Journal of Network and Computer Applications 33 (2010) 115–127

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

A path-traceable query routing mechanism for search in unstructured peer-to-peer networks

Ming Xu [a,b], Shuigeng Zhou [a,b,*], Jihong Guan [c], Xiaohua Hu [d]

[a] School of Computer Science, Fudan University, Shanghai 200433, China
[b] Shanghai Key Lab of Intelligent Information Processing, Shanghai 200433, China
[c] Department of Computer Science and Technology, Tongji University, Shanghai 201804, China
[d] College of Information Science and Technology, Drexel University, Philadelphia, PA 19104, USA

ARTICLE INFO

Article history:
Received 2 July 2009
Received in revised form
14 November 2009
Accepted 27 November 2009

Keywords:
Query routing
Peer-to-peer (P2P)
Network topologies
Dynamic network

ABSTRACT

Unstructured Peer-to-Peer (P2P) networks have become a very popular architecture for content distribution in large-scale and dynamic environments. Searching for content in unstructured P2P networks is a challenging task because the distribution of objects has no association with the organization of peers. Proposed methods in recent years either depend too much on objects replication rate or suffer from a sharp decline in performance when objects stored in peers change rapidly, although their performance is better than flooding or random walk algorithms to some extent. In this paper, we propose a novel query routing mechanism for improving query performance in unstructured P2P networks. We design a data structure called *traceable gain matrix* (TGM) that records every query's gain at each peer along the query hit path, and allows for optimizing query routing decision effectively. Experimental results show that our query routing mechanism achieves relatively high query hit rate with low bandwidth consumption in different types of network topologies under static and dynamic network conditions.

1. Introduction

In the past decade, peer-to-peer (P2P) networks have rapidly evolved and have become an important part of the Internet. P2P systems are application layer overlay networks that enable users to share resources in a distributed manner. There are mainly two kinds of overlays for P2P networks: structured and unstructured. Structured P2P networks have been developed to improve the performance of data discovery in such a way that each object is identified by a key and peers are organized into a structured graph, which maps each key to a corresponding peer. By far, the most common-used structured P2P networks are based on *distributed hash table* (DHT), in which a variant of consistent hashing is used to assign ownership of each object to a particular peer (Stoica et al., 2001; Ratnasamy et al., 2001; Rowstron et al., 2001; Zhao et al., 2001). It is commonly believed that structured P2P networks are more expensive to maintain than unstructured P2P networks and the constraints imposed by the structure make them hard to improve scalability. Unstructured P2P networks are highly robust to peer failure or churn, and they are also relatively straightforward to implement multiple keyword search or partial match (Zhang and Hu, 2007). So many P2P applications are still deployed on unstructured P2P networks.

There are many successful applications of unstructured P2P networks, such as Gnutella (Frankel and Pepper, 2000), Freenet (Clarke et al., 2000), KaZaA (Heinla et al., 2001) and Skype (Heinla et al., 2003). In these systems, a large number of peers collaborate in a dynamic and ad hoc manner and share information in large-scale distributed environments without any centralized coordination.

Typically, a P2P network involves millions of peers, each of which may join and leave the network in an unpredictable manner, and the objects kept in peers are changing from time to time. There is no correlation between a peer and the objects managed by it in unstructured P2P networks. If a peer wants to find a desired object in the network, the query message has to be flooded through the network to search as many peers as possible. Each query message has a unique message ID. A message that has the same message ID as the one received previously by the same peer is considered a redundant message and will be discarded. If the object is kept by only a few peers, the search either is prohibitively cost or fails eventually. So a challenging issue is how to search the objects efficiently without incurring heavy traffic in unstructured P2P networks.

Gnutella (Frankel and Pepper, 2000) uses flooding based routing algorithm with time-to-live (TTL) to search for objects.

* Corresponding author at: School of Computer Science, Fudan University, Shanghai 200433, China. Tel./fax: +86 21 55664298.
E-mail addresses: mingxu@fudan.edu.cn (M. Xu), sgzhou@fudan.edu.cn (S. Zhou), jhguan@tongji.edu.cn (J. Guan), thu@cis.drexel.edu (X. Hu).

1084-8045/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2009.11.003

This method is suitable for dynamic network environments but may lead to too many messages, particularly in high connectivity overlay networks. More than 70% of the generated messages are redundant in a flooding with a TTL of 7 in a moderately connected Gnutella network (Jiang et al., 2008). Also, it is not easy to choose an appropriate TTL value when the overlay topology is unknown.

Random walk algorithms have been studied in Gkantsidis et al. (2004, 2005), Lv et al. (2002), and Bisnik and Abouzeid(2007), which can achieve better results than flooding when the overlay topology is clustered and similar search requests are re-issued repeatedly, as well as the entire topology does not change dramatically (less than 40%) (Gkantsidis et al., 2004). In a *k*-walker random walk algorithm (Lv et al., 2002), the query is forwarded to *k* randomly selected neighbors. Those neighbors in turn forward to *k* randomly selected neighbors. Unlike flooding, the overhead of random walk is independent of the underlying topology, but the performance of random walk largely depends on the choice of *k* and TTL value. Intuitively, the average number of peers required to be probed for discovering an object is inversely proportional to the popularity of the object (Bisnik and Abouzeid , 2007). Choosing low values of *k* and TTL value for searching objects with low popularity would result in low query hit rate and high delays while choosing high values of *k* and TTL value for searching objects with high popularity would result in excessive bandwidth consumption.

Generally speaking, random walk based algorithms reduce the network load generated by each query, but massively increase the search latency compared with flooding based algorithms. There is usually a tradeoff between random walk and flooding, and the comparison of their performance is addressed in Lv et al. (2002). Chawathe et al. (2003) presented a Gnutella-like P2P file-sharing system called *Gia*, which replaces Gnutella's flooding with random walk. *Gia* incorporates heterogeneity into system design and tries to direct queries towards the high-capacity nodes, which are typically much better able to answer the queries.

In this paper, we propose a *p* ath-traceable *q* uery *r* outing (short for *PQR*) mechanism for search in unstructured P2P networks. We design a novel data structure called *traceable gain matrix* (short for *TGM*) that records every query's gain at each peer along the query hit path, and allows for optimizing query routing decision. After a query hit happens at a target peer, the query message will be forwarded to all neighbors of the target peer for calculating the number of query hit peers if its hop number is less than the TTL value. The purpose is to increase the gain value if duplicated objects are distributed in a clustering manner. The gain value will be transferred iteratively between adjacent peers along the path from the target peer to the source peer with exponential decay of distance. When a peer receives a new query message, it calculates the gain values of its neighbors to which the query has been forwarded and terminated with query hit. The neighbor with the highest gain value of current query will be selected as candidate with the highest probability to forward the query message. PQR exhibits the following merits:

- Can achieve relatively high query hit rate with low bandwidth consumption.
- Be effective in different network topologies.
- Can be adapted to dynamic environments where peers join and leave the networks randomly and continuously.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 presents technical details of our PQR method. Section 4 describes performance evaluation. Finally, Section 5 concludes the paper.

2. Related work

A number of solutions have been proposed to address the issue of query routing in unstructured P2P networks.

Cohen et al. (2007) proposed a new class of decentralized P2P architectures called *associative overlays*. The central ingredient in their design is guided search on guide-rules overlays. The set of peers belonging to some guide-rules should contain data items that are semantically similar. The guide-rules compulsorily require each peer, for each guide rule it belongs to, maintains a small list of other peers belonging to the same guide rule. The propagation of guided-search queries is restricted to guide-rules specified by the originating peer, which can dramatically increase the effectiveness of search for rare items. But the query performance relies heavily on the automatically extracted guide rules. If there are no correlation between items, guided-search has no advantage over blind search. Moreover, this strategy becomes less effective when items stored in each peer change from time to time. The maintenance cost is also extremely large when peers join and leave the network randomly and continuously.

Sarshar et al. (2004) introduced the percolation search algorithm for locating and retrieving content in random networks with power-law and heavy-tailed degree distributions. Each node in the network duplicates its content list through a random walk starting from itself. To start a query, a query request is implanted through a random walk starting from the requester. When the search begins, each node with a query implantation starts a probabilistic broadcast search, where it sends a query to each of its neighbors with probability. The percolation search algorithm requires computing the number of high degree nodes in a given network and choose appropriate random walk length in order to ensure that the query can be passed to a highly connected node with high probability. So this strategy can only be effective in specific networks with given content distribution.

Michlmayr (2006) presented ant algorithms for distributed query routing based on the *ant colony optimization* meta-heuristic. This method depends on a large number of repeated queries to build a *pheromone trail* and is feasible only in static network environments. Morselli et al. (2005) proposed *local minima search* (LMS) protocol for efficient lookup in unstructured P2P networks. LMS borrows the ideas of *namespace virtualization* and *consistent hashing* employed in most DHT based structured P2P networks. In LMS, the owner of each object places replicas of the object on several peers. Like in a DHT, LMS places replicas onto peers which have IDs "close" to the object. This method need some mechanism to compute how many replicas of its items a peer should place in order to be easily found by others which is quite unlikely to occur in a real P2P system.

Kumar et al. (2005) designed an *exponential decay bloom filter* (EDBF) for space-efficient representation of routing tables and used these probabilistic routing tables for forwarding queries. The EDBF is a bandwidth-efficient data structure to compress the query routing information in an array of bits. At a peer with degree *d*, its query routing table consists of *d* EDBF data structures of the same array size. Query objects can be added to the EDBF conveniently (the more the objects are added to the array, the larger the probability of false positive is), but removing one or more objects from the EDBF is not permitted. Routing table update must create fresh EDBFs periodically unless no object is removed from the peers. The use of routing indices in P2P networks was proposed by Crespo and Molina (2002). The authors presented three routing indices schemes: the compound, the hop-count, and the exponential routing indices to forward queries to neighbors that are more likely to have answers, rather than by selecting neighbors at random or by flooding the network. Their schemes assume that the query documents are classified under

particular topics, and all peers in the system need to index the number of documents under each topic of interest through each neighbor, which may consume a lot of storage space for storing index information of each neighbor. Connelly et al. (2006) proposed an adaptive query routing method to forward query messages based on rules generated through the use of association analysis, which has been widely studied in the data mining community. The maintenance of effective rule sets is very difficult in dynamic environments where peers joining and leaving the network continuously. Tsoumakos and Roussopoulos (2003) proposed an *adaptive probabilistic search* (APS) method for unstructured P2P networks. In the forwarding process, a peer chooses its next-hop neighbor(s) not randomly, but uses the probabilities given by its index values. Upon walker termination, if the walker is successful, index values relative to the requested object along the walker's path should be increased (optimistic approach), and if the walker fails, related index values should be decreased (pessimistic approach). The adaptivity of APS approach is not so good when objects stored in peers change rapidly. If objects with high forwarding probabilities are removed from the network, the system will endure many rounds of failure to decrease their index values in related peers unless the index table update process comes soon.

3. Technique

In this section we present the PQR method in detail.

3.1. Query message

A query message *mes* initiated at peer p_s is represented by a tuple of six elements: $\langle id, p_s, state, hops, q_j, path \rangle$. *id* stands for the unique identifier of a message in the network. p_s refers to the source peer that initiates the query. It consists of a 32-bit IP address and a 16-bit port number. We define three states (*initiate*, *forward* and *terminate*) for a query message. *hops* denotes the life time of a query message. Its value increases 1 after being transmitted from one peer to another. In order to prevent a query from incurring too much traffic in the network, we define TTL value as an upper bound to query message life time. Query vector q_j contains the objects requested by the query. *path* is a list of peers that have processed the query, including peer p_s.

3.2. Traceable gain matrix

TGM is a key component of PQR with a compound data structure that maintains query routing information. Before the construction of TGM at a peer, we need two auxiliary data structures: *QueryTable* and *PeerTable*. A peer's *QueryTable* is a query set that records all queries initiated or forwarded from this peer with query hit. A peer's *PeerTable* is a peer set that records its neighbor(s) to which one or more queries once forwarded with query hit. So a peer's *PeerTable* is a subset of its neighbors. After a query hit happens, the query that matches with one or more objects will be stored at each peer's *QueryTable* along the path and each peer's adjacent peer towards the target peer will be added to its *PeerTable* one by one (NOT including the target peer).

Given an object set O with k elements, a query set Q with n elements, and a peer set P with m elements. For an arbitrary object $o_l \in O$ $(1 \le l \le k)$, an arbitrary query $q_j \in Q$ $(1 \le j \le n)$, and an arbitrary peer $p_i \in P$ $(1 \le i \le m)$, the query object set of q_j is $o_{q_j} \subseteq O$ $(1 \le j \le n)$. TGM is an $m \times n$ matrix like this:

$$\begin{array}{c} \\ p_1 \\ \vdots \\ p_i \\ \vdots \\ p_m \end{array} \begin{array}{c} \begin{array}{ccccccc} q_1 & q_2 & \cdots & q_j & \cdots & q_n \end{array} \\ \begin{pmatrix} gain_{11} & gain_{12} & \cdots & gain_{1j} & \cdots & gain_{1n} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ gain_{i1} & gain_{i2} & \cdots & gain_{ij} & \cdots & gain_{in} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ gain_{m1} & gain_{m2} & \cdots & gain_{mj} & \cdots & gain_{mn} \end{pmatrix} \end{array},$$

where each item has a compound data structure. The number of rows in a peer's TGM is equal to the length of its *PeerTable* and the number of columns in a peer's TGM is equal to the length of its *QueryTable*.

At the beginning of query processing, both *QueryTable* and *PeerTable* are empty at each peer, as well as the corresponding TGM. Fig. 1 illustrates the transfer of gain values with exponential decay after query hit. Each circle denotes a peer in the P2P overlay network. Each arrow denotes the transfer of gain value between adjacent peers along a query hit path. More vertical lines across an arrow mean more gains. There are three query hit paths labeled as #1, #2 and #3 in Fig. 1. Suppose peer p_s initiates a query q_j with the query object set $o_{q_j} = \{o_1, o_2\}$ and sends a query message to its neighbor peer p_i. At last, the query hit happens at the target peer p_t (o_1 or o_2 is found at peer p_t for 'OR' query or o_1 and o_2 are found at peer p_t for 'AND' query). If the current *hops* is smaller than TTL, the query message will be forwarded to p_t's neighbors in order to search more peers that meet the query (gray circles denote query hit peers). Suppose q_j is the jth element in p_s's *QueryTable* and p_i is the ith element in p_s's *PeerTable*, then $\langle \{p_i, p_k, \ldots, p_u, p_t\}, value \rangle$ will be added to $gain_{ij}$. The value of $gain_{ij}$ is calculated as

$$|gain_{ij}| = \sum_{p_t \in D} \frac{(1-\alpha) \cdot \omega_{p_t} + C \cdot \alpha}{dist(p_s, p_t)^{\lambda}} \quad (1)$$

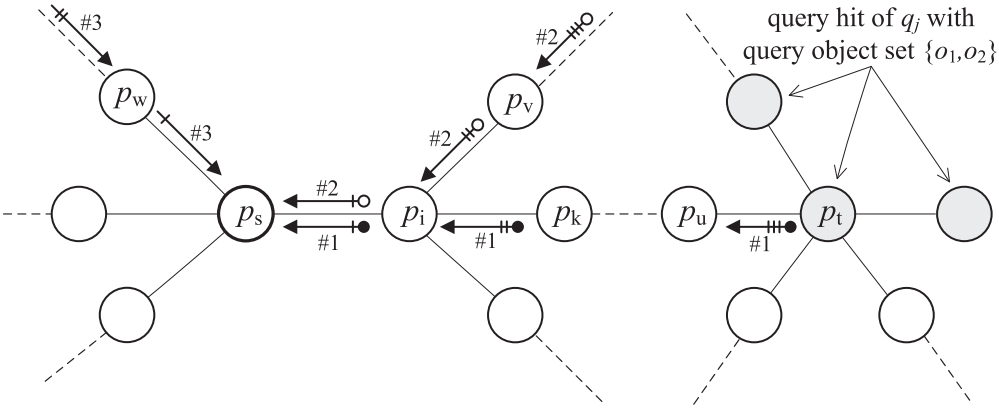

Fig. 1. The transfer of gain values with exponential decay after query hit.

with $0<\alpha<1$ and $0<\lambda<\infty$, where D is a collection of peers at which query hit happens (via peer p_i). p_t is one of the target peer in collection D and ω_{p_t} is the total query hit number at the target peer p_t's neighbors. Duplicate elements are not allowed in D to make sure that the gain value from the target peer p_t and p_t's neighbors will not be calculated repeatedly no matter how many query hit happens at the target peer p_t. α is a parameter to measure the importance between query hit at the target peer p_t and p_t's neighbors. C is a constant value to denote the weight of the target peer p_t. $dist(p_s,p_t)$ denotes the path length from the current peer p_s to the target peer p_t and exponential decay parameter λ is used to adjust the influence of $dist(p_s,p_t)$ on the gain value. If $\lambda\to\infty$, then the gain value from the target peer p_t and p_t's neighbors can only be transferred to one peer along the whole path; and if $\lambda\to 0$, then the gain value from the target peer p_t and p_t's neighbors will be equivalently transferred to all peers along the query hit path one by one.

Suppose the TGM at peer p_s has r rows, then the probability of forwarding query q_j to peer p_i is calculated as follows:

$$P(q_j,p_i)=\frac{|gain_{ij}|}{\sum_{i=1}^{r}|gain_{ij}|}. \tag{2}$$

After the transfer of gain value along the path #1 ($p_t\to p_u,\ldots,p_k\to p_i,p_i\to p_s$) with exponential decay of distance, forwarding another query q_i with the same query object set as q_j from peer p_i to peer p_k has more probability of query hit than forwarding the same query from peer p_s to peer p_i under the same conditions ($q_i=q_j$ if $o_{q_i}=o_{q_j}$).

Note that $|gain_{ij}|$ stands for the total gain value transferred from peer p_i to peer p_s. Maybe a query q_j sent to another neighbor peer p_w also hits successfully at a target peer (NOT peer p_t), but it makes no contribution to $gain_{ij}$ since peer p_w is not the ith element in p_s's *PeerTable*. The arrows in path #3 denote the transfer of gain value from another target peer (NOT peer p_t) and its neighbors to the source peer p_s via peer p_w for the query q_j initiated at peer p_s. So the gain value transferred along path #3 will not be added to $|gain_{ij}|$. The arrows in path #2 denote the transfer of gain value from a target peer (NOT peer p_t) and its neighbors to the source peer p_s via peer p_i for the query q_j initiated at peer p_s. So the gain value transferred from peer p_i to p_s along path #2 can be added to $|gain_{ij}|$ as well as path #1. For example, suppose the gain value transferred from peer p_i to p_s in path #1 is 5, the gain value transferred from peer p_i to p_s in path #2 is 2 and the gain value transferred from peer p_w to p_s in path #3 is 3, then the probability of forwarding the query q_j from peer p_s to peer p_i is $(5+2)/(5+2+3)=70\%$. Peer p_s cannot forward the query q_j directly to the target peer p_t because peer p_t is not recorded in peer p_s's *PeerTable*. If a new query comes without matching with any element in current peer's*QueryTable*, it will be forwarded to a random neighbor peer except the peer that has ever sent or forwarded this query.

The space complexity of each peer is $O(m\times n)$. Such space cost is not a burden in current P2P networks because TGMs are not needed to be transmitted across the peers. Each peer needs only to keep a small size of TGM and least-recently-used strategy is used to update TGM when it is full. The space cost of *QueryTable* and *PeerTable* is negligibly small compared with that of TGM at each peer because of their linear length.

3.3. Query routing and TGM update

The Path-traceable query routing mechanism includes the process of query routing and the process of update. Algorithm 1 describes them in detail.

Algorithm 1. Path-traceable query routing algorithm.

Input: network size, network topology model and related parameters, TTL value, query peer number;
Output: message *id*, message *hops* and query result (1/0);

Peer p_t gets a query message $mes\langle id,p_s,state,hops,q_j,path\rangle$;
if q_j hits at p_t **then**
  Forwards *mes* to p_t's neighbors and calculates the total query hit number;
  Peer $p_s\leftarrow mes.p_s$;//Iterates all peers in *mes.path* for update
  for all peers in *mes.path* **do**
    if $p_s.QueryTable\neq\varphi$ AND $q_j\in p_s.QueryTable$ **then**
      return q_j's position j;
    else
      $p_s.QueryTable.Add(q_j)$;
    end if
    Peer $p_i\leftarrow mes.path.next()$;
    if $p_i\in p_s.PeerTable$ **then**
      return p_i's position i;
    else
      $p_s.PeerTable.Add(p_i)$;
    end if
    Calculates the gain value $value_k$ from p_i to p_s according to formula (1);
    $gain_{ij}.Add(mes.path,value_k)$;
    $p_s\leftarrow p_i$;
  end for
  return *mes.id*, *mes.hops* and 1;//Query hit
else if *mes.hops*= TTL **then**
  Peer $p_s\leftarrow mes.p_s$;
  for all peers in *mes.path* **do**
    Peer $p_i\leftarrow mes.path.next()$;
    if $p_i\in p_s.PeerTable$ AND $q_j\in p_s.QueryTable$ **then**
      return p_i's position i and q_j's position j;
      if $p_t\in gain_{ij}.path_k$ **then**
        $gain_{ij}.Remove\ (path_k,value_k)$;
      end if
      if $gain_{ij}=\varphi(i=1,2,\ldots m)$ **then**
        $p_s.QueryTable.Remove\ (q_j)$;
        Removes the jth column from $p_s.TGM$;
      end if
    end if
    $p_s\leftarrow p_i$;
  end for
  return *mes.id*, *mes.hops* and 0;//Query miss
end if
// Query not hit at p_t and *mes.hops* < TTL;
Forwards *mes* to p_t's neighbor p_x according to formula (2);
while p_t receives an *ERROR* message after forwarding *mes* to p_x **do**
  return p_x's position i at $p_t.PeerTable$;
  $p_t.PeerTable.Remove\ (p_x)$;
  Removes the ith row from $p_t.TGM$;
  for all peers in *mes.path* **do**
    for (i = 1 to m; j = 1 to n) **do**
      if $p_x\in gain_{ij}.path_k$ **then**
        $gain_{ij}.Remove\ (path_k,value_k)$;
      end if
    end for
  end for
  Forwards *mes* to p_t's neighbor p_x according to formula (2);
end while

After a query hit happens at a target peer p_t, the query message needs to be forwarded to all neighbors of the peer p_t for calculating the number of query hit peers if its hops is less than TTL value. Then the gain will be transferred one by one at each pair of adjacent peers along the path from target peer towards source peer with exponential decay of distance as described from line 1 to line 21.

As introduced formerly, $gain_{ij}$ has a compound data structure. Each item in $gain_{ij}$ is denoted by $\langle path_k, value_k \rangle$, where $path_k$ is an ordered peer list from current peer's neighbor to the target peer along the query hit path. For example, suppose peer p_s in Fig. 1 initiates a query message *mes* with query vector q_j. Peer p_s's TGM is

$$\begin{array}{c} \\ p_i \\ p_w \\ \vdots \\ p_m \end{array} \begin{array}{c} \begin{array}{cccc} q_1 & q_2 & \cdots & q_j \end{array} \\ \begin{pmatrix} gain_{11} & gain_{12} & \cdots & gain_{1j} \\ gain_{21} & gain_{22} & \cdots & gain_{2j} \\ \vdots & \vdots & \cdots & \vdots \\ gain_{m1} & gain_{m2} & \cdots & \varphi \end{pmatrix} \end{array},$$

where $gain_{1j} = (\langle \{p_i, p_k, \ldots p_u, p_t\}, 5 \rangle, \langle \{p_i, p_v, \ldots\}, 2 \rangle)$, $gain_{2j} = (\langle \{p_w, \ldots\}, 3 \rangle)$ and $gain_{ij} = \varphi$ $(i = 3, \ldots, m)$. If *mes* arrives at the last peer p_t without query hit ($mes.hops = \text{TTL}$), the intermediate routing peers $(p_i, p_k, \ldots, p_u)$ will check their TGMs as follows. Firstly, each peer searches query vector q_j in its *QueryTable*. If query vector q_j is not found in *QueryTable*, then no update process is needed. Otherwise, query vector q_j's position in *QueryTable* will be returned as a key to locate the corresponding column in current peer's TGM (suppose the column number is j). Secondly, each item of the jth column must be scanned to find peer p_t. If peer p_t is included in $path_k$ of $gain_{ij}$, then $\langle path_k, value_k \rangle$ will be removed from $gain_{ij}$. As a result, $\langle \{p_i, p_k, \ldots, p_u, p_t\}, 5 \rangle$ is removed from $gain_{1j}$ and the probability of forwarding query q_j from peer p_s to peer p_i drops to $2/(2+3) = 40\%$ according to formula (2). Thirdly, if every item in the j th column of TGM equals to φ, then the whole column will be removed as well as the corresponding element in current peer's *QueryTable*. The whole process is described from line 22 to line 39.

If query q_j not hits at peer p_t and p_t is not the last peer ($mes.hops < \text{TTL}$), it will calculate the gain values of its neighbors to which the same query is once forwarded and terminated with query hit. The neighbor with the biggest gain value on current query will be selected as candidate with the highest probability to forward the query message according to formula (2). If p_t finds that the candidate peer p_x fails or leaves the P2P network, the update process will be handled as follows. Firstly, peer p_x will be removed from peer p_t's *PeerTable* as well as its neighbor table. Additionally, the corresponding row in peer p_t's TGM will also be removed. Secondly, all peers along the path from source peer to peer p_t will scan each item in their TGMs to find peer p_x. If peer p_x is included in $path_k$ of $gain_{ij}$, then $\langle path_k, value_k \rangle$ will be removed from $gain_{ij}$. Thirdly, peer p_t will forward *mes* to another neighbor iteratively. The corresponding process is described from line 40 to line 54.

When a neighbor peer fails or leaves the network, no update happens until one peer needs to forward a query message to that peer. The update cost includes sending update messages to the peers in the current query routing path as well as scanning and updating TGMs in these peers. Suppose all k walkers travel h hops and find that the last peers leave the network, then the cost of sending update messages is $O(k \cdot h)$. In the worst case, the cost is $k \cdot \text{TTL}$. Moreover, suppose the peer that fails or leaves the network is included in each peer's TGM along the query routing path and the size of TGM is m rows and n columns, then the cost of updating TGMs is $O(k \cdot h \cdot m \cdot n)$ in the worst case. In fact, the update cost for handling peer failure or churn is not big in real network environments. The reason lies in three aspects. First, the value of k is very small. Our algorithm can achieve good query performance even when $k = 1$. Second, the probability of forwarding a query message to a peer that is TTL hops away is rather small. Third, if no query message is forwarded to the peer that fails or leaves the network, then no update happens.

TGM needs update messages for its production as the index table in the APS method does. Suppose there are r query requests and k walkers. Theoretically, both PQR and APS produce $O(r \cdot k \cdot \text{TTL})$ update messages in the worst case.

APS uses two update policies (optimistic and pessimistic approaches) to reduce the number of update messages. After a walker terminated, if the walker is successful, there is nothing to be done in the optimistic approach. If the walker fails, the index table of each peer along the walker's query routing path, from the last-visited peer towards the requester peer, must be updated. In the pessimistic approach, this update procedure takes place after a walker succeeds, and there is nothing to be done when a walker fails.

In order to minimize the number of update messages, PQR takes three measures as follows:

- If a query fails after TTL hops, the peer before the last-visited peer checks its TGM. If the requested object is not included in any item of the TGM, no update message will be propagated to any peer because the TGM does not account for the failure in this situation. If the requested object is included in a certain related item of the TGM, it means that the object stored ever in the peer has been removed and at most TTL-2 update messages will be produced in this situation. During the process of update, update messages are propagated one by one in the reverse direction of query message. At each step, the intermediate peer checks the current *path* in its TGM. If the path is not contained in any item of the TGM, the update procedure will stop. In such a case, less than TTL-2 update messages will be produced.
- After a query hit happens, the query message with query result will be returned to the original peer that initiates the query request. The original peer extracts the *path* from the query message and scans the *path* in each item in the corresponding row of its TGM. If the *path* is already included in the corresponding items of the TGM, no update message will be propagated to any peer because the gain value from the same target peer cannot be calculated repeatedly according to formula (1).
- No big value is used for the parameter k because even with a small k our method can achieve a high query hit rate.

Our algorithm is totally distributed and each peer does not need any global information of the network. In fact, each peer in the network only knows its neighbors and has no explicit knowledge of other peers.

4. Performance evaluation

We use PeerSim (Jesi, 2003) to implement and evaluate our method. PeerSim is an open source, Java based, P2P simulation framework for large-scale and dynamic environments.

4.1. Network topologies

We deploy query routing strategies in three representative network topologies to compare their performance and to evaluate the impact of network topologies on query performance.

4.1.1. Random graph networks

Random graphs were first defined by Erdös and Rényi (1959). A random graph can be denoted as $G(n,p)$. It means every pair of a set of n vertices is chosen to join an edge with probability p. Many properties of the random graphs were studied by Erdös and Rényi in a series of papers in the 1960s (Erdös and Rényi, 1959, 1960, 1961).

A random graph can be obtained by starting with a set of n vertices and adding edges between them randomly. Different random graph models produce different probability distributions on graphs. The probability of a vertex having degree k is

$$p(k)=\binom{n}{k}p^k(1-p)^{n-k}\simeq\frac{\mu^k e^{-\mu}}{k!}, \tag{3}$$

which has a Poisson degree distribution since the presence or absence of edges is independent.

One of the most important properties of the random graph is *phase transition* (Erdös and Rényi, 1959), from a low-density, low-p state in which there are very few edges and all components are small, having an exponential size distribution and finite mean size, to a high-density, high-p state in which an extensive fraction of vertices are joined together in a single giant component, and the remain vertices occupy smaller components with an exponential size distribution and finite mean size.

4.1.2. Small-world networks

Small-world networks have received a lot of attention from many research fields in the last decade. There are different realizations of small-world networks, but the original model proposed by Watts and Strogatz (1998) is by far the most widely studied. Generally speaking, the small-world network has two structural properties: (1) characteristic path length and (2) clustering coefficient (Watts and Strogatz, 1998).

To mathematically define the two properties, let $G=(V,E)$ be an undirected, simple (no self-loops, no multiple edges) graph with a set of vertices V and a set of edges E between them. The distance d_{ij} between two vertices i and j in V is the number of edges along the shortest path connecting them. The characteristic path length $L(G)$ is defined as the average distance between any pair of two vertices in V, that is,

$$L(G)=\frac{1}{|V|(|V|-1)}\sum_{ij\in V}d_{ij},\quad i\neq j. \tag{4}$$

The clustering coefficient $C(G)$ is defined as follows. Suppose that a vertex $v\in V$ has k_v neighbors; then at most $M_v=k_v(k_v-1)/2$ edges can exist between them (this occurs when every neighbor of v is connected at every other neighbor of v). Let N_v denote the actually existed edges between these neighbors; then the clustering coefficient C_v of vertex v is defined as N_v/M_v. Define the clustering coefficient $C(G)$ as the average of C_v over all v:

$$C(G)=\frac{1}{|V|}\sum_{v\in V}C_v. \tag{5}$$

A simple way to construct a small-world network starts by building a ring R_n with n vertices, each connected to its k nearest neighbors by undirected edges. The small-world network is then created by choosing a vertex and the edge that connects it to its nearest neighbor in a clockwise sense and with probability β reconnect this edge to a vertex chosen uniformly at random over the entire ring. If rewiring an edge would lead to a duplicate edge, it is left unchanged. The rewiring process makes the graph transform from a regular ring ($\beta=0$) to a random graph ($\beta=1$), with intermediate value of β, the graph is a small-world network (as β increases, the graph becomes increasingly disordered).

Kleinberg (2000) defined an infinite family of network models that naturally generalized the small-world model in Watts and Strogatz (1998) and then proved that there is exactly one model within this family for which a decentralized algorithm exists to find short paths with high probability.

4.1.3. Scale-free networks

Barabási and Albert found that several important networks such as the World Wide Web, protein regulatory networks, and metabolic networks also have the small-world properties but their degree distribution function is different (Barabási and Albert, 1999; Albert and Barabási, 2002). In these networks, some nodes which they called *hubs*, have many more connections than other nodes and the networks as a whole have a power-law distribution of the number of links connecting to a node. These networks have been called scale-free because the degree probability distribution function follows a power law. The mechanism of preferential attachment has been proposed to explain power law degree distributions in some networks. The nodes with larger degree are more likely to be candidates for attachment of new nodes, which lead to a class of graphs with power-law degree distribution as follows:

$$P(k)=k^{-\gamma}, \tag{6}$$

where k is the number of connections to other nodes and γ is a constant value.

The Barabási and Albert model starts with a small number m_0 nodes. At each time step add new node with $m\leq m_0$ links to existing network. The probability that a new node will be connected to node i depends on its degree k_i:

$$\pi(k_i)=\frac{k_i}{\sum_j k_j}, \tag{7}$$

where Σ denotes all the nodes in network.

Palmer and Steffan (2000) presented two algorithms for generating network topologies that obey power-law. The algorithms in Palmer and Steffan (2000) can provide more control over the network structures and generate more complicated topologies such as weighted graphs. For simplicity, we generate the scale-free network according to the Barabási and Albert model, which satisfies the requirement of simulating network environments for our experiments.

4.2. Experimental setting

Peers initiate queries for various objects. These objects are distributed across the network according to a replication distribution model, which indicates what objects are stored at each peer. All of our experiments are based on the Zipf-like distribution model, which is frequently used to simulate the replication objects on the web and in P2P systems such as Napster and Gnutella (Almeida et al., 1996). We assume that there are m objects and let q_i be the relative popularity of the ith object, then

$$q_i\propto\frac{1}{i^{\alpha}}, \tag{8}$$

where $\sum_{i=1}^{m}q_i=1$.

Constant rate of replication distribution model is not considered here because it is unlikely to occur in real P2P systems.

The query distribution determines the frequency of each object appeared in queries. Generally speaking, popular objects get more requests than unpopular ones and Zipf-like distribution model is also appropriate under this circumstance. We only simulate queries for content actually hosted in the network and only 'AND' query type is considered in this paper for simplicity. Experimental parameters and their default values are listed in Table 1.

We define "one-peer-one-query" and "one-peer-multi-query" modes in this paper. The former means each peer can initiate only one query and the latter means each peer can initiate multiple queries.

Two performance metrics including query hit rate and average number of messages per query are used to evaluate the query performance of three query routing strategies. A good query routing strategy should try to achieve high query hit rate with low average number of messages per query. In addition, memory space consumption and replacement frequency are also considered for APS and PQR, which are negligibly small for random walk.

4.3. Experimental results

Here, we put the performance evaluation results into three parts: the first part is obtained from static network environments, where during each experiment the size of network is fixed and no peer failure and churn; the second part is from dynamic network environments with peer failure/departure; and the third part covers the results of memory and message consumption for TGM generation and maintenance.

Table 1
Experimental parameters and their default values.

Experimental parameters	Default values
Network size	10 000
Number of query request peers	1000, 10 000
Replication distribution	Zipf ($\alpha = 0.82$)
Query distribution	Zipf ($\alpha = 0.9$)
Number of objects	50 000
Number of walkers	1,3
Average query length	2.117
α in TGM	0.5
λ in TGM	2
C in TGM	10
TGM size	20×50
Average node degree in random graph network	20
Average node degree in scale-free network	20
β in small-world network	0.5

4.3.1. Results in static network environments

Here, we try to evaluate the performance of three query routing strategies PQR, APS and random walk in static network environments where no node failure/departure happens.

In the first set of experiments, we adopt "one-peer-one-query" mode and deploy only one walker in each network. Fig. 2 shows the query hit rate for different TTL values in "one-peer-one-query" mode. There are 1000 query request peers and 1000 query messages that follow the Zipf-like query distribution. The curves of APS intertwine with those of random walk in the three networks. The query hit rate of PQR noticeably exceeds that of APS and random walk by about 10% when TTL value is greater than 4. The corresponding curves of average number of messages per query in different environments are illustrated in Fig. 3. The average number of messages per query in APS is almost the same as that of random walk in random graph network, but is slightly lower in scale-free network and small-world network. The performance of PQR is only a little better than APS and random walk. For example, when TTL value is 10, the average number of messages per query in random walk is 6.906 and the average number of messages per query in PQR is 6.355, as shown in Fig. 3(c).

In the second set of experiments, we adopt the "one-peer-multi-query" mode with Zipf-like query distribution at each query peer and deploy three walkers in each network. There are 10 000 query request peers with each peer initiating 40 queries. Other parameters take default values listed in Table 1. Fig. 4

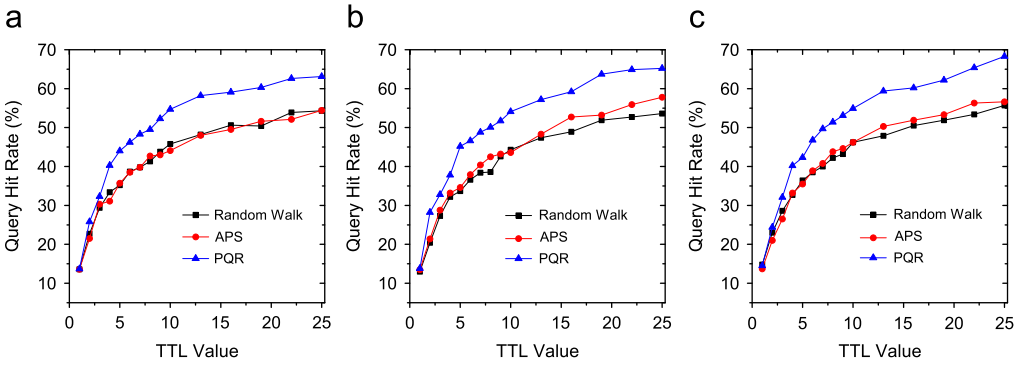

Fig. 2. Query hit rate vs. TTL value ("one-peer-one-query" mode). (a) Random graph network. (b) Scale-free network. (c) Small-world network.

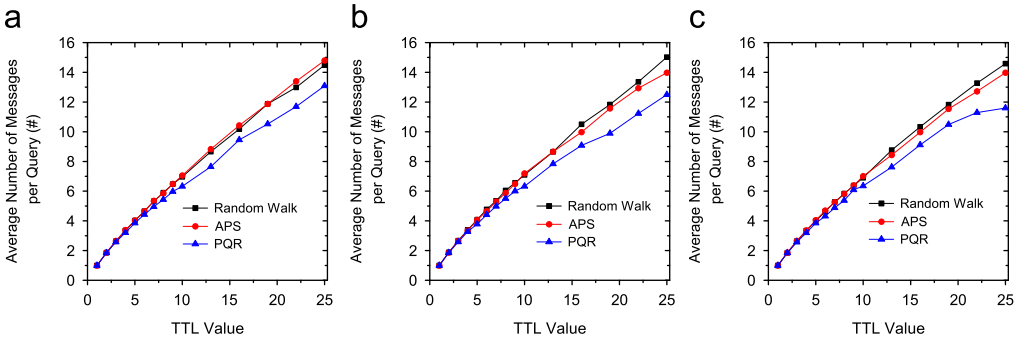

Fig. 3. Average number of messages per query vs. TTL value ("one-peer-one-query" mode). (a) Random graph network. (b) Scale-free network. (c) Small-world network.

displays the query hit rate for different TTL values in "one-peer-multi-query" mode. We can see that in all networks, PQR performs the best, and APS the second. Particularly, as shown in Fig. 4(c), in small-world network the query hit rate of PQR exceeds that of APS and random walk by 17.2% and 32.5% in the best cases, respectively. Fig. 5 shows the average number of messages per query in different networks. The performance of PQR is better than that of APS and random walk. Especially, in the small-world network, when TTL value comes up to 13, the average number of messages per query of PQR is only 58.2% of random walk and 75.4% of APS. We can conclude that the performance of PQR benefits much from small-world network.

By comparing Figs. 2 and 4, Figs. 3 and 5, we can see that the more queries are issued, the better query hit rate and the lower network consuming are obtained, for both PQR and APS as well as random walk. However, carefully comparing the results of Figs. 3 and 5, we notice that the difference in the average number of messages per query is not so big as expected. The reason lies in the fact: in Fig. 3, only one walker is employed, while in Fig. 5, three walkers are employed. And more walkers will cause more messages. This factor counteracts in some degree the benefit brought by larger number of queries. It is worthy of noting that the value of TTL also affects the difference in network consuming between Figs. 3 and 5. As we can see, when TTL = 10, the average number of messages per query of PQR in random graph network is 6.328 in "one-peer-one-query" mode and 4.937 in "one-peer-multi-query" mode, while for TTL = 25, the average number of messages per query is 13.086 in "one-peer-one-query" mode and 8.265 in "one-peer-multi-query" mode. The reason is as follows: when TTL value is not big (TTL ≤ 10), there are not many overlapping edges in different query hit paths and the difference in the average number of messages per query is not so big between the two modes. With the increase of TTL value, many overlapping edges in different query hit paths emerge. Thus, more and more queries have a higher probability of choosing shorter paths to the target peers, and subsequently the difference in the average number of messages per query becomes larger between the two modes.

To further check how the number of queries issued by each peer impacts query processing performance of PQR, we conduct another set of experiments, and the results are illustrated in Fig. 6, where three networks of different topologies are used and each network has 10 000 query request peers with one walker for each query. We can see that as the number of queries issued by each peer increases, the query hit rate first goes up steadily and then enters a relatively stable level. On the contrary, network consuming first goes down sharply and then turn to a very slowly decreasing trend. The reason is that as more queries are processed, more experiences are amassed and recorded in the TGMs, which will help the handling of following queries, and consequently improve query hit rate and lower network consuming.

We also see that PQR obtains the best performance in small-world network due to its characteristics of small average path length and high clustering coefficient, while the performance in scale-free network is worse than that in random graph network because data objects are not distributed over peers according to their degrees. Therefore, high-degree peers in scale-free network get more possibility of being visited, but they cannot provide more objects than low-degree peers can.

Three default values are assigned to parameters α, λ and C of TGM in PQR. In fact, the value of λ affects query hit rate in a obvious way. Fig. 7 shows how λ impacts query hit rate of PQR when the values of α and C are chosen according to Table 1. The width of the central peak on PQR curve is narrow. As $\lambda \to 0$, the query hit rate of PQR approaches to that of APS. When the value of λ increases beyond the central peak area, the query hit rate of PQR falls rapidly towards that of random walk.

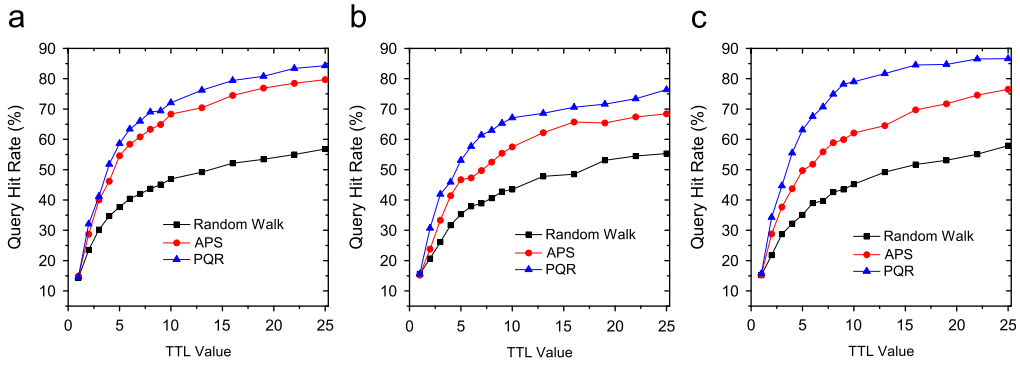

Fig. 4. Query hit rate vs. TTL value ("one-peer-multi-query" mode). (a) Random graph network. (b) Scale-free network. (c) Small-world network.

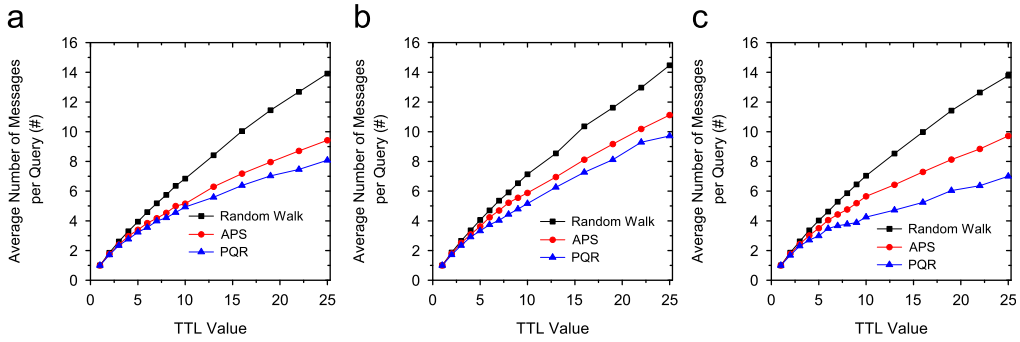

Fig. 5. Average number of messages per query vs. TTL value ("one-peer-multi-query" mode). (a) Random graph network. (b) Scale-free network. (c) Small-world network.

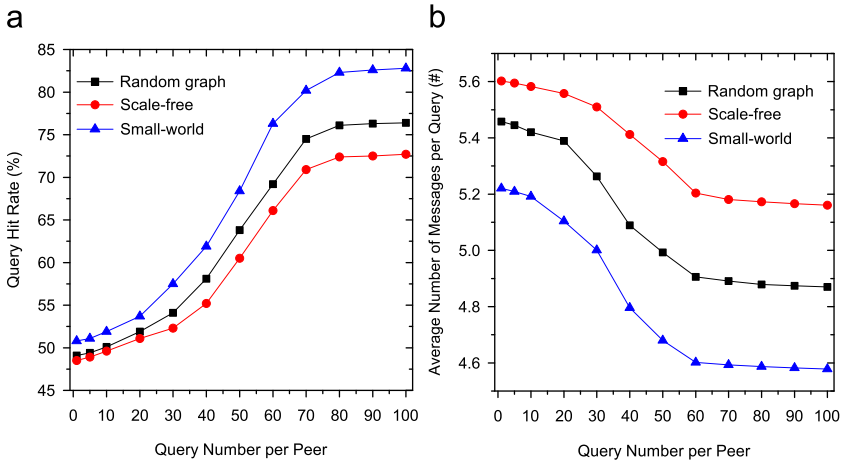

Fig. 6. The effect of query number per peer on the query performance of PQR in three networks (TTL = 10 in "one-peer-multi-query" mode with one walker). (a) Query hit rate. (b) Average number of messages per query.

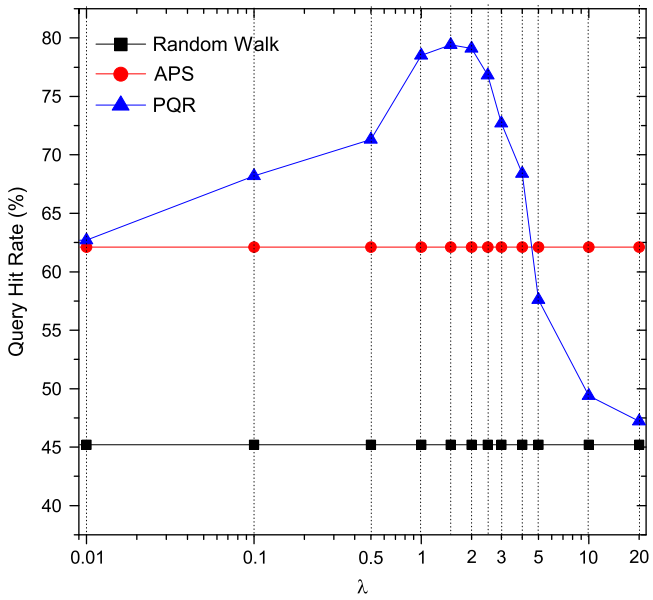

Fig. 7. Query hit rate vs. λ (TTL = 10 in "one-peer-multi-query" mode and small-world network).

4.3.2. Dynamic network environments

In order to evaluate the query routing performance in dynamic environments, we conduct a series of experiments under churn and peer failure conditions. The total experimental runtime is divided into 1000 cycles. During the churn period, the network size $N(\tau)$ will be the function of time τ with upper bound N_{max} and lower bound N_{min} in our experiments. The network size is defined as follows:

$$N(\tau) = \frac{N_{max}+N_{min}}{2} + \left\lfloor \frac{N_{max}-N_{min}}{2} * \sin\frac{\tau * \pi}{period} \right\rfloor, \quad (9)$$

where parameter *period* is used to define the length of churn period and τ is derived from *cycle id* $(1, 2, \ldots, 1000)$ in simulation. We set $N_{max} = 12\,000$, $N_{min} = 8000$ and $period = 127$ in our experiments. If a peer leaves the network, the objects stored in this peer will be discarded. If a peer joins the network, it will also get data objects with Zipf-like replication distribution. Furthermore, the objects stored in 20% of peers will be reallocated in simulation of churn.

We assign each peer a failure probability p for the purpose of comparing query routing performance of the three strategies under peer failure. We set $p = 0.2$ in simulation.

Fig. 8 shows the comparison of query hit rates of the three query routing strategies under churn in a random graph network. We find that the query hit rate of APS is a little bit lower than that of random walk in "one-peer-one-query" mode and the query hit rate of PQR exceeds both random walk and APS by about 8.6% on average. In the "one-peer-multi-query" mode, the query hit rate of PQR exceeds APS by about 13.1% on average, as shown in Fig. 8(b), while the query hit rate of APS is 5.1% better on average than that of random walk.

Fig. 9 illustrates the comparison of query hit rate of the three query routing strategies under peer failure in a random graph network. The query hit rate of APS is slightly better than that of random walk under peer failure, for both "one-peer-one-query" and "one-peer-multi-query" modes. The query hit rate of PQR exceeds that of APS by about 8.9% in "one-peer-one-query" mode and 11.8% in "one-peer-multi-query" mode.

Fig. 10 presents the comparison results of the three query routing strategies in the "one-peer-multi-query" mode, under the situations that churn or peer failure happens in a small-world network. The query hit rate of APS exceeds that of random walk by about 6.1%, and the query hit rate of PQR surpasses that of APS by about 17.3% on average under churn as shown in Fig. 10(a), and by about 12.4% on average under peer failure as shown in Fig. 10(b).

4.3.3. Cost for TGM generation and maintenance

PQR achieves high query hit rate and low number of messages for query answering at the cost of memory space. APS also consumes a certain amount of memory space for the maintenance of its index table. A TGM is a matrix of 20 rows and 50 columns, which is similar in size to the index table of APS in our experiments. We employ least recently used (LRU) strategy to replace items in PQR's TGM and APS' index table.

Figs. 11 and 12 illustrate the comparisons of space cost per peer and total number of replacements for different query number per peer between APS and PQR in "one-peer-multi-query" mode. Although PQR consumes more memory space than APS under the same condition, the cost is quite acceptable. For example, PQR consumes 393 kb memory per peer in a small-world network, where there are 10 000 query request peers and each peer initiates 40 queries. The total number of replacements in APS is smaller than that of PQR when the query number per peer is not

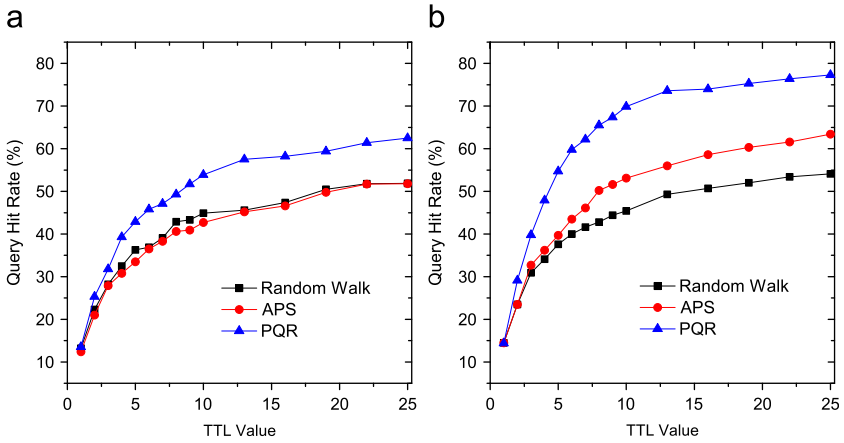

Fig. 8. Query hit rate vs. TTL value (under churn in random graph network). (a) "One-peer-one-query" mode. (b) "One-peer-multi-query" mode.

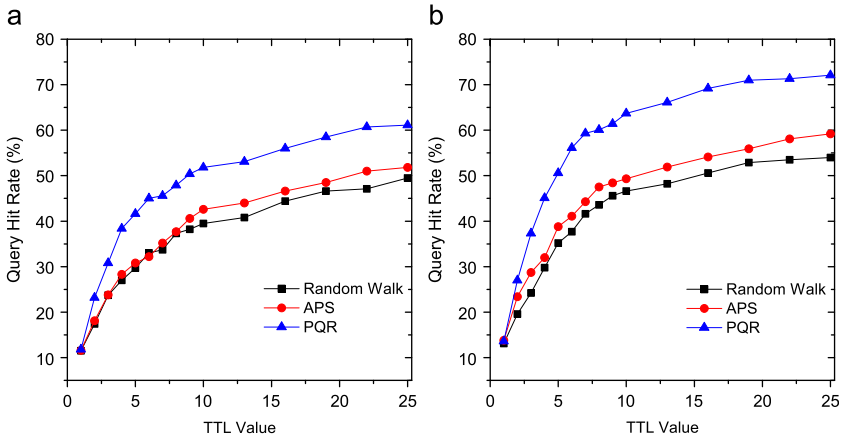

Fig. 9. Query hit rate vs. TTL value (under peer failure in random graph network). (a) "One-peer-one-query" mode. (b) "One-peer-multi-query" mode.

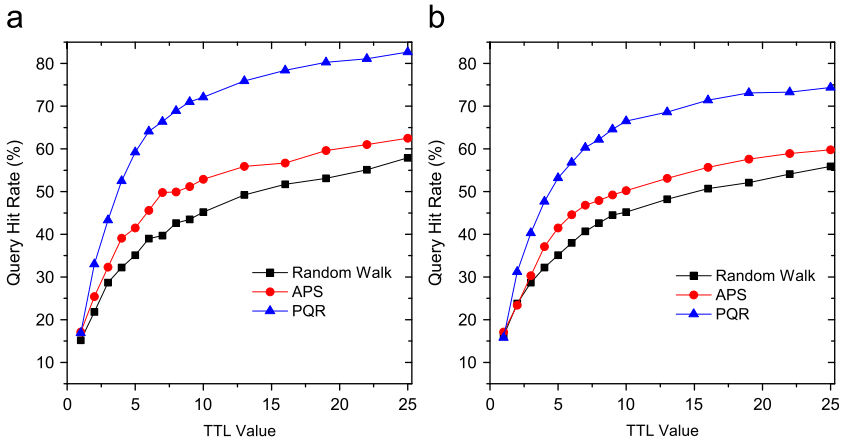

Fig. 10. Query hit rate vs. TTL value (under churn and peer failure in "one-peer-multi-query" mode and small-world network). (a) Churn. (b) Peer failure.

quite large. But with the increase of query number per peer, the curves of APS turn more steep than that of PQR.

Fig. 13 displays the comparisons of the total access times to TGM in PQR and to index table in APS per query for different numbers of query peers, each of which initiates 40 queries in random graph and small-world networks. Generally, the total access times is proportional to the number of query peers and TTL value in both cases, and the total access times to TGM of PQR is more than that to the index table of APS under the same conditions, which means that TGM of PQR has a higher utility than the index table of APS.

In the last set of experiments, we compare the cost of TGM maintenance in PQR and the cost of index table maintenance in APS under different conditions in a small-world network. Four cases are considered: "one-peer-one-query" mode, "one-peer-multi-query" mode, with peer failure and with peer churn. In the case of "one-peer-one-query" mode, 1000 query request peers are deployed and the total 1000 queries follow the Zipf distribution.

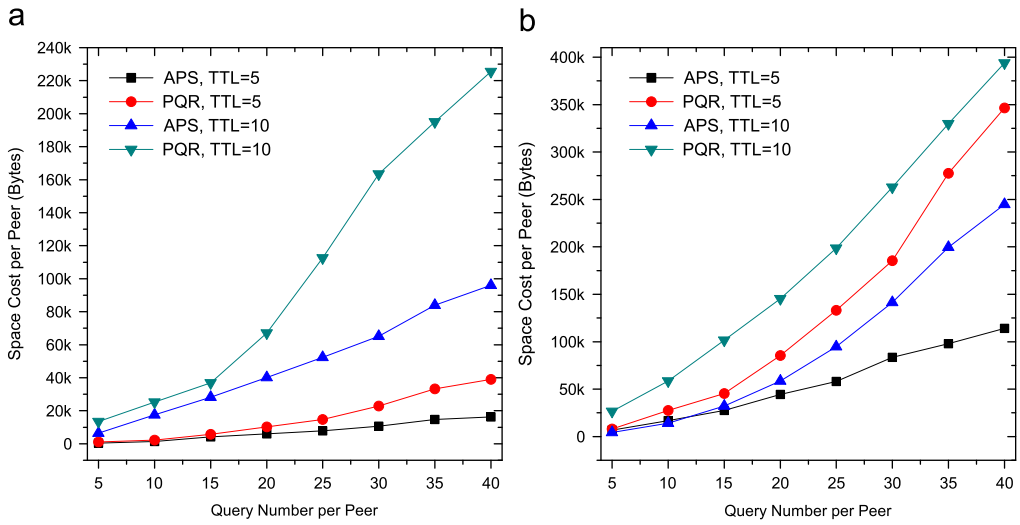

Fig. 11. Space cost vs. query number per peer ("one-peer-multi-query" mode). (a) Random graph network. (b) Small-world network.

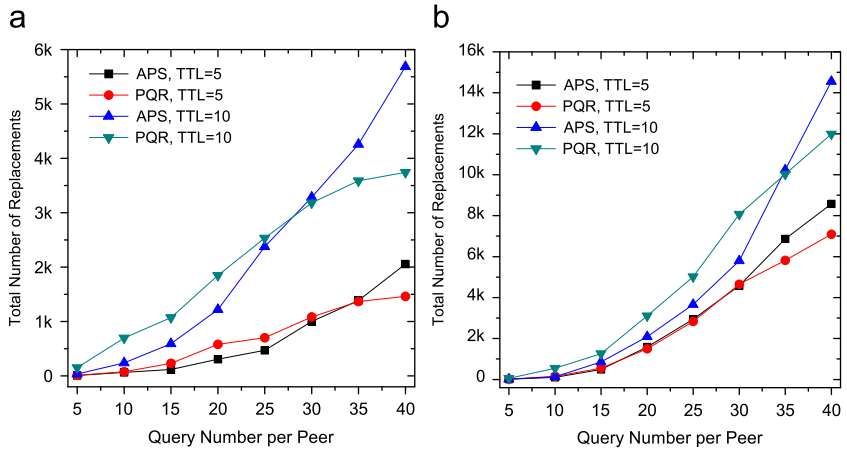

Fig. 12. Total number of replacements vs. query number per peer ("one-peer-multi-query" mode). (a) Random graph network. (b) Small-world network.

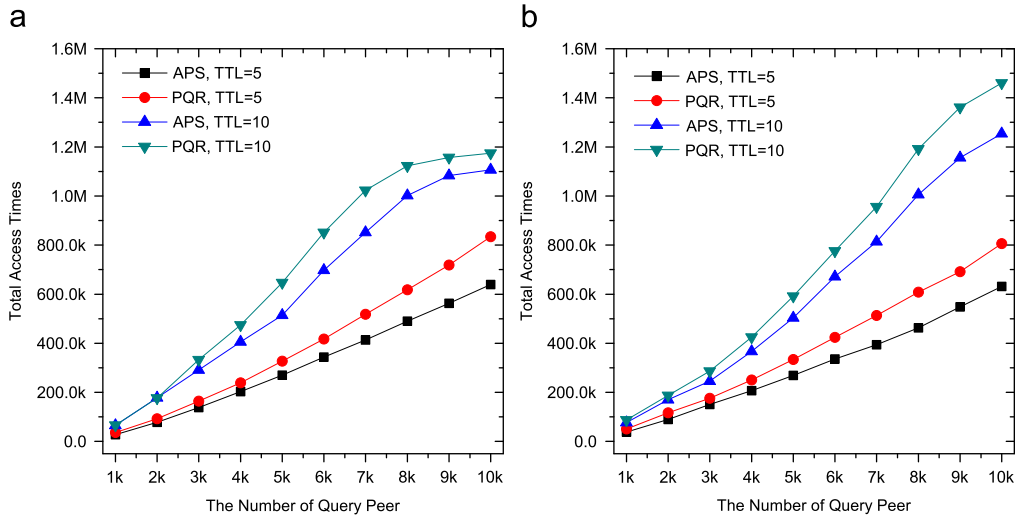

Fig. 13. Total access times vs. the number of query peer (40 queries per peer). (a) Random graph network. (b) Small-world network.

For the case of "one-peer-multi-query" mode, 10 000 query request peers are deployed, each of which initiates 40 queries. The first two cases are conducted in a static small-world network. In the third case, 2000 peers leave the network during the experiment. And in the fourth case, there are 2000 peers join and leave the network. Corresponding experimental results are presented in Figs. 14(a)–(d). From Fig. 14, we can see that PQR needs more update messages than APS in "one-peer-one-query"

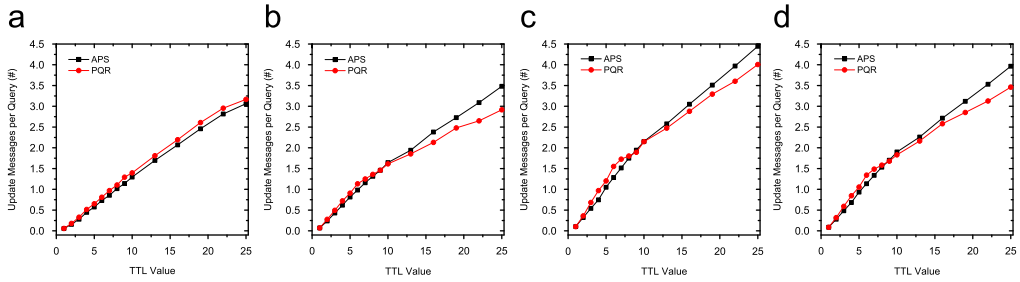

Fig. 14. Total number of update messages vs. TTL value in small-world network. (a) "One-peer-one-query" mode. (b) "One-peer-multi-query" mode. (c) Peer failure. (d) Churn.

mode, but the difference is quite small. In the "one-peer-multi-query" mode, PQR consumes more update messages than APS when TTL <10. However, when TTL ≥ 10, PQR performs better than APS. Figs. 14(c) and (d) show similar trends to that in Fig. 14(b), both APS and PQR consume the most update messages under peer failure.

4.4. Discussion

We deploy only 1 and 3 walkers in our experiments in order to compare the query routing performance directly. With the walkers increasing, performances