesting node.

If the current degree is less than *max_links*, the node automatically accepts the requester as a new neighbor. Otherwise, the node has to check whether it can find a suitable existing neighbor to drop and replace it with the requesting node. X makes such a decision in the following manner. From all of X's neighbors that are not poorly connected and whose *overall_scores* are lower than that of Y, X chooses the neighbor Z with the lowest *overall_score* to drop and adds Y as its new neighbor. A poorly connected node is a node whose degree is less than or equal to the minimum number of neighbors required, namely *min_links*.

Fig.4 illustrates the topology modification process at node 1. Node 4 is the candidate node in the host cache

with the highest *overall_score*. The maximum number of neighbors allowed is 6. Because this maximum is reached, node 1 has to drop existing neighbor 9 in favor of new neighbor 4.

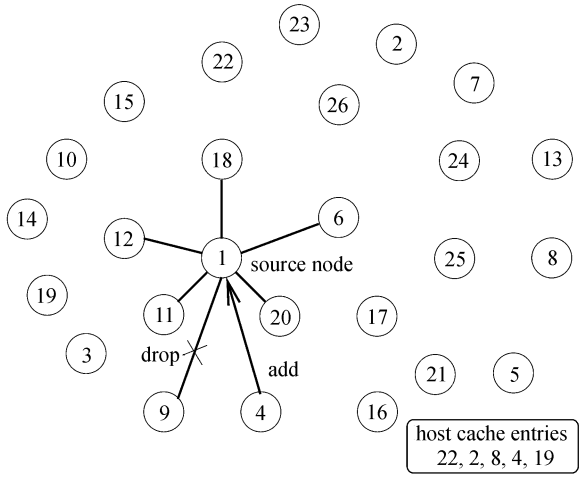

Fig.4. Topology modification process at node 1.

4.3.5 Handling Dynamics

Changes in the document collection may invalidate existing class vectors and neighbors. Each node periodically detects whether the contents of its neighbors have changed and keeps updating the *overall_score* of all of its neighbors. If many documents on a neighbor have been added, removed, or changed, then the clustering algorithm is run again on that neighbor and the class vectors are updated. Each node discovers these changes, obtains updated class vectors from its neighbors, recalculates the *overall_scores*, and updates its neighbors when necessary. If the *overall_score* of an existing neighbor is too low (e.g., lower than a certain threshold), in order to keep all short and long links intact we do not simply drop that neighbor immediately. Instead, we wait for a good candidate node to replace that neighbor during periodic topology adaptation.

To reduce the maintenance overhead of the topology adaptation, changed class vectors because of document addition/deletion/update are not sent in its original format. Instead, the vector differences are delivered. In addition, the periodic keep-alive messages can carry the vector differences to further reduce the overhead. Compression techniques may also be applied to the vector deltas.

Node leaves or node failures may also make existing neighbors invalid. When a node leaves the network, it notifies its neighbors about this change. Each neighbor removes this node, adds a new one from the host cache, and creates new short and long links. A failed node is detected by its neighbors through keep-alive messages. The neighbors of the failed node remove this node, add a new one from their own host cache, and create new short and long links. Node addition in this scenario follows the same rule as in Subsection 4.3.4. In summary, our topology adaptation algorithm can fit a dynamic situation well.

4.4 Selective One-hop Class Vector Replication

Each node should store information about the class vectors of all its neighbors in order to assist the search process. We only store class vectors of the classes connected via long links. That is why we call it selective replication. During the search process in CSS(1), queries are routed through one of the long links (need to compare and select one) and flooded through all short links (no need to compare and select). Thus, we need to replicate information about long links for selection purposes. If a neighbor leaves the network, then the information about its class vectors will be deleted. If the documents on a node's neighbors have been changed, it will receive the updated class vectors and then recalculate the *overall_score*. In CSS(2), each node does not need to store class vectors of its neighbors. Instead, each node should continue to update and guarantee that the *overall_score* is the latest. In summary, our replication algorithm can handle dynamic situations, such as node join/leave or the change of documents on neighbor nodes.

4.5 Search Protocol

The topology adaptation algorithm refines the network topology, and selective one-hop class vector replication informs each node of the class vectors of its neighbors. In this subsection, we discuss two content-based search protocols. One protocol, named CSS(1), is totally class-based and virtual link assisted. It means that queries are routed from one class to another along virtual links. The other protocol, called CSS(2), is partially class-based or node-based, which means that queries are routed from one node to another. CSS(2) can also be considered a variant of CSS(1).

4.5.1 Search Protocol CSS(1)

A query is in two separate modes during the search process: first in *directed walk* mode (walk with each hop selected intelligently) and then *flooding* mode. The goal of the directed walk is to locate a class within a semantic group that is relevant to the query. This relevant class contains the first relevant document. In this mode, queries are forwarded along long links because a relevant document may not be on the query source

or the node receiving the query. Long links connect irrelevant document classes. After the class containing the first relevant document is found, the query is set to be in flooding mode. The goal of this mode is to find more relevant documents. The relevant class discovered in the direct walk mode starts the flooding of queries along short links. This mode uses short links because a relevant class is found and short links connect relevant classes.

When a node initiates a query, it first calculates and compares the relevance scores between the query vector and all its class vectors. The query is then routed to the class with the highest relevance score. This class is called the *query source class*.

After the query source class is found, the query is set to be in directed walk mode. When receiving a query in this mode, each class looks up its locally stored documents for those satisfying the query. A relevance score is calculated between the query and each document using (1). If it is higher than a certain threshold, this document is identified as a relevant document for the query. If at least one relevant document is found, then this class (say A) of the node (say X) is called a *virtual semantic group target class*, or *target class* in short. At this time, the query ends directed walk mode and enters flooding mode. If all documents in class A are identified as non-relevant, class A selects another class (say B) whose class vector is most relevant to the query vector according to (3) from all other classes on the same node X and classes connected via X's long links.

