

A dynamic routing protocol for keyword search in unstructured peer-to-peer networks

Cong Shi ^{*}, Dingyi Han, Yuanjie Liu, Shicong Meng, Yong Yu

*APEX Data and Knowledge Management Lab, Department of Computer Science and Engineering,
Shanghai Jiao Tong University, Shanghai 200030, China*

Available online 16 August 2007

Abstract

The idea of building query-oriented routing indices has changed the way of improving keyword search efficiency from the basis as it can learn the content distribution from the query routing process. It gradually improves search efficiency without excessive network overhead for the construction and maintenance of routing indices. However, previously proposed protocol is not practically effective due to the slow improvement of routing efficiency.

In this paper, we propose a novel protocol for query-oriented routing indices which quickly achieves high search efficiency at low cost. The maintenance mechanism employs reinforcement learning to exploit mass peer behavior. It explicitly uses the expected number of returned results to depict the content distribution, which helps quickly approximate the real distribution. The routing mechanism is to retrieve as many contents as possible and help speed up the learning process. To further improve the search efficiency, several methods are taken to optimize the routing and maintenance mechanism. In dealing with multi-keyword queries, the information of corresponding keywords is also used to forward the queries. In addition, to accelerate the learning speed, a rough description of content distribution is achieved when the query is first seen. The experimental evaluation shows that the mechanism achieves high routing efficiency, quick learning ability, and satisfactory performance under churn.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Peer-to-peer; Keyword search; Routing index; Reinforcement learning

1. Introduction

P2P systems like Kazaa [1] and Gnutella V0.6 [2] suffer from their significant network overhead on flooding queries among supernodes (or ultrapeers) [3]. A lot of work has been done to reduce network overhead and enhance query result quality. They can be categorized into two classes, *blind routing* and *routing indices*.

Improvements of flooding [4,5] and random walk [6,15,16] belong to the first type as they retain the “blind” nature of query forwarding (i.e., query routing is independent of the query string). Their merits are the resilience to the dynamics of P2P networks and the dramatic reduction

in network overhead. Compared to them, mechanisms based on routing indices, content-oriented routing indices [3,7,17] and query-oriented one [8], are able to achieve higher efficiency by recording content distribution information.

The mechanism based on query-oriented routing indices utilizes the historical information of queries and query hits to help route future queries. No excessive network overhead is necessary for the construction of routing indices, a primary advantage over content-oriented ones. Previous work [9–11] shows that the popularity of queries fits a Zipf-like distribution in deployed P2P networks, i.e., a small set of query strings represent the majority of the queries. There is a chance that building indices for only those popular queries, instead of all the shared contents, will achieve good routing performance with little network overhead. Therefore, it is a potential mechanism to achieve the merits of both blind routing and routing indices.

^{*} Corresponding author. Tel.: +86 021 54745879 603.

E-mail addresses: cshi@apex.sjtu.edu.cn (C. Shi), handy@apex.sjtu.edu.cn (D. Han), lyjgeorge@apex.sjtu.edu.cn (Y. Liu), bill@apex.sjtu.edu.cn (S. Meng), yyu@apex.sjtu.edu.cn (Y. Yu).

However, previously proposed query-oriented routing index [8] is not practically effective due to the intuitive strategy for the maintenance of routing indices. Without an effective learning method, it only achieves slow improvement in routing efficiency. As mentioned above, the query-oriented routing indices guide the query to the related peers, while the results help maintain the routing indices. The process is similar to the scenario investigated by reinforcement learning [12]. It is highly possible that reinforcement learning is able to quickly enhance the routing efficiency.

In this paper, we propose a learning-based query routing protocol (LBQR) to achieve efficient query-oriented routing indices. LBQR utilizes mass peer behavior to learn the distribution of the contents that peers are interested in. Reinforcement learning is applied to construct and maintain routing indices. A formalized update strategy is adopted to form an accurate depiction of the content distribution. It explicitly approximates the expectation of returned contents, which determines the fast learning ability. An efficient routing strategy is used to route queries, which tries to search as many contents as possible and helps speed up the learning process. Due to the Zipf-like distribution of queries, every peer only needs to keep a small local index. LRU (Least-Recently-Used) strategy is used to replace items when the index is full. The information recorded in LBQR dynamically changes with peers' interest.

In LBQR, every entry for a query string in the routing indices keeps two variants, *weight* and *visit*, per neighbor. Weight depicts the estimation of the number of returned contents when a query is forwarded to the neighbor. Visit records the statistical information of query forwarding, which helps to approximate the expected value of weight. When a peer initiates or receives a query, weight determines the probability that a neighbor is chosen to route the query. When the query hits arrive, they are combined to form the feedback and are used to update weight.

To further improve the search efficiency, some optimization methods are taken to improve the effect of routing and maintenance mechanism. First, we try to improve the search performance of multi-keyword queries. Previous work [13] shows that more than 80% of queries contains two or more keywords. The enhancement of search efficiency of multi-keyword queries will greatly improve the overall search efficiency. As implied above, every query string is treated as a single keyword in LBQR, and there is no collaboration among different entries of the routing indices in query routing. Therefore, we combine the information of every keyword to help route the corresponding queries, which will improve their search efficiency. Second, we want to form a rough description of the content distribution when the query is first seen. LBQR gradually learns the content distribution. If a rough description of the distribution can be got at the very beginning, the learning speed can be accelerated. Therefore, we flood the queries in a small region to achieve the rough description when they are first seen.

Our experimental evaluation shows that the search efficiency of LBQR is similar to that of content-oriented routing indices, and it is times more efficient than previously proposed query-oriented routing indices and “blind” routing mechanisms. LBQR's quick learning ability is also verified, which paves the way for wide-area deployment. LBQR is also resilient to churn, and it manages to converge to a stable state under churn. Moreover, those optimization methods further enhance the search efficiency.

The rest of the paper is organized as follows. The related work is introduced in Section 2. Section 3 presents a detailed description of LBQR's design. The optimization methods are discussed in Section 4. In Section 5, the performance of LBQR is evaluated. We conclude the work in Section 6.

2. Related work

To address the problem of query routing in unstructured P2P networks, many solutions have been proposed. Major improvements can be classified into two categories: blind routing and mechanisms based on routing indices.

Blind routing includes improvements of flooding and random walk with its improvements. In the expanding ring [14], several successive flooding searches are initiated with increasing TTLs until enough responses are received. In LightFlood [5], a tree-like sub-overlay is constructed to minimize the redundant messages and retain the same search scope as flooding. In random walk, a peer randomly chooses a neighbor to forward the query. Its improvements break the equality among peers. In Max-Degree-biased walk [15], a peer forwards queries to the highest-degree neighbor. In Max-Feedback-biased walk [16], a peer forwards queries to the highest-feedback neighbor.

The use of routing indices in P2P networks was proposed, most of which are based on the shared contents. In RI [7], the contents are classified under “topics” and peers index the number of documents under each topic reachable through each neighbor. In one hop replication [17], peers index the contents shared by their neighbors. In SQR [3], Exponential Decay Bloom Filter is designed to encode probabilistic routing tables in a highly compressed manner.

Query-oriented routing indices was first introduced in APS [8]. In this mechanism, each node keeps a local index consisting of one entry for each object it has requested or forwarded a request for, per neighbor. Searching is based on the simultaneous deployment of k walkers and probabilistic forwarding. Every walker terminates with either a success or a failure. Upon walker termination, “optimistic” or “pessimistic” approach is applied to update the index. No extra network overhead is necessary to maintain the routing indices, and the content distribution can be gradually learned. In [18], we propose a learning based query routing mechanism to remedy the demerits of APS. It utilizes mass peer behavior to learn the content distribution during the query routing process.

3. Learning-based query routing

In this section, we describe the design of LBQR in detail. We begin with a brief overview of the solution. Then the query forwarding mechanism is presented. We end this session with a description of the mechanism that constructs and maintains the routing indices of LBQR with reinforcement learning.

3.1. Overview

In LBQR, every peer assigns its connections a set of weights. Each one of the weights is corresponding to a query string. We use connections instead of neighbors, because peers several hops away are taken into account as well as immediate neighbors. Query Q 's weight for connection C at peer A is a measure of how likely A considers that it will get satisfactory results from C after forwarding Q to C .

In LBQR, there are two core mechanisms, query routing and routing indices maintenance. Algorithm 1 illustrates the general procedures of the two mechanisms.

To achieve quick improvement in the routing efficiency, reinforcement learning is used in the maintenance procedure. Besides searching enough satisfactory contents, the query routing mechanism should also adopt proper strategy to speed up the learning process.

Due to the Zipf-like distribution of queries' popularity, LRU strategy is used to replace items when the routing index is full. Therefore, a small query-oriented routing index is sufficient to help achieve high routing efficiency.

With the Zipf distribution of queries in practical P2P systems, LBQR aims at improving the overall performance of query routing by enhancing that of few queries frequently issued in the P2P networks. In case the distribution of queries does not follow the Zipf distribution, the performance of LBQR will be affected to some extent. However, the performance will still be acceptable. In the worst case, i.e., the number of queries is uniformly distributed, every peer can only collect a little information for

every query. In this situation, when a peer receives a query, every connection has almost the same weight and is randomly chosen with nearly the same possibility, which is in the same spirit of random walk. Therefore, even in the worst situation which may never exist in practice LBQR can still obtain similar performance as random walk.

The design of LBQR makes it an effective routing protocol with little overhead. Since the queries and query hits are the only information used to maintain LBQR, no excessive network cost is produced. The experimental evaluation establishes LBQR's high efficiency and quick learning ability, which ensures the practical effectiveness.

3.2. Query forwarding

The query forwarding process determines what the peer initiating the query gets and further affects the maintenance process of LBQR, since the feedbacks of queries are used to update the routing indices. A balance between search and maintenance should be achieved. When a peer initiates or receives a query, it should determine both how to route the query and how many copies of the query to be transmitted.

When peer A is going to transmit k copies of query Q_j , it chooses k connections one after another. In a certain round, suppose A has M candidate connections C_1, C_2, \dots, C_M ($M > 0$) to forward the query (source connection from which the query comes and those connections which have already been chosen are excluded.). $weight(C_i, Q_j)$ indicates query Q_j 's weight that peer A assigns to connection C_i . Then, peer A will choose C_i to route query Q_j with the forwarding probability:

$$\Pr_j(C_i) = \frac{weight(C_i, Q_j)}{\sum_{i=1}^M weight(C_i, Q_j)} \quad (1)$$

If $weight(C_i, Q_j)$ is smaller than a predefined threshold ε , we replace $weight(C_i, Q_j)$ with ε to ensure the positive probabil-

Algorithm 1: The Core Protocol of LBQR

1:	procedure ROUTEQUERY(Q, S)	▷ Q is a query, S is ▷ the source peer, RI is the routing index
2:	if not $RI.exist(query.key)$ then	
3:	$RI.addItem(Q.key)$;	▷ Q 's key is never seen
4:	end if	
5:	if not $Terminate(Q)$ then	
6:	$P \leftarrow RI.Lookup(Q.key)$;	▷ choose a peer
7:	$P.RouteQuery(Q, this)$;	
8:	end if	
9:	$DeliverQueryHit(Search(query.key), S)$;	
10:	end procedure	
1:	procedure UPDATERI($Hits, Q, P$)	▷ utilize the query ▷ hits for Q from peer P to update RI ▷ form the feedback
2:	$F \leftarrow Feedback(Hits)$;	
3:	$RI.updateItem(Q.key, P, F)$;	
4:	end procedure	

ity for every connection to forward queries. The threshold ε is employed due to the mismatch between $weight(C_i, Q_j)$ and the expected value of $weight(C_i, Q_j)$ which is caused by the dynamics of the P2P network.

When a peer receives a query, it first checks the query's id. If the query is a redundant one, it simply drops it.

There are two common strategies used to determine the number of query's copies to be transmitted, forwarding the query to all the connections which is adopted by flooding and choosing one connection to route the query which is adopted by random walk. The former one leads to severe network overhead and is not scalable, while the latter one results in long response time. To overcome those demerits, we adopt a strategy somewhere between the two: the number of copies transmitted by a peer is

$$\#Query = \frac{\alpha}{Hop + 1} \quad (2)$$

where α is a predefined constant, and Hop is the hop distance between current peer and the query originator. The peer initiating the query forwards α copies, while the peers α hop away stop the routing process. Calculating all the copies of a query transmitted, we find that the upper bound of total copies is

$$O \leq \sum_{n=0}^{+\infty} \frac{\alpha^n}{n!} = e^\alpha \quad (3)$$

which is only relevant to the predefined constant α . This strategy has several additional advantages. The distribution of the query's copies is sparse, which helps reduce unnecessary network overhead caused by query redundancy [19,20]. Moreover, it helps the query originators and peers close to them speed up the learning process, since a large subset of their connections are explored per search.

3.3. Routing indices construction and maintenance

The query hits received by the peers on the routing path are combined to form their *feedbacks* for the query. Obviously, the expected value of *feedback* is a good estimation of the distribution of the contents matching a particular query string. We utilize reinforcement learning to gradually approximate the expectation of *feedback*. In addition, churn's impact on LBQR is also under consideration.

After a peer forwards the query copies to some connections, it will get *feedback* from those connections. $f_A(C_i, Q_j)$ indicates the *feedback* for query Q_j that peer A gets from connection C_i . It includes two parts, the query hit returned by immediate neighbor B and the *feedbacks* received by B . The number of contents, Num_B , is used to represent the importance of the query hit returned by B . To peer A , the *feedbacks* received by B are not so determinate as what B responses. Therefore, it times a constant γ ($0 < \gamma < 1$) to

reduce the impact of indeterminacy. Then, $f_A(C_i, Q_j)$ can be expressed by the recurrence:

$$f_A(C_i, Q_j) = Num_B + \gamma \times \sum_{m=0}^M f_B(C_m, Q_j) \quad (4)$$

If Num_n is used to represent the number of contents returned by peers n hops away, the *feedback* can also be stated as:

$$f(C_i, Q_j) = \sum_{n=0}^{+\infty} \gamma^n \times Num_n \quad (5)$$

To calculate $f(C_i, Q_j)$, a timer is started after forwarding a query. The query hits received before time out are taken into account.

As mentioned above, the expected value of *feedback* is a good estimation of content distribution, $weight(C_i, Q_j)$.

$$weight(C_i, Q_j) = E[f(C_i, Q_j)] \quad (6)$$

It is demanding, if not impossible, for peers to get the expectation of *feedback* for a particular query. Our goal is to utilize mass peer behavior to gradually approximate the expected value. Let $weight_n(C_i, Q_j)$ represent the estimation of $weight(C_i, Q_j)$ when a peer has forwarded Q_j to C_i for n times. The maintenance strategy is described as follows:

$$weight_n(C_i, Q_j) = \alpha_n \times weight_{n-1}(C_i, Q_j) + (1 - \alpha_n) \times f_n(C_i, Q_j) \quad (7)$$

$$\alpha_n = \frac{visit_{n-1}(C_i, Q_j)}{visit_n(C_i, Q_j)}, \quad visit_0(C_i, Q_j) = 0$$

where $visit_n(C_i, Q_j)$ records the times that Q_j has been routed to C_i . As shown in Eq. 7, $weight_n(C_i, Q_j)$ is the combination of previous estimation and current *feedback*. In fact, when Q_j is seen for the first time, $\alpha_1 = 0$, and $weight_1(C_i, Q_j) = f_1(C_i, Q_j)$. Therefore, the creation and maintenance are actually the same process.

The maintenance of $visit_n(C_i, Q_j)$ should also be determined. A simple strategy is to set $visit_n(C_i, Q_j) = n$. However, the peer originating the query will get more query hits from the connections, while peers several hops away from the originator will get fewer query hits. It is not fair to treat these two cases equally. In fact, as shown in Eq. 7, $visit_n(C_i, Q_j)$ reflects the importance of $weight_n(C_i, Q_j)$. Therefore, it is much more proper to use the number of forwarded query copies to measure the importance of *feedback*. Suppose peer A is Hop_n hop away from the originator. According to Eq. 2, the maintenance strategy for $visit_n(C_i, Q_j)$ can be expressed as:

$$visit_n(C_i, Q_j) = visit_{n-1}(C_i, Q_j) + \frac{1}{Hop_n + 1} \quad (8)$$

It can be proved that $weight_n(C_i, Q_j)$ converges to $weight(C_i, Q_j)$ as $n \rightarrow \infty$. Therefore, we can conclude that the maintenance strategy is reasonable.

Theorem 1. Let C_i be a connection of the peer in question. Let Q_j be a query string. Let $weight(C_i, Q_j)$ be the expectation of C_i 's feedback for Q_j . Let $weight_n(C_i, Q_j)$ be the estimation of $weight(C_i, Q_j)$ when queries with string Q_j have been forwarded to C_i by current peer for n times. Then for all C_i and Q_j , $weight_n(C_i, Q_j) \rightarrow weight(C_i, Q_j)$ as $n \rightarrow \infty$, with probability 1.

Proof. Eq. 7 shows the recurrence form of $weight_n(C_i, Q_j)$. We use P_k to indicate $\frac{visit_k - visit_{k-1}}{visit_n}$, where $visit_0 = 0$. In fact, P_k indicates the relative importance of the k th query's feedback compared to the total feedbacks. Therefore, $weight_n(C_i, Q_j)$ can also be expressed as:

$$weight_n(C_i, Q_j) = \sum_{k=1}^n P_k \times f_k(C_i, Q_j), \quad \sum_{k=1}^n P_k = 1$$

According to the law of large numbers, $\sum_{k=1}^n P_k \times f_k(C_i, Q_j) \rightarrow E[f(C_i, Q_j)]$ as $n \rightarrow \infty$, with probability 1. Replace $\sum_{k=1}^n P_k \times f_k(C_i, Q_j)$ with $weight_n(C_i, Q_j)$, and replace $E[f(C_i, Q_j)]$ with $weight(C_i, Q_j)$. Therefore, $weight_n(C_i, Q_j) \rightarrow weight(C_i, Q_j)$ as $n \rightarrow \infty$, with probability 1. \square

Churn is a severe problem in P2P networks. When a peer departs from the network, the routing indices kept by other peers are impacted and should be updated. In P2P networks with content-oriented routing indices used, information is frequently exchanged among peers to update routing indices, which causes serious network overhead [3]. LBQR is dynamically maintained during the query routing process, therefore, no excessive network overhead is consumed. However, the routing indices will fail to accurately depict the distribution of contents if the network dramatically changes.

It is rational to suppose that the P2P network topology does not change much in a short time. Therefore, we set a time threshold, τ . When $weight_n(C_i, Q_j)$ is to be updated, we only take into consideration the part of $weight_n(C_i, Q_j)$ received within τ time slot. Therefore, the routing indices indicates the estimation of content distribution within a short time, close to the real distribution.

4. Optimization

Our goal is to improve the search efficiency and accelerate the learning speed. We notice that the information of corresponding keywords in multi-keyword queries is not taken into account. The utilization of the information may enhance the search efficiency of multi-keyword queries. In addition, the content distribution can not be obtained when only a few queries are issued, which impacts the search efficiency of these queries. If a rough description of the content distribution is achieved at the beginning of learning process, LBQR will be even effective to unpopular queries. Therefore, the optimization methods focus on these two aspects.

4.1. Multi-keyword queries

Previously work [13] shows that the average number of keywords per query is 3.74 in Gnutella. There are only about 18% single keyword queries. In LBQR, multi-keyword queries are treated the same as the single keyword queries. However, if the extra information of multi-keyword queries can be utilized, it will greatly enhance the average search efficiency. In dealing with the multi-keyword queries, the information of every keyword in the query can be used to help routing the query and achieving the content distribution for the query.

Suppose query Q is composed of keywords K_1, K_2, \dots, K_n which are independent from each other. $weight(C_i, K_j)$ indicates the weight of K_j for connection C_i recorded in the indices. Then the combined distribution can be expressed as:

$$Pr_{cd,Q}(C_i) = \frac{\prod_{j=1}^n weight(C_i, K_j)}{\sum_{i=1}^M \prod_{j=1}^n weight(C_i, K_j)} \quad (9)$$

It is highly possible that some keywords have not been seen, namely, there is no information about them. Their weights are replaced with predefined threshold, ε . It should also be considered how to utilize the information of multi-keyword queries. For example, a peer has information about three single keywords (K_1, K_2 , and K_3) and a multi-keyword query Q (composed of K_1 and K_2). When it is to route a 3-keyword query (composed of K_1, K_2 , and K_3), it should determine how to combine the information. In this case, the information of K_1, K_2 , and Q will be combined first as follows:

$$weight_Q(C_i) = \beta \times weight(C_i, Q) + (1 - \beta) \times \prod_{j=1}^2 weight(C_i, K_j)$$

$$\beta = \frac{visit(C_i, Q)}{visit(C_i, Q) + \min(visit(C_i, K_1), visit(C_i, K_2))}$$

Then, $weight_Q(C_i)$ and $weight(C_i, K_3)$ will be combined as shown in Eq. 9.

The accuracy of combined distribution, $Pr_{cd,Q}(C_i)$, is determined by that of every keyword. It can be depicted by $\min(visit(C_i, K_1), visit(C_i, K_2), \dots, visit(C_i, K_n))$. Combining the content distribution for query string Q and above-achieved combined distribution, we get the forwarding probability for multi-keyword queries:

$$Pr_{m,Q}(C_i) = \beta \times Pr_Q(C_i) + (1 - \beta) \times Pr_{cd,Q}(C_i) \quad (10)$$

$$\beta = \frac{visit(C_i, Q)}{visit(C_i, Q) + \min(visit(C_i, K_1), \dots, visit(C_i, K_n))}$$

When the query hits are received, $weight(C_i, Q)$ and $visit(C_i, Q)$ are updated. We do not update the $weight$ and $visit$ value for the corresponding keywords with the feedback of multi-keyword query, since it is almost impossible to depict

single keyword's distribution with that of multi-keyword queries. Therefore, it helps enhance the search efficiency of multi-keyword queries and have no impact on other queries.

4.2. Rough description of content distribution

LBQR gradually learns the distribution of contents which are of interest to peers, and the search efficiency is improved with the learning process. Therefore, when only a few queries are issued, it fails to achieve high search efficiency. However, if we can get a rough description of content distribution when the query is first seen, high search efficiency can be achieved even if the query is not very popular.

If a query is flooded among peers, the accurate distribution of contents matching the query can be achieved. Moreover, the larger the region is covered, the more accurate the distribution will be. However, it will cause severe network overhead if the query is flooded in a large region. Our goal is to get a rough description of content distribution. The content distribution within a small region is enough. Therefore, the optimized strategy, f-LBQR, is as follows: once a query is first seen to a peer, it floods the query within two hops. Then, the content distribution within the region is gained. f-LBQR helps peers to achieve the rough description of content distribution with only a little extra network overhead.

5. Evaluation

In this section, we present simulation-based evaluation of LBQR and compare its performance with three representative mechanisms (random walk [6], SQR [3], and APS [8]) which, respectively, belong to the two categories mentioned above. Then, the characteristic of LBQR is analyzed. At the end of section, the impact of optimization methods on LBQR is accessed.

5.1. Methodology

5.1.1. Performance metrics

A good routing mechanism should try to optimize both efficiency and Quality of Service (QoS). Efficiency focuses on the resources utilization, such as bandwidth, while QoS focuses on user-perceived qualities, such as the number of results returned. More specifically, we use the following metrics:

Response rate is the ratio of returned contents to the distinct peers receiving the query. It takes both QoS and efficiency into account. The higher the *response rate* is, the more effective the routing mechanism will be.

Recall is the ratio of returned contents to the total contents matching the query in the P2P system. It helps assess the QoS of LBQR. The more results are returned, the more choice peers will have when downloading the content. In

addition, *recall* helps to evaluate LBQR's routing performance with various content popularity.

Query redundancy is the number of redundant queries that peers on the routing path received per searching per peer. Generally, the redundant query messages a peer receives in a certain routing process have little help to find more contents, and lead to unnecessary network overhead.

Convergence. The query-oriented routing indices evolve with the query routing process. According to [Theorem 1](#), the weights recorded in LBQR will gradually converge to stable values. *Convergence* depicts the stability of the LBQR. In addition, it also exhibits the learning efficiency of query-oriented routing indices. The quicker the *convergence* is, the more valuable the routing indices will be when practically deployed.

Churn seriously affects the performance and maintenance of routing indices. When measuring *churn*'s impact on the performance of routing mechanism, the effect of content popularity should be avoided. In simulation, the peers leaving the network immediately join the network by connecting some other peers, which avoids the variation of contents. Therefore, we define *churn* as the number of peers leaving and joining the network per time slot.

5.1.2. Topologies

To be practically deployed to P2P networks, the performance of LBQR in various P2P topologies should be evaluated. Many popular P2P systems, such as Kazaa and Gnutella V0.6, employ hierarchical topologies which make them more scalable. In these networks, supernodes with good network connectivity are responsible for the primary work of query routing. Therefore, we focus on the topology of the supernode layer. Typically, we use a 2000 peer network. The following two canonical topologies are used to evaluate the routing mechanism.

Power law graph. The power law distribution is a popular phenomenon in complex networks. Freely evolving P2P networks have been shown to exhibit power-law network characteristics [21]. Hence we organize peers into a power-law network. When joining the network, peers choose 5 nodes to connect. Peers connect to a node i with probability $\frac{d_i}{\sum_{j \in N} d_j}$, where N is the set of nodes currently in the network and d_i is the node degree of peer i , which yields a power-law network [22].

Random regular graph. Previous work [23] has shown that the random regular graphs have good properties, including low diameter, good connectivity, small second eigenvalue, and good conductance. Random regular graph is used as a canonical model of a well connected network. Upon joining the network, peers randomly choose to connect 5 peers whose degree is not more than the upper bound of peer degree, 7.

5.1.3. Baseline

The goal of LBQR is to achieve the robustness of blind routing and the efficiency of content-oriented routing

indices. It should overcome the drawback of previously proposed query-oriented routing indices and be practically viable. Therefore, to assess the performance of LBQR, we choose random walk [6], SQR [3], and APS [8] as baselines, which are canonical or well-performed mechanisms of these categories. The set of mechanisms we study is by no means complete, but our choices are representative of the body of prior work in this area and cover both existing systems as well as recent ones proposed in the research literature.

5.1.4. Other details of experiments

In practical P2P systems, the final performance of keyword search is also affected by the information retrieval methods used, the contents, and the query strings, which is out of the scope of this paper. To avoid the impact of these factors on the evaluation of LBQR, we use synthetic contents and queries. Each content is assigned a unique ID. Every query uses the content ID to indicate which content it searches. The distribution of contents and queries with different ID accords with the data we crawled from Gnutella [10].

In the experiments, one peer is randomly chosen to initiate a query per time slot. There are 10,000 queries issued, and the mean value is used to evaluate those routing mechanisms. To compare the performance of various mechanisms (LBQR, random walk, SQR, and APS), the number of peers receiving the query (namely, coverage) is set to be the same.

5.2. Experiment result

5.2.1. Routing performance

We first evaluate the performance of LBQR and compare it with that of random walk (RW), SQR, and APS. The parameters of SQR are the same as those used in [3]. In the implementation of APS, 10 walkers are employed, which is quite enough due to the low peer degree.

Two parameters directly influence the routing performance. One is the predefined constant α in Eq. 2 which determines the coverage. The other is the popularity of contents queried. In the first set of experiments, we randomly distribute 100 identical contents among the peers, and vary the constant α (i.e., 5, 6, 7, 8, 9, 10). Figs. 1 and 2 show the experiment results. In the second set of experiments, we set α to be 6, and vary the number of contents shared in the P2P network (i.e., 10, 20, 50, 100). Figs. 3 and 4 exhibit the experiment results.

5.2.1.1. Various α value. As shown in Figs. 1 and 2(a), LBQR's response rate is almost constant. It slightly reduces with the increment of α . The other three mechanisms exhibit the similar phenomenon. Comparing response rate of those routing mechanisms, we find that LBQR achieves similar performance as SQR, a little better than APS, and about three times better than random walk.

As shown in Figs. 1 and 2(b), LBQR's recall increases as α grows. Meanwhile, APS achieves extremely low recall value because of the routing strategy that its walkers terminate when a content is found or a peer receive a redundant query.

Next, we analyze query redundancy which impacts the network overhead during the query routing process. Among the three baselines, APS achieves the best query redundancy, which is also determined by its routing strategy. However, considering its low recall, the good query redundancy is less meaningful. When α is small, LBQR's query redundancy approximates to 0, the ideal value. It even performs better than APS. When α is large, it is still much better than RW and SQR. LBQR's low query redundancy is caused by Eq. 2. As the distance between current peer and the initiator increases, the number of query's copies forwarded declines. Thus, the query redundancy is reduced.

To choose a proper α value, we should consider both recall and the number of messages transmitted (i.e., $n = Coverage \times (1 + QueryRedundancy)$). Large α value leads to both high recall and large message overhead. Typically, α should be set to be 6 or 7, in which case the network overhead is low, and recall is acceptable.

5.2.1.2. Various content popularity. The impact of content popularity on routing performance is shown in Figs. 3 and 4. LBQR's recall is generally constant despite of the content popularity variation. Its response rate is related to the content popularity as well as recall value. Therefore, in the case when recall is almost constant, it is proportional to the content popularity. The impact of content popularity on LBQR's query redundancy is small, which indicates that it is only related to α .

We also notice that LBQR's recall varies more notably in power law topology than in random regular topology. In power law topology, the peer degree varies dramatically. Peers with high degree tend to receive more queries [14]. Therefore, the content distribution affects the recall value.

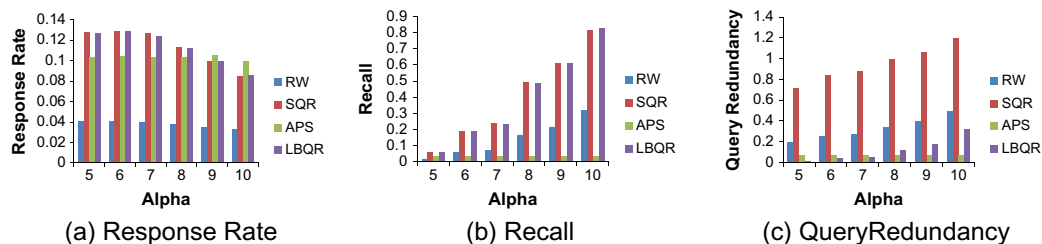


Fig. 1. Routing performance with various α values in random regular graph.

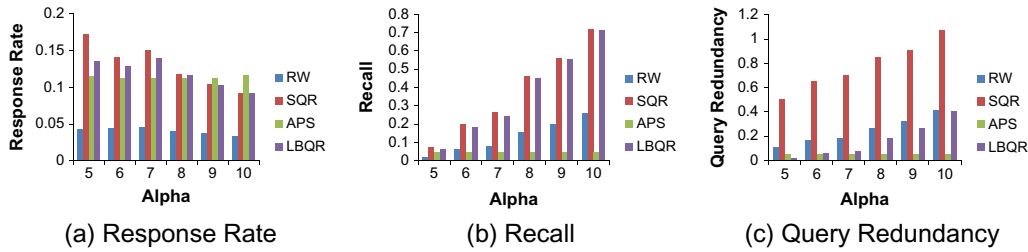
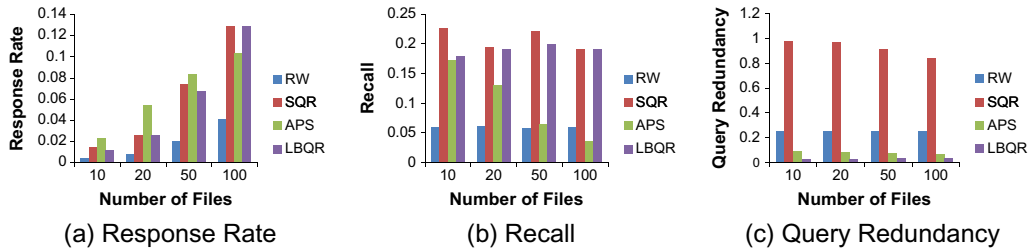
Fig. 2. Routing performance with various α values in power law graph.

Fig. 3. Routing performance with various content popularity in random regular graph.

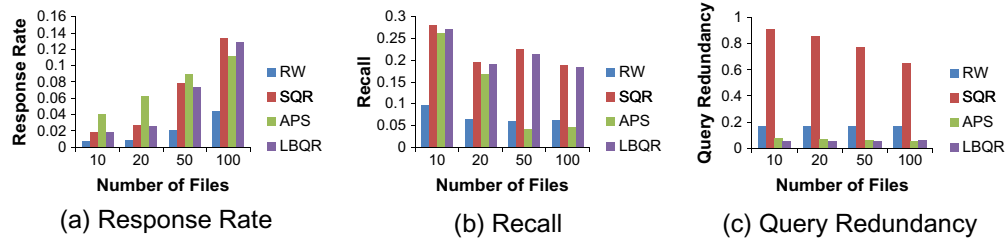


Fig. 4. Routing performance with various content popularity in power law graph.

When α is set to be 6, and content popularity varies from 10 to 100, SQR always has the highest recall value among the three baselines. LBQR manages to achieve comparable recall as SQR and produce lowest query redundancy.

5.2.1.3. Learning rate. It seems that there is no big difference between the performance of LBQR and that of APS according to the results shown in the four figures. However, to be practically effective, the mechanisms should quickly achieve high efficiency. Fig. 5 demonstrates the learning rate of LBQR and APS when $\alpha = 6$ and the number of contents is 100. It shows that LBQR's response rate dramatically increases when a few queries are issued, and it quickly arrives at a stable state. At the meantime, APS's response rate is almost proportional to the number of queries issued. Although it achieves similar performance as LBQR when 10,000 queries are issued, it fails to quickly improve the efficiency with fewer queries issued. Experiments with various parameter values exhibit similar result. The high learning rate ensures LBQR's practical deployment, and it is LBQR's primary advantage over APS.

5.2.2. Characteristic of LBQR

The characteristic of LBQR is also analyzed. We focus on LBQR's convergence and its performance under churn.

5.2.2.1. Convergence. We apply reinforcement learning to the construction and maintenance of LBQR's routing indices. Therefore, LBQR evolves with the query routing process. The above analysis shows that the performance of LBQR converges to a stable state. We want to verify the convergence and to figure out the factors affecting the convergence. We vary the predefined constant α and the content popularity as conducted in above experiments. The results are demonstrated in Fig. 6. The X-axis is the number of queries issued (or times of learning). The Y-axis is the number of contents returned.

As shown in Fig. 6(a) and (b), the experiments with various α values (i.e., 5, 6, 7, 8, 9, 10) have all reached a stable state when several hundreds of queries are issued. LBQR with large α value converges more quickly than those with small ones. For example, when α is set to be 10, LBQR converges when 100 queries are issued. The larger α is, the more peers receive the query, which helps speed up the learning process. Fig. 6(c) and (d) show that the content popularity does not affect the convergence remarkably.

In unstructured P2P systems like Gnutella and Kazaa, supernodes are responsible for routing all the queries issued. In addition, when downloading a certain content, peers will automatically issue queries to find more content

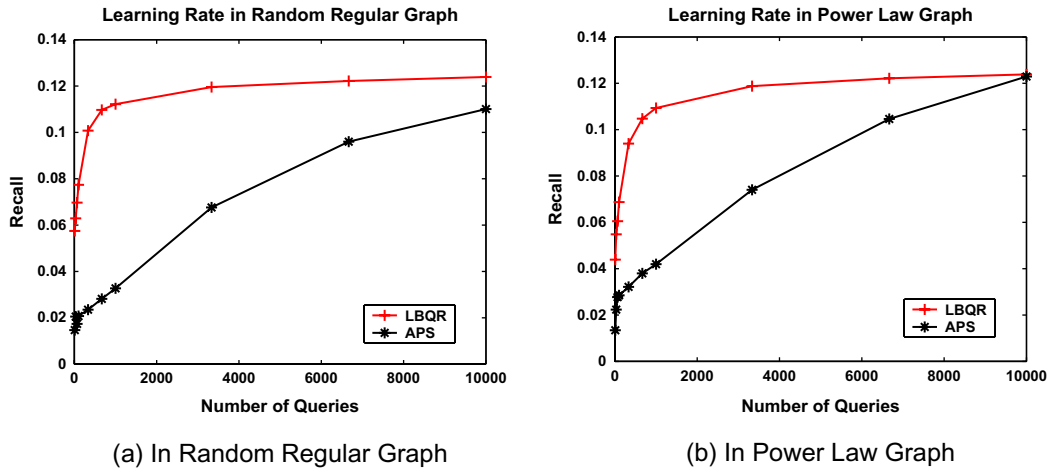


Fig. 5. Compare of learning rate.

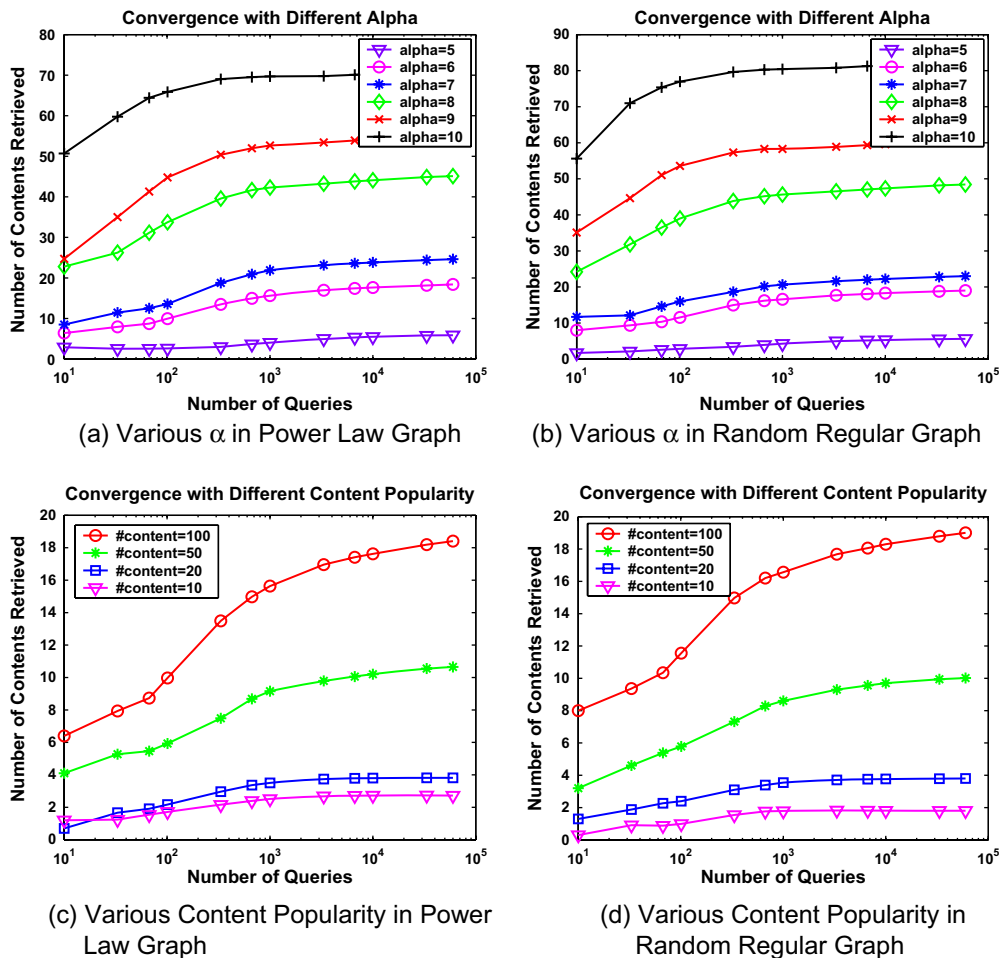


Fig. 6. Convergence of LBQR.

providers. These mechanisms will result in the frequent presence of queries with identical query string. They will speed up the learning process and also validate the feasibility of LBQR.

5.2.2.2. Performance under churn. Churn is a serious phenomenon in P2P networks. In this subsection, we evaluate churn's impact on the performance of LBQR. In the experiments, we randomly distribute 100 contents among peers.

The predefined constant, α , is set to be 6. LBQR’s performance is evaluated under different churn (i.e., 1, 5, 10, 20). The convergence of LBQR under churn is also assessed.

Fig. 7 shows the performance of LBQR under churn in two network topologies. As churn increases, LBQR’s recall declines. The impacts are similar in both networks. Using $\frac{recall_1 - recall_2}{churn_1 - churn_2}$ to analyze the trend of churn’s effect, we find that the ratio declines with the increment of churn (i.e., average: 0.56, 0.29, 0.15). The performance of LBQR under churn is satisfactory. Even under churn 20, it retrieves twice more contents than random walk. The churn dramatically increases the network overhead of SQR to ensure the correctness of routing indices. However, it imposes no burden on LBQR. Considering both recall and network overhead, LBQR’s performance under churn is acceptable.

The convergence of LBQR under churn is also analyzed. Fig. 8 shows the convergence process under various churn. LBQR reaches stable states in all these experiments. It indicates that the churn does not affect LBQR’s convergence.

We conclude that LBQR performs satisfactorily under churn, though churn impacts a little on its performance.

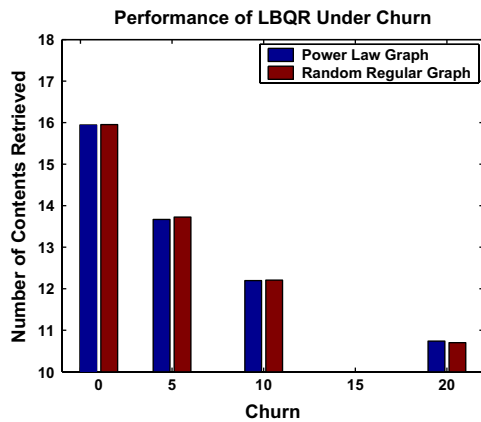


Fig. 7. Performance of LBQR under churn.

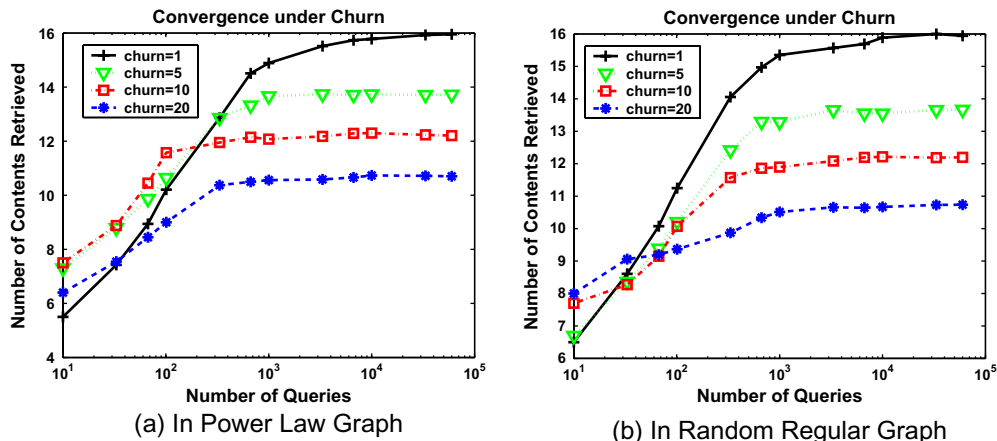


Fig. 8. Convergence of LBQR under churn

5.2.3. Optimization

5.2.3.1. Multi-keyword queries. To enhance the search efficiency for multi-keyword search, the information of every single keyword is also used to help route the query. Previous work [13] shows that about 50% of the queries contained two or three terms. Therefore, we assess the optimization method’s impact on 2-keyword queries and 3-keyword queries.

In the first set of experiments, α is set to be 6. A query with two keywords, A and B , is employed. 40 contents are only related to A , 30 contents are only related to B , and 30 contents are related to both A and B . The total 100 contents are randomly stored in 100 distinct peers. The choice of content popularity ensures that $P(A) \times P(B) \neq P(A \cap B)$ and $P(A) \neq P(B) \neq P(A \cap B)$, which indicates that A and B are neither totally independent nor directly related, a common case to multi-keyword queries. The query popularity is proportional to the corresponding content popularity, i.e., 40%, 30%, and 30%. To analyze the effect of the optimization method, it is compared with that of single keyword queries with the same content distribution.

As mentioned above, the optimization methods only impact the performance of the multi-keyword queries. Therefore, in this set of experiments we only analyze the recall value of the 2-keyword query. Fig. 9 shows the results. In both networks, the recall of multi-keyword query is larger than that of single keyword query from the very beginning. The curves of 2-keyword queries and those of single keyword queries are similar. It indicates that the optimization method does not seriously affect the learning process of LBQR. Its primary effect is to increase the recall value. In addition, the improvement is more obvious in power law graph than in random regular graph. In the power law overlay network, peers with high peer degree have more impact on search than those with low peer degree. Therefore, the adjustment caused by the optimization methods is amplified by them, and better performance is achieved.

An interesting phenomenon is that it takes 2-keyword queries less time to arrive at the stable state than the single

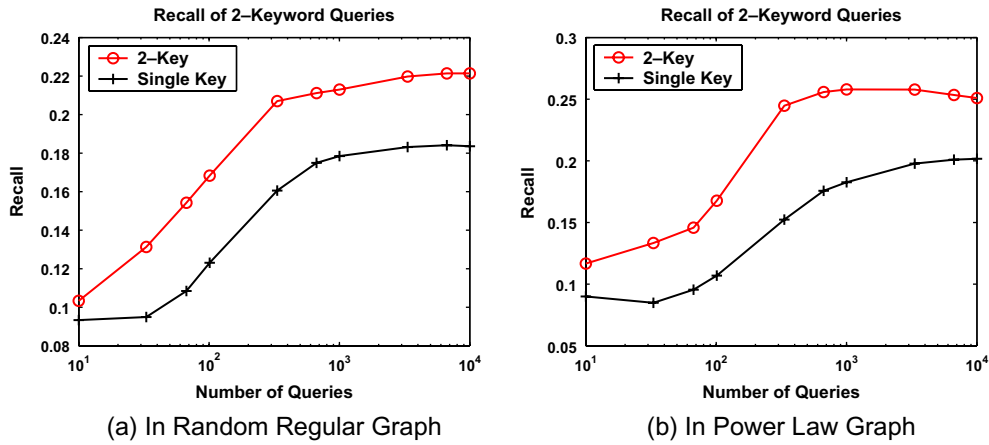


Fig. 9. Recall of 2-keyword queries.

keyword queries. As shown in Fig. 9, the recall of single keyword queries converges when about 700 queries are issued, while that of multi-keyword queries manages to converge when only 300 queries are issued. The combination of various information used in query routing helps multi-keyword queries achieve stable performance quicker.

The first set of experiments exhibit the optimization method’s impact on 2-keyword queries. We wonder whether the method is effective to queries with various keywords, or it is only effective to 2-keyword queries. Therefore, the second sets of experiments is conducted, in which a query with three keywords, *A*, *B*, and *C*, is used. 50 contents are only related to *A*, 60 contents are only related to *B*, 65 contents are only related to *C*, and 25 contents are related to *A*, *B*, and *C*. The query popularity is proportional to the content popularity. As the first set of experiments, we compare the recall of 3-keyword queries and single keyword queries with same content popularity.

The experiment results are shown in Fig. 10. 3-Keyword queries achieve higher recall than single keyword queries. The stable recall value of 3-keyword queries is also higher than 2-keyword queries. More information can be used to route 3-keyword queries than 2-keyword queries and single keyword queries, which leads to the experiment results. It

seems that the more keywords a query contains, the higher recall it will be. However, it is the case only when all the keywords are very popular. When some of the keywords are never seen, less information can be utilized. Nevertheless, the optimization method is able to utilize the extra information of every keyword to enhance the search efficiency.

The two sets of experiments verify the effectiveness of the optimization methods. With the optimization method, higher search efficiency is achieved with no extra burden on the system.

5.2.3.2. *Rough description of content distribution.* To achieve a rough description of content distribution, every peer floods the query in a small region when it is first seen (namely f-LBQR). As shown in above experiments, it takes LBQR a period of time to achieve stable, well-performed search efficiency. If the search efficiency can attain higher value, LBQR will be more effective even to unpopular queries. In the experiments, α is set to be 6. One hundred identical contents are randomly distributed among peers. When the query is flooded, more content matching the query can be retrieved as more peers receive the query. To compare

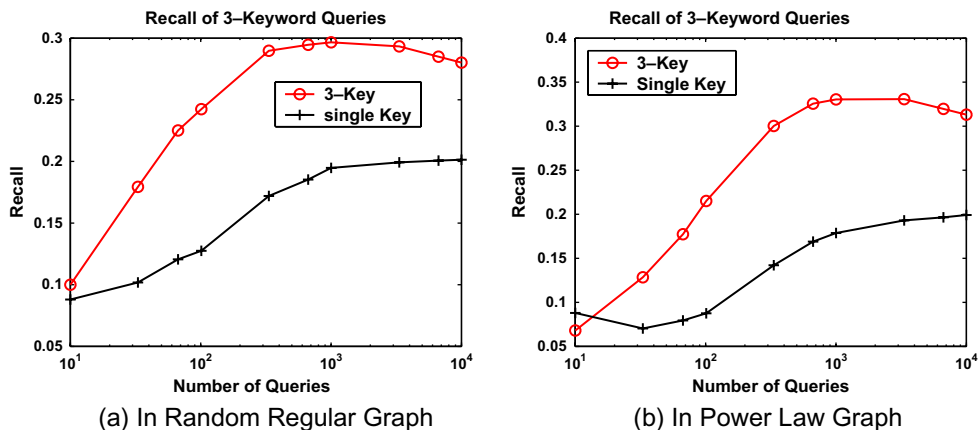


Fig. 10. Recall of 3-keyword queries.

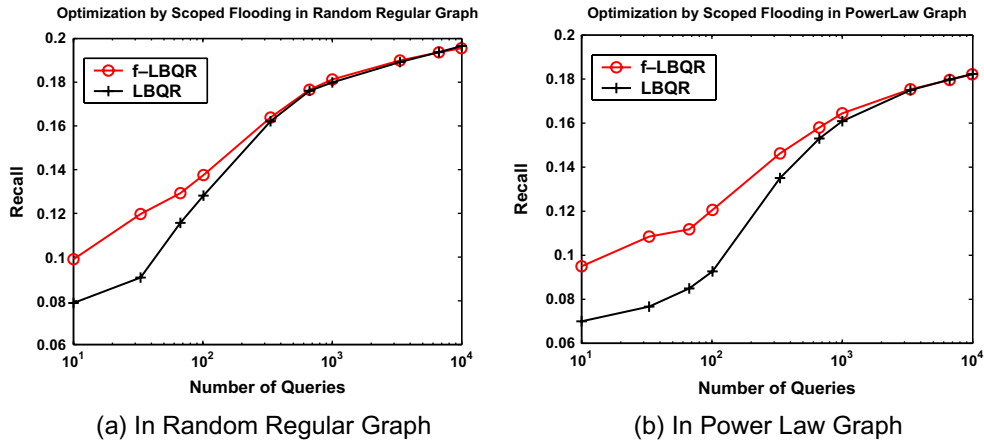


Fig. 11. Recall of single keyword queries with rough description of content distribution.

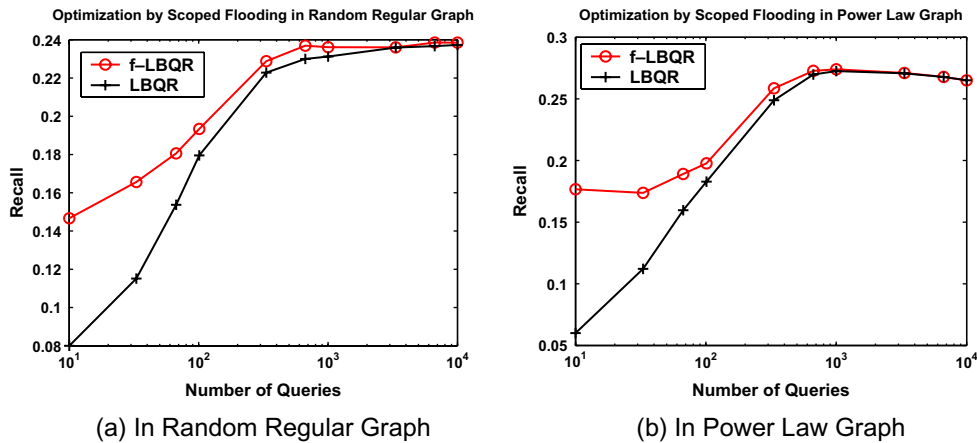


Fig. 12. Recall of multi-keyword queries with rough description of content distribution.

the effect of f-LBQR and LBQR, we do not take the results of flooded queries into account.

We first analyze f-LBQR's impact on single keyword queries. Fig. 11 shows the experiment results in the two networks. In the very beginning, f-LBQR performs better than LBQR. When only 10 queries are issued, the recall of f-LBQR is 0.1, which is about two times higher than that of random walk, and is about half of the stable value. It indicates that f-LBQR ensures the high search efficiency at the beginning of the learning process. The recall values of f-LBQR and LBQR converges to the same stable values with the learning process. The results shows that the effect of f-LBQR is to enhance the initial search efficiency, which is exactly what it is expected to be. Compare Fig. 11(a), f-LBQR is more effective in power law graph and in random regular graph. The principle behind the phenomenon is that the a priori information of content distribution has more impact on the peers with high peer degree than those with low peer degree.

Next, we analyze f-LBQR's impact on multi-keyword queries. In this set of experiments, the two optimization methods are both used. We wonder whether f-LBQR can further improve the search efficiency of multi-keyword queries.

In the experiments, α is set to be 6. A query with two keywords, A and B , is employed. 40 contents are only related to A , 30 contents are only related to B , and 30 contents are related to both A and B . Fig. 12 shows the experiment results.

f-LBQR's impact on multi-keyword queries is similar to that on single keyword queries. f-LBQR's recall is higher than LBQR's one in the beginning. When only 10 queries are issued in power-law network, f-LBQR's recall is about 0.18, approximating the stable recall value of single keyword queries. It achieves similar performance in random regular network as well. The recall values of f-LBQR and LBQR converges to the same stable values with the learning process.

6. Conclusion

In this paper, we present a novel protocol, LBQR, to achieve practically effective query-oriented routing indices in unstructured P2P networks. Reinforcement learning is applied to construct and maintain the query-oriented routing indices.

LBQR has a lot of advantages over previously proposed routing mechanisms. First, the mechanism manages to quickly achieve high routing efficiency, which ensures its

practical deployment. Second, no excessive network overhead is consumed to construct and maintain the routing indices. Due to the dynamic nature of P2P networks, content-oriented routing indices should be frequently updated, which results in huge network overhead. However, we utilize mass peer behavior to maintain the routing indices, and no excessive network overhead is necessary. Third, the construction of routing indices is based on what peers search, instead of what peers share. Therefore, only a little memory is used to record the routing information of contents which are of little interest to peers in the network. Last but not least, experimental evaluation shows that the query-oriented routing indices are resilient to churn. Although the routing performance declines to some extent under churn, it still converges to a satisfactory stable state.

Furthermore, we proposed two methods to further optimize LBQR. The strategy for multi-keyword queries helps enhance the search efficiency. Scoped flooding is used to form a rough description of content distribution in the beginning of learning process. It helps to improve the initial search efficiency, which makes LBQR effective to unpopular queries. These optimization methods impose little or only a little burden on the P2P systems, while they greatly enhance the search efficiency.

In short, learning-based query routing is novel and effective for keyword search in unstructured P2P networks.

References

- [1] Kazaa, <http://www.kazaa.com>.
- [2] Gnutella, <http://www.gnutella.com>.
- [3] A. Kumar, J.J. Xu, E.W. Zegura, Efficient and scalable query routing for unstructured peer-to-peer networks, in: 24th Annual IEEE Conference on Computer Communications (INFOCOM), 2005.
- [4] S. Saroiu, P.K. Gummadi, S.D. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking, 2002.
- [5] S. Jiang, L. Guo, X. Zhang, Lightflood: an efficient flooding scheme for file search in unstructured peer-to-peer systems, in: 32nd International Conference on Parallel Processing (ICPP), 2003.
- [6] C. Gkantsidis, M. Mihail, A. Saberi, Random walks in peer-to-peer networks, in: 23th Annual IEEE Conference on Computer Communications (INFOCOM), 2004.
- [7] A. Crespo, H. Garcia-Molina, Routing indices for peer-to-peer systems, in: The 22th International Conference on Distributed Computing Systems (ICDCS), 2002.
- [8] D. Tsoumakos, N. Roussopoulos, Adaptive probabilistic search for peer-to-peer networks, in: The third IEEE International Conference on Peer-to-Peer Computing (P2P), 2003.
- [9] A. Klemm, C. Lindemann, O.P. Waldhorst, Relating query popularity and file replication in the gnutella peer-to-peer network, in: 11th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB), 2004.
- [10] S. Meng, C. Shi, X. Zhu, Y. Yu, A statistical study of today's gnutella, in: The Eighth Asia Pacific Web Conference (APWeb), 2006.
- [11] K. Sripanidkulchai, The popularity of gnutella queries and its implications on scalability, Featured on O'Reilly's www.openp2p.com website (February 2001).
- [12] T.M. Mitchell, Machine Learning, McGraw-Hill, New York, 1997.
- [13] S.H. Kwok, C.C. Yang, Searching the peer-to-peer networks: the community and their queries, Journal of the American Society for Information Science and Technology 55 (9) (2004) 783–793.
- [14] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: The 16th Annual International Conference on Supercomputing (ICS), 2002.
- [15] L.A. Adamic, R.M. Lukose, A.R. Puniyani, B.A. Huberman, Search in power-law networks, Physical Review E 64 (4) (2001) 046135.
- [16] B. Yang, H. Garcia-Molina, Efficient search in peer-to-peer networks, in: The 22th International Conference on Distributed Computing Systems (ICDCS), 2002.
- [17] Y. Chawathe, S. Ratnasamy, L. Breslau, Making gnutella-like p2p systems scalable, in: ACM Special Interest Group on Data Communications (SIGCOMM), 2003.
- [18] C. Shi, S. Meng, Y. Liu, D. Han, Y. Yu, Reinforcement learning for query-oriented routing indices in unstructured peer-to-peer networks, in: The sixth IEEE International Conference on Peer-to-Peer Computing (P2P), 2006.
- [19] Y. Liu, L. Xiao, L.M. Ni, Building a scalable bipartite p2p overlay network, in: IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2004.
- [20] Y. Liu, Z. Zhuang, L. Xiao, L.M. Ni, A distributed approach to solving overlay mismatching problem, in: The 24th International Conference on Distributed Computing Systems (ICDCS), 2004.
- [21] M. Ripeanu, I. Foster, Mapping the gnutella network: macroscopic properties of large-scale peer-to-peer systems, in: First International Workshop on Peer-to-Peer Systems (IPTPS), 2002.
- [22] A. Medina, I. Matta, J. Byers, On the origin of power laws in internet topologies, in: Technical report, Boston University Computer Science Department, 2000.
- [23] C. Gkantsidis, M. Mihail, A. Saberi, Hybrid search schemes for unstructured peer-to-peer networks, in: 24th Annual IEEE Conference on Computer Communications (INFOCOM), 2005.



Cong Shi received the B.S. degree in Computer Science from Shanghai Jiaotong University, PR China, in 2005. He is currently working towards a M.S. degree in Computer Science at the same university. His research interests include P2P networks, mobile networks and distributed systems.



Dingyi Han is currently an assistant professor at the faculty of Computer Science & Engineering, Shanghai Jiaotong University, PR China. He earned a B.S. degree in Computer Science in 2002 and a Ph.D. in Computer Science in 2007 from Shanghai Jiaotong University, PR China. His research interests include Web Science, Peer-to-Peer Technology and Semantic Technology.



Yuanjie Liu received the B.S. degree in Computer Science from Shanghai Jiaotong University, PR China, in 2007 and he is now pursuing a M.S. degree in Computer Science from Shanghai Jiaotong University. His research interests include information retrieval, P2P network, natural language processing and web mining.



Shicong Meng received the B.S. degree from East China Normal University and M.S. degree from Shanghai Jiaotong University, China. He is currently working toward the Ph.D. degree in Computer Science at Georgia Institute of Technology, Atlanta, GA. His research concerns distributed systems, especially algorithms and architectures for large-scale overlay systems.



Yong Yu is currently a professor at the faculty of Computer Science & Engineering, Shanghai Jiaotong University, PR China. He earned a M.S. degree in Computer Science from East China Normal University in 1986. His research topics include web search, semantic search, P2P search, semantic web, ontology engineering and wireless applications.