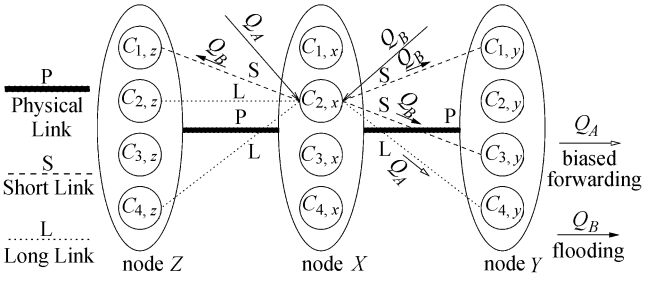

Fig.5. Query routing graph for CSS(1). ($C_{1,x}$ means class 1 on node X).

Fig.5 illustrates how node X intelligently chooses a class to forward the query given the set of virtual links in Fig.3. The query Q_A begins with the class $C_{2,x}$ of node X and no documents in $C_{2,x}$ are relevant to the query. So, class $C_{2,x}$ tries to find the most suitable class to biasedly forward the query. Its choices include all other classes on the same node X ($C_{1,x}$, $C_{3,x}$, $C_{4,x}$) and classes connected via X's long links ($C_{4,y}$, $C_{2,z}$, $C_{4,z}$). Class $C_{2,x}$ finally chooses $C_{4,y}$ since its class vector is most relevant to the query vector. The directed walk continues similarly until a target class is found.

The target class then sets the query to be in flooding mode and floods the query along all its short links. Each semantically-related class receiving the query looks up all documents in the class and floods the query along its own short links. Fig.5 shows that if $C_{2,x}$ is the target class for the query Q_B, it will flood Q_B along its short links to the classes $C_{1,y}$, $C_{3,y}$, and $C_{1,z}$. In addition, the radius of flooding is controlled via TTL. The relevant documents found within the semantic class group are reported to the target class directly. The target class is responsible for aggregating all these files and reporting them directly to the query source. If the number of relevant documents discovered so far is below the user expectation, as specified in the original query, the target class starts another directed walk and the above search process is repeated. CSS(1) is illustrated in Algorithm 1.

Algorithm 1. CSS(1)

```
1:  A query reaches the source node X
2:  /*Initialization*/
3:  the query is located in the identified query source class
4:  set query_mode ← directed walk
5:  while insufficient responses and TTL bound not reached do
6:      if query_mode = directed walk then
7:          look up relevant documents in the current class
8:          if one relevant document found then
9:              mark current class: a target class
10:             set query_mode ← flooding
11:         else
12:             define set C ← {all other classes in the same node X} ∪ {classes connected via node X's long links}
13              forward the query to the class in set C which is most relevant to the query
14:         end if
15:     end if
16:     if query_mode = flooding then
17:         flood the query along all short links
18:         if no short link found or beyond flood radius then
19:             set query_mode ← directed walk
20:         end if
21:     end if
22: end while
23: return
```

We also use TTL to bound the duration of the

directed walk and the book-keeping technique to avoid redundant paths. In CSS(1), each query is assigned a unique GUID by its originator class. Each class remembers the classes to which it has already forwarded queries for a given GUID. If a query with the same GUID comes back to the class, it will be forwarded to a different class with the highest relevance score in directed walk mode, or it will simply be discarded in flooding mode. However, to guarantee forwarding progress, if a class has already sent the query to all possible classes, it flushes the book-keeping state and starts reusing classes for directed walk.

In summary, the directed walk guides the query along long links towards a target class with similar semantic content. Flooding uses short links to locate enough desired documents near the target class. Our totally class-based search protocol CSS(1) makes searching efficient by using a smaller search unit: a class of documents on a node instead of all documents on a node.

4.5.2 Search Protocol CSS(2)

We describe our second search protocol CSS(2) as follows. A query is always in *directed walk* mode (walk with each hop selected intelligently) during the search process. When a node initiates a query, it first searches its own contents and then routes the query to one of its neighbors. The query usually searches several classes on a node at a time so it visits each node only once during the whole search process.

After the query source finishes checking its local files, the query is routed to one of its neighbors with the highest *overall_score*. On receiving the query, the neighbor calculates the relevance scores between the query vector and all of its class vectors. The classes whose relevance scores are higher than a certain threshold are selected and searched one by one. A relevance score is calculated between the query and each document in a chosen class using (1). If the score is higher than a certain threshold, this document is considered to be relevant to the query and is included in the result set for the query. All relevant documents found within the selected classes are directly reported to the query source. When the number of relevant documents discovered so far meets the requirement, the entire search stops.

Fig.6 illustrates how node X conducts the search and intelligently chooses a neighbor to forward the query. When a query reaches node X, it calculates the relevance scores between its query vector and all class vectors ($C_{1,x}$, $C_{2,x}$, $C_{3,x}$ and $C_{4,x}$) on node X. The classes $C_{2,x}$ and $C_{3,x}$ are chosen since their relevance scores are higher than a preset threshold. After searching classes $C_{2,x}$ and $C_{3,x}$, the query is routed to node Y whose *overall_score* is the highest among node X's neighbors. We then look up the selected classes $C_{1,y}$ and $C_{3,y}$ on node Y. The search process continues until there are sufficient responses or the TTL expires. CSS(2) is illustrated in Algorithm 2.

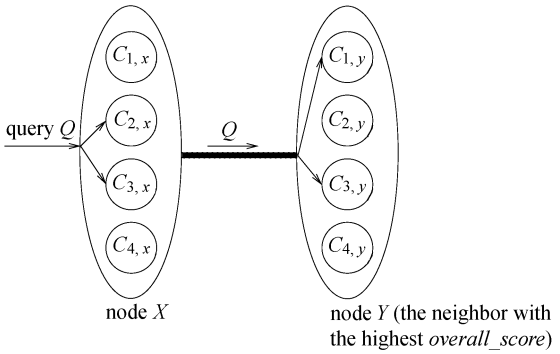

Fig.6. Query routing graph for CSS(2). ($C_{1,x}$ means class 1 on node X).

Algorithm 2. CSS(2)

```
A query Q reaches the node X
/* Initialization */
define set C ← null
define T ← a preset threshold
set query_mode ← directed walk
while insufficient responses and TTL bound not reached do
    calculate the relevance scores between the query vector and all of node X's class vectors
    for each node X's class i do
        if (the relevance score between query Q and class i) > T then
            set C ← set C ∪ class i
        endif
    endfor
    for each class j in set C do
        look up relevant documents in class j
    end for
    report all relevant documents directly to the query initiator node
    forward the query to the neighbor of X with highest overall_score
end while
return
```

The book-keeping technique is also used in CSS(2) to avoid redundant paths. The technique differs from that in CSS(1) because in CSS(2) each node is responsible for remembering the visited neighbors, whereas in CSS(1) each class (like a virtual node) does its own

book-keeping.

In summary, in CSS(2), given a query, some selected classes on a node are looked up and the query is biasedly forwarded in favor of nodes with higher *overall_scores*. Our partially class-based search protocol CSS(2) makes searching efficient by clustering documents into classes and by routing queries with reduced message overhead.

4.5.3 Comparison of CSS(1) and CSS(2)

We design two search protocols CSS(1) and CSS(2). They both take advantage of clustering documents into classes, but are different in essence. CSS(1) is totally class-based while CSS(2) is partially class-based or node-based. During the search process, CSS(1) routes a query from one class to another via virtual links. Therefore, it is possible that a query visits different classes on the same node at different times, which results in high message overhead. CSS(2) is designed to search all qualified classes at a time so that a query only visits a node once, thereby lowering message overhead. The maximum recall for a search using CSS(1) is 100% since a query visits all classes, e.g., all documents in the network. However, the maximum recall for a search using CSS(2) is only around 80%. This is because a query only visits qualified classes that are similar enough to the query instead of all classes in the network. Table 2 summarizes the comparison.

Table 2. Comparison of Search Protocols CSS(1) and CSS(2)

	CSS(1)	CSS(2)
Search Characteristics	Class-based	Node-based
Node Visiting	A query may visit a node multiple times	A query only visits a node once
Message Overhead	High	Low
Maximum Recall	100%	Around 80%

5 Simulation

In this section, we present the results of our class-based search scheme (CSS). We first discuss text datasets used in the simulation and then explain performance metrics. After that, simulation settings are described. Finally, CSS is evaluated in different network configurations and document distributions.

5.1 Text Datasets

We use all documents of AP newswire 1988 along with Topics 151∼200 in TREC disks 1&2 to evaluate our search scheme. They can be found in TREC's ad hoc test collections[39]. The goal of TREC is to provide a benchmark for evaluation in information retrieval from large text collections. We then extract the text field from these documents by removing irrelevant parts. Rainbow[40], a powerful toolkit used to preprocess raw text files for further classification and clustering, is then used to calculate the term frequencies for all terms (words) in these documents. Rainbow is also able to perform a stemming process with the aid of Porter stemmer. Finally, there are 79 986 files distributed over 1 000 nodes. The queries we use are from TREC topics 151∼200. The query vectors are obtained from the text field of these topics using Rainbow. Thus, each query is not a keyword-based search, but rather a text-based search. The query vectors are also stemmed with stop words removed. In addition, the TREC website[39] provides relevance judgement files that can be viewed as "correct answers" for the above 50 queries. These assessment files are obtained by manual identification and are vital to our simulation.

5.2 Performance Metrics

The following performance metrics are used in the simulation.

1) *Recall*. It measures the coverage of available relevant results. It is defined as the number of relevant documents retrieved divided by the total number of relevant documents in the system.

2) *Precision*. It is defined as the number of truly relevant documents divided by the total number of documents retrieved.

3) *Search Cost*. It is defined as the percentage of classes in the network visited by a query. An efficient search algorithm incurs a low search cost. The less classes a query visits, the shorter the search response time, and the less the computing resource consumption, especially when there are a huge number of documents in the network.

5.3 Comparison Criteria

As described in the search protocol section, CSS(1) is a class-based search protocol while GES and CSS(2) are node-based. This means that CSS(1) routes a query from one class to another while GES and CSS(2) route a query between nodes. Therefore, how can we compare the performance of CSS(1) and GES/CSS(2) using the same criteria? First, we globally cluster all documents in the text datasets into twenty classes by using OSKM[27]. Second, a tunable parameter, *cpn* is defined to reflect different document distribution. *cpn* is the abbreviation of *classpernode*, which is the number of classes placed at each node. If *cpn* equals 5, there are five classes of documents per node, which means that we randomly pick up some documents from five classes of the above twenty global classes and assign them to nodes. Third, the metric we use is *percentage of classes*

visited. There is no problem with class-based CSS(1). As for GES, we turn to the conversion that the number of classes visited equals the product of the number of nodes visited and *cpn*. As for CSS(2), the number of classes visited is the total number of classes that are actually selected for the query to search. As a result, the same metric is applicable to all search protocols.

5.4 Simulation Setting

We use a custom simulator to test our search scheme. Table 3 lists the simulation parameters and their values. A random graph with an average degree of 6 is generated first as the initial topology, then the topology adaptation algorithm is run for several rounds to reorganize the initial topology. The search process is then performed on the refined topology. In our search scheme (CSS), there are two fixed parameters (*min_links* and *max_links*) and five tunable parameters (*classpernode* or *cpn*, *short_rel_thres*, *long_rel_thres*, w, *good_neigh_thres*). We set *min_links* = 3 and *max_links* = 10, which are the min. and max. limits for node degree. The tunable parameter *cpn* is set from 1 to 10. *short_rel_thres* and *long_rel_thres* represent two thresholds to define short and long links, respectively. We set their values to 0.7 and 0.3, respectively. The weight factor w ($w = 3$) is used to strengthen the effect of short links when calculating *overall_score* because the number of short links is much less than that of long links. We use 3.0 as the value of the last tunable parameter *good_neigh_thres*, which is the criterion for "good" candidate nodes to be put into the cache in the probe query process.

Table 3. Simulation Parameter Setting

Parameter	Value
min_links	3
max_links	10
short_rel_thres	0.7
long_rel_thres	0.3
cpn	1 to 10
w	3
good_neigh_thres	3.0

All values of the above tunable parameters are obtained from experiments. So, they are heuristic values. However, there are some rules as to why we choose these particular values. We set the value of *short_rel_thres* larger so that each short link is built between two classes that are very relevant. This makes flooding short links very effective in finding highly relevant documents. As for the value of *long_rel_thres*, if it is too small, then there are too few long links for a query to find a class relevant enough to route itself to. If it is too large, then there will be too many long links for directing a query, which consumes more resources and causes a longer delay. Finally, if the host cache is large, we can set *good_neigh_thres* smaller so that more candidate nodes can be considered.

5.5 Simulation Results

We compare the performance of our search scheme (CSS), including CSS(1) and CSS(2), with GES in different document distributions and different network configurations. We also discuss the maintenance overhead of CSS and GES.

5.5.1 Number of Classes per Node

Fig.7 shows the performance with different values of *classpernode* (*cpn*) ranging from 1 to 10. Four typical graphs with *cpn* = 1, 4, 7, 10 are illustrated. Some observations are as follows:

1) CSS(1) is the best, CSS(2) is the second, and GES is the worst when the parameter *cpn* varies. If *cpn* equals 1, then CSS(1) is reduced to GES and they behave in the same way. Thus, the search results with *cpn* = 1 are identical, as shown in Fig.7(a). However, in Figs. 7(b)∼7(d), both CSS(1) and CSS(2) outperform GES by achieving higher recall at the same search cost. This can be explained as follows. Since a query is routed through classes in CSS(1), or it selects part of all classes to search in CSS(2), it can locate the target class more precisely and quickly without visiting many irrelevant documents. We also see that CSS(1) performs better than CSS(2). The reason is that though both CSS(1) and CSS(2) only search part of all documents on a node, they behave in a different way. CSS(2) searches several classes at a time while CSS(1) searches one class at a time. This means that CSS(1) uses a smaller search unit so that it achieves higher recall when visiting the same number of classes.

2) The trend from Figs. 7(b)∼7(d) indicates that both CSS(1) and CSS(2) perform better than GES when the number of classes at each node increases. This is because the advantage of clustering documents into classes will be more obvious as the number of classes increases. The more classes on a node, the more choices for a query to investigate and the higher precision the search has. We demonstrate that both CSS(1) and CSS(2) outperform GES in different document distributions. CSS(1) achieves better search efficiency than CSS(2) with the cost of higher message overhead. As the number of classes at each node increases, the gap between CSS(1) and CSS(2) becomes larger.

5.5.2 Network Configuration

Node Degree. Figs. 8(a)∼8(b) compare the performance with different initial node degrees (deg = 3, 6,

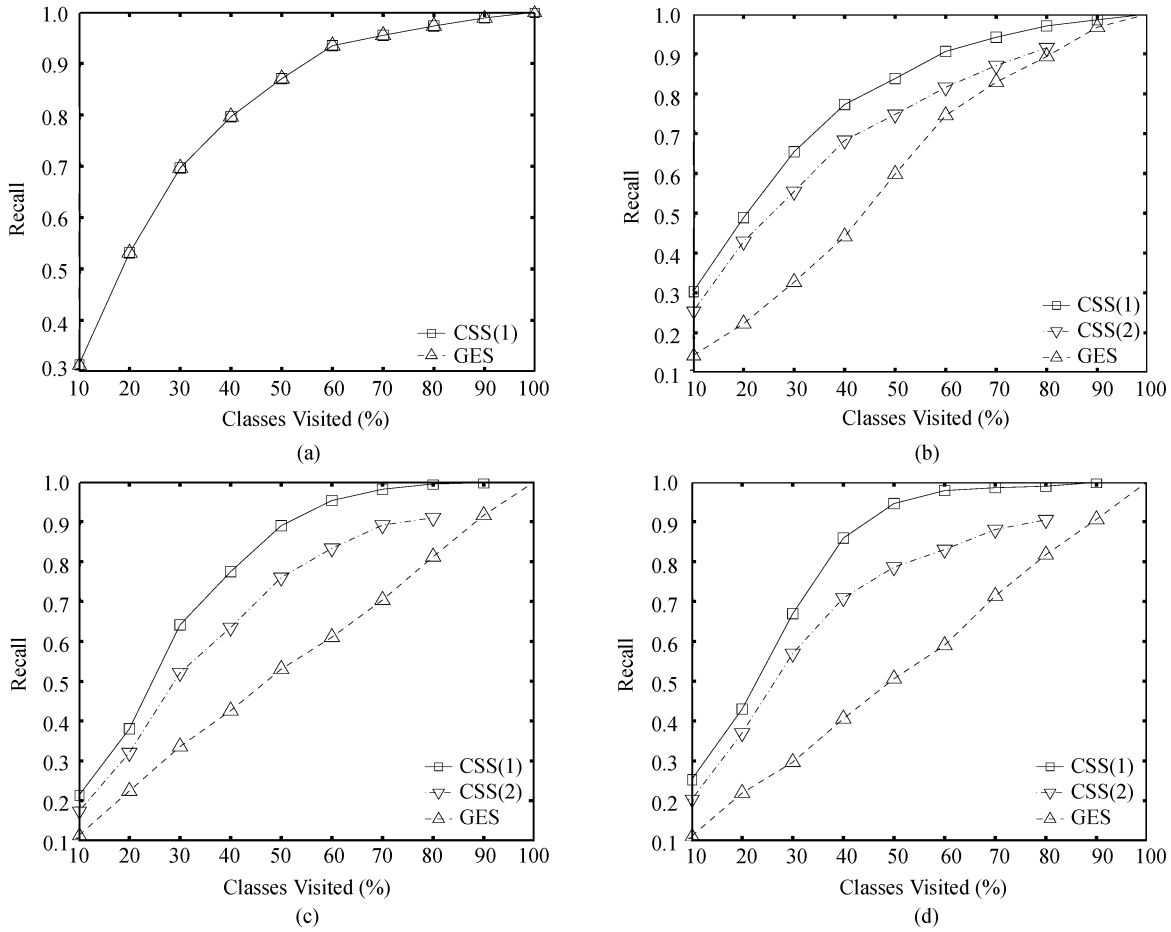

Fig.7. CSS vs GES with different numbers of classes per node (nodenum = 1000, deg = 6). (a) Number of classes per node = 1. (b) Number of classes per node = 4. (c) Number of classes per node = 7. (d) Number of classes per node = 10.

9). We can see that CSS(1) is still the best and CSS(2) beats GES. However, the difference between the group of curves of each protocol is very small. This is because the topology adaptation algorithm plays a significant role in addition and replacement of neighbors so that initial node degree seems less important.

Network Size. Figs. 8(c)∼8(d) compare the performance with different number of nodes in the network (*nodenum* = 500, 1 000, 2 000). Both CSS(1) and CSS(2) exceed GES. CSS(1) outperforms CSS(2) in most cases. Also, the difference between the group of curves of each protocol is relatively larger than that in Figs. 8(a)∼8(b). This is because the number of nodes in the network has a direct impact on topology adaptation and search process. CSS proves to be applicable to P2P networks of different sizes.

Network Type. Figs. 9(a)∼9(c) compare the performance of CSS(1) and GES in different types of networks: random graph, power-law graph, and square-root graph. The number of classes per node is 7. CSS(1) performs better than GES in all three types of networks. The performance trends are similar in the three network types. Figs. 10(a)∼10(c) compare CSS(2) to GES in the same network scenario. CSS(2) also has a better performance than GES in all three network types. The network type does not make a difference in the performance of CSS(1) against CSS(2). CSS(1) still does better than CSS(2) in the three types of networks, as shown in Figs. 11(a)∼11(c).

5.5.3 *Precision-vs-Recall*

Precision vs recall is a standard measure in information retrieval. Fig.12 presents a graph of precision vs recall. We can observe that the precision of CSS(1) or CSS(2) is higher than that of GES in the same recall, which means both CSS(1) and CSS(2) are more precise in the search process due to their class-based search characteristics. CSS(1) wins over CSS(2) with respect to search precision since it uses a smaller search unit. However, the highest precision is around 32%, which seems low. This is because the truly relevant documents for a query in the relevance judgement file are

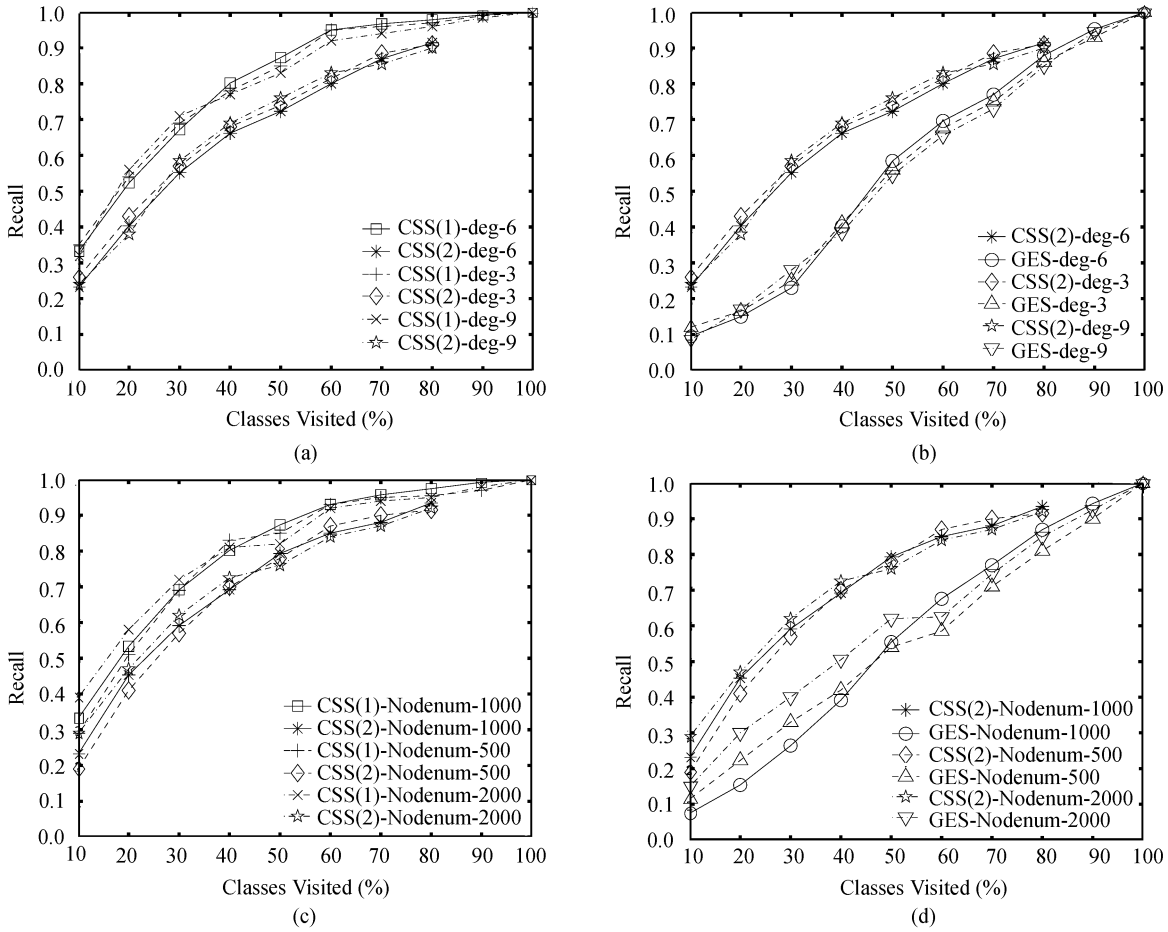

Fig.8. CSS vs GES with different network configurations. (a) CSS(1) vs CSS(2) with different node degrees (cpn = 5, nodenum = 1 000). (b) CSS(2) vs GES with different node degrees (cpn = 5, nodenum = 1 000). (c) CSS(1) vs CSS(2) with different node numbers in the network (cpn = 5, deg = 6). (d) CSS(2) vs GES with different node numbers in the network (cpn = 5, deg = 6).

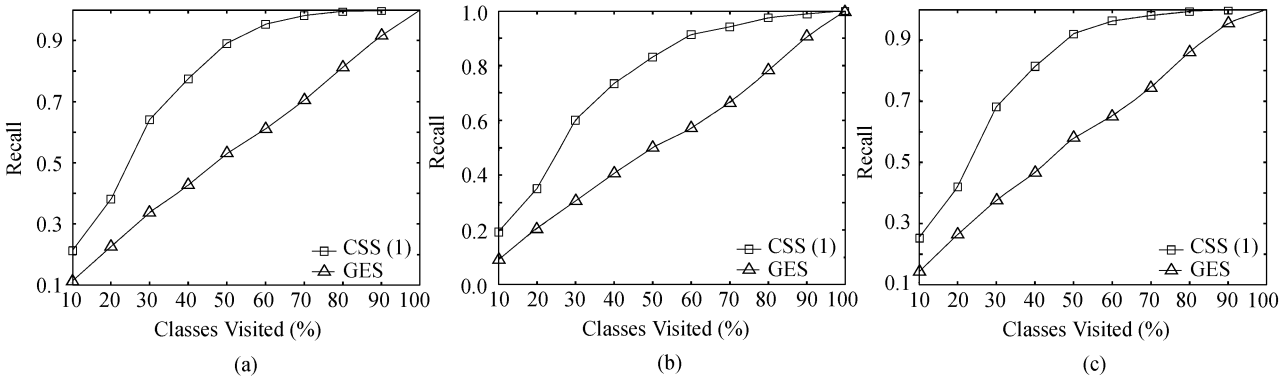

Fig.9. CSS(1) vs GES with different network types with cpn = 7. (a) Random graph. (b) Power law graph. (c) Square root graph.

only around one-thousandth of the whole document collection (text datasets). Besides, we aim to conduct a full search to find all truly relevant documents since there are not many. This is why we cannot set a high threshold for document retrieval.

5.5.4 *Topology Adaptation Overhead*

In CSS, each node sends periodic probe queries, looking for good candidate neighbors with high overall scores. Each probe query carries the class vectors

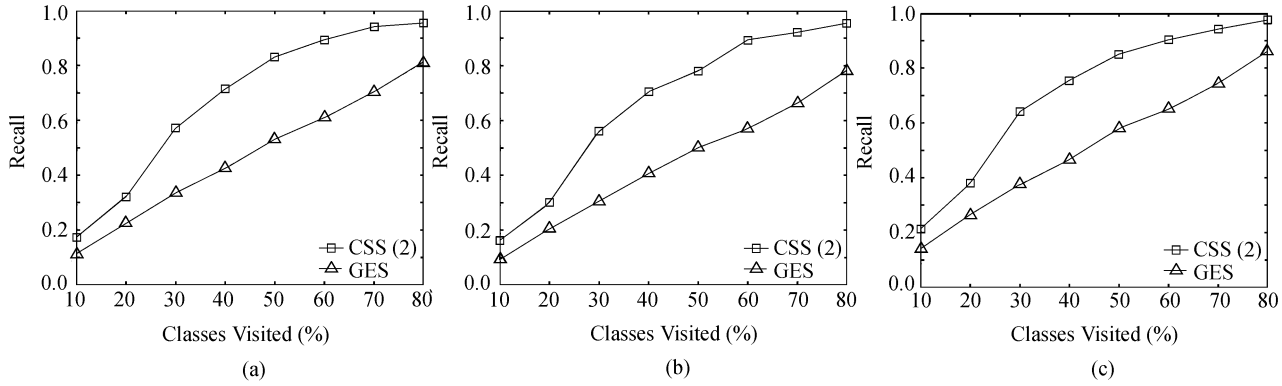

Fig.10. CSS(2) vs GES with different network types with cpn = 7. (a) Random graph. (b) Power law graph. (c) Square root graph.

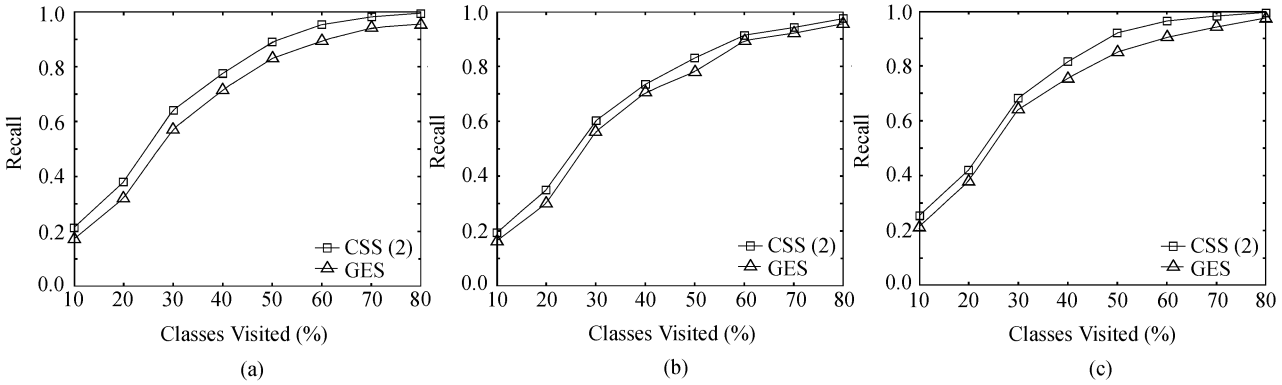

Fig.11. CSS(1) vs CSS(2) with different network types with cpn = 7. (a) Random graph. (b) Power law graph. (c) Square root graph.

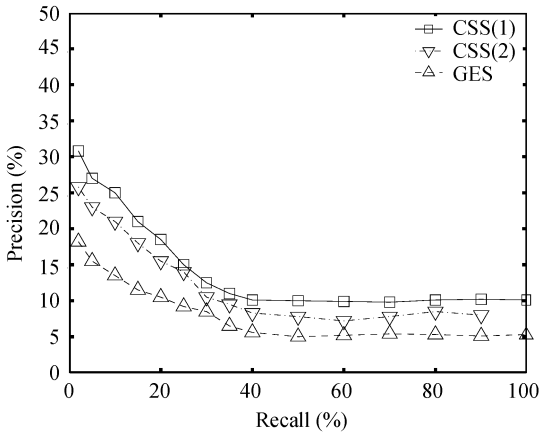

Fig.12. Precision vs recall (cpn = 5, nodenum = 1000, deg = 6).

of the node initiating the query. The response message to a probe query includes only overall scores of candidate nodes. Class vectors of a candidate are delivered to a node only when the candidate is chosen as a new neighbor by this node. In GES, each node periodically sends two probe query messages. One searches for potential good candidates as random link neighbors, and the other for good candidates as semantic link neighbors. Each probe query carries the node vector of the node issuing the query. Each response message contains only the relevance score of a candidate. A node pulls the node vectors of candidates in its host caches only when it chooses a candidate as a new neighbor. Both CSS and GES forward probe query messages using random walk with bounded TTL. The two probe queries in GES can be considered as an equivalent of one probe query, carrying two node vectors.

In CSS, each node also periodically informs its neighbors about updates in its class vectors. Each vector-update message carries the vector differences of changed class vectors. In GES, each node sends periodic messages, notifying its neighbors about the changes in its node vectors. Each vector-update message includes the vector differences of changed node vectors.

Assuming the same probe period, then the topology adaptation in CSS incurs the same number of overhead messages as the topology adaptation in GES. The difference is in the message sizes, mainly caused by the number of vectors carried within the message. In CSS each probe message contains multiple class vectors. In GES each probe message carries equivalently two node vectors. In CSS, each vector-update message transports multiple vector deltas while in GES, only one delta per vector-update message is transported. Assuming the same document collection on the same network, node

vectors and class vectors have the same sizes. In the work for GES[16,25], experiments show that vector sizes of 20 and 50 offer an acceptable performance. Therefore the overhead increase in CSS will not be much more than that in GES. Furthermore, compression techniques can be employed to reduce the overhead. The keep-alive messages can also piggyback vector-update information.

6 Conclusion

In this paper, we extended GES and presented a class-based semantic searching scheme (CSS) in Gnutella-like unstructured P2P systems. CSS exploits a state-of-the-art data clustering algorithm and makes each component class-based. We also designed two different search protocols: CSS(1) and CSS(2). CSS(1) is totally class-based while CSS(2) is partially class-based. The simulation shows that both CSS(1) and CSS(2) are more efficient than GES in all cases. Compared with CSS(2), CSS(1) achieves higher recall and precision with larger message overhead. In summary, CSS is more suitable when the documents on each node are heterogeneous while GES is applicable when the documents are uniform. CSS(1) and CSS(2) have complementary merits. In the future, we will conduct more simulations using larger datasets and different metrics, such as precision@10. We will also conduct extensive simulations that evaluate the topology adaptation overhead of CSS algorithms against other similar schemes such as GES.

References

[1] Li X, Wu J. Searching techniques in peer-to-peer networks. Handbook of Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, CRC Press, 2005, http://www.kiv.zcn.lz/∼ledvina/DHT/p2psurvey.pdf.

[2] Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content addressable network. In *Proc. the 2003 Conf. Applications, Technologies, Architecture, and Protocols for Computer Communications* (*SIGCOMM 2001*), San Diego, USA, August 27-31, 2001, pp.161-172.

[3] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. the 18th IFIP/ACM International Conference on Distributed System Platforms* (*Middleware 2001*), Heidelberg, Germany, November 12-16, 2001, pp.329-350.

[4] Stoica I, Morris R, Nowell D L, Karger D, Kaashoek M, Dabek F, Balakrishnan H. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 2003, 11(1): 17-32.

[5] Yu D, Chen X, Chang Y. An improved P2P model based on Chord. In *Proc. the 6th International Conference on Parallel and Distributed Computing Applications and Technologies* (*PDCAT 2005*), Dalian, China, December 5-8, 2005, pp.807-811.

[6] Xue K, Hongk P, Li J. FS-chord: A new P2P model with fractional steps joining. In *Proc. Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services* (*AICT-ICIW 2006*), Guadeloupe, French Caribbean, February 19-25, 2006.

[7] Cai J, Shao X, Ma W. Ontology driven semantic search over structured P2P network. In *Proc. the 9th International Conference on Hybrid Intelligent Systems* (*HIS 2009*), Shenyang, China, August 12-14, 2009, pp.29-34.

[8] Dragan F, Gardarin G, Yeh L. A semantic layer for publishing and localizing XML data for a P2P XQuery mediator. In *Proc. the 17th International World Wide Web Conference* (*WWW 2008*), Beijing, China, April 21-25, 2008, pp.1105-1106.

[9] Zhu Y, Hu Y. Efficient semantic search on DHT overlays. *Journal of Parallel and Distributed Computing*, 2007, 67(5): 604-616.

[10] Clarke I, Sandberg O, Wiley B, Hong T W. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. the 2000 Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, USA, July 25-26, 2000, pp.46-66.

[11] Manku G S, Bawa M, Raghavan P. Symphony: Distributed hashing in a small world. In *Proc. the 4th USENIX Symposium on Internet Technology and Systems* (*USITS 2003*), Seattle, USA, March 26-28, 2003.

[12] The Gnutella Protocol Specification V0.4. http://www.stanford.edu/class/cs244b/gnutella_protocol_ 0.4.pdf.

[13] Lv Q, Cao P, Cohen E *et al.* Search and replication in unstructured peer-to-peer networks. In *Proc. the 16th ACM International Conference on Supercomputing* (*ACM ICS 2002*), New York, USA, June 22-26, 2002, pp.84-95.

[14] Yang B, Garcia-Molina H. Improving search in peer-to-peer networks. In *Proc. the 22nd IEEE International Conference on Distributed Computing* (*IEEE ICDCS 2002*), Vienna, Austria, July 2-5, 2002.

[15] Crespo A, Garcia-Molina H. Routing indices for peer-to-peer systems. In *Proc. the 22nd International Conference on Distributed Computing Systems* (*IEEE ICDCS 2002*), Vienna, Austria, July 2-5, 2002.

[16] Zhu Y, Yang X, Hu Y. Making search efficient on Gnutella-like P2P systems. In *Proc. the 19th IEEE International Parallel & Distributed Processing Symposium* (*IPDPS 2005*), Denver, USA, April 3-8, 2005.

[17] Faye D, Nachouki G, Valduriez P. Semantic query routing in SenPeer, a P2P data management system. In *Lecture Notes in Computer Science 4658*, Enokido T *et al.* (eds.), Springer-Verlag, 2007, pp.365-374.

[18] Dohnal V, Sedmidubsky J. Query routing mechanisms in self-organizing search systems. In *Proc. the 2nd International Workshop on Similarity Search and Applications* (*SISAP 2009*), Prague, Czech Republic, August 29-30, 2009, pp.132-139.

[19] Haase P, Siebes R, Harmelen F V. Expertise-based peer selection in Peer-to-Peer networks. *Knowledge and Information Systems*, 2008, 15(1): 75-107.

[20] Pirro G, Ruffolo M, Talia D. Advanced semantic search and retrieval in a collaborative peer-to-peer system. In *Proc. the 2008 International Workshop on Content Management and Delivery in Large-Scale Networks* (*UPGRADE-CN 2008*), Boston, USA, June 23-27, 2008, pp.65-72.

[21] Bawa M, Manku G, Raghavan P. SETS: Search enhanced by topic segmentation. In *Proc. the 26th Annual International ACM SIGIR Conference* (*SIGIR 2003*), Toronto, Canada, July 28-August 1, 2003, pp.306-313.

[22] Shen H T, Shu Y, Yu B. Efficient semantic-based content search in P2P network. *IEEE Transactions on Knowledge and Data Engineering*, 2004, 16(7): 813-826.

[23] Zhou Y, Croft W B, Levine B N. Content-based search in

peer-to-peer networks. Technical Report, University of Massachusetts, 2004.
[24] Witschel H F. Content-oriented topology restructuring for search in P2P networks. Technical Report, University of Leipzig, Germany, 2005.
[25] Zhu Y, Hu Y. Enhancing search performance on Gnutella-like P2P systems. *IEEE Transactions on Parallel and Distributed Systems, 2006, 17*(12): 1482-1495.
[26] Yang X, Hu Y. SEIF: Search enhanced by intelligent feedback in unstructured P2P networks. In *Proc. International Conference on Parallel Processing*, Vienna, Austria, September 22-25, 2009, pp.494-501.
[27] Zhong S. Efficient online spherical K-means clustering. In *Proc. IEEE Int. Joint Conf. Neural Networks* (*IJCNN 2005*), Montreal, Canada, July 31-August 4, 2005, pp.3180-3185.
[28] Wang Q, Li R, Chen L, Lian J, Özsu M T. Speed up semantic search in P2P networks. In *Proc. the ACM 17th Conference on Information and Knowledge Management* (*CIKM 2008*), Napa Valley, USA, October 26-30, 2008, pp.1341-1342.
[29] Kacimi M, Yetongnon K. Similarity search in a hybrid overlay P2P network. In *Proc. the 11th IEEE Symposium on Computers and Communications* (*ISCC 2006*), Cagliari, Italy, June 26-29, 2006.
[30] Comito C, Patarin S, Talia D. A semantic overlay network for P2P schema-based data integration. In *Proc. the 11th IEEE Symposium on Computers and Communications* (*ISCC 2006*), Cagliari, Italy, June 26-29, 2006.
[31] Yang X, Hu Y. Search enhanced by distributed semantic clustering in Gnutella-like P2P systems. In *Proc. the 15th International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication* (*MASCOTS 2007*), Istanbul, Turkey, October 24-26, 2007, pp.318-324.
[32] Huang J, Li X, Wu J. A class-based search system in unstructured P2P networks. In *Proc. the 21st International Conference on Advanced Networking and Applications*, Niagara Falls, Canada, May 21-23, 2007, pp.76-83.
[33] Ng C H, Sia K C. Peer clustering and firework query model. In *Proc. the 11th International World Wide Web Conference* (*WWW 2002*), Honolulu, Hawaii, USA, May 7-11, 2002.
[34] Crespo A, Garcia-Molina H. Semantic overlay networks for P2P systems. In *Lecture Notes in Computer Science 3601*, Garbonell J G, Siekmann J (eds.), Springer-Verlag, 2005, pp.1-13.
[35] Lin K, Wang C, Chou C, Golubchik L. SocioNet: A social-based multimedia access system for unstructured P2P networks. *IEEE Transactions on Parallel and Distributed Systems*, 2010, 21(7): 1027-1041.
[36] Deconinck G, Vanthournout K. Agora: A semantic overlay network. *International Journal of Critical Infrastructures*, 2009, 5(1/2): 175-195.
[37] Chawathe Y, Ratnasamy S, Breslau L, Lanham N, Sheaker S. Making gnutella-like P2P systems scalable. In *Proc. the 2003 Conf. Applications, Technologies, Architecture, and Protocols for Computer Communications* (*SIGCOMM 2003*), Karlsruhe, Germany, August 25-29, 2003, pp.407-418.
[38] Berry M W, Drmac Z, Jessup E R. Matrices, vector spaces, and information retrieval. *SIAM Review*, 1999, 41(2): 335-362.
[39] Text REtrieval Conference (TREC). http://trec.nist.gov, May, 2010.
[40] McCallum A K. Rainbow toolkit. http://www.cs.cmu.edu/~mccallum/bow/, May, 2010.

Jun-Cheng Huang received the Bachelor degree in electronic engineering from Fudan University, China in 2004 and the Master's degree in computer engineering from Florida Atlantic University, USA, in 2006. He is currently working in Shanghai Hewlett-Packard Co., Ltd. as a senior software engineer. He used to be a software engineer in Motorola USA division. His research interests lie in distributed computing, routing protocols, mobile computing, wireless and P2P networks. He is also interested in the development of large-scale enterprise software and platform.

Xiu-Qi Li is an assistant professor at Department of Computer Science and Mathematics in University of North Carolina at Pembroke (UNCP), Pembroke, North Carolina, USA. Prior to joining UNCP, she worked as a senior instructor in Florida Atlantic University, Boca Raton, Florida, USA for five years. She earned Faculty Summer Research Fellowship in 2010. She served as program committee member and session chair in 12 conferences. She holds 26 peer-reviewed journal and conference papers. Her research interests include networking, security, multimedia, and web mining. She is a member of ACM and IEEE.

Jie Wu is chair and professor in the Department of Computer and Information Sciences, Temple University. Prior to joining Temple University, he was a program director at National Science Foundation of USA. His research interests include wireless networks and mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. He has published more than 500 papers in various journals and conference proceedings. He serves in the editorial board of the IEEE Transactions on Computers, Journal of Parallel and Distributed Computing, IEEE Transactions on Mobile Computing. Dr. Wu was also general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, and DCOSS 2009. He is program co-chair for IEEE INFOCOM 2011. He has served as an IEEE Computer Society distinguished visitor. Currently, Dr. Wu is the chairman of the IEEE Technical Committee on Distributed Processing (TCDP) and an ACM distinguished speaker. Dr. Wu is a Fellow of the IEEE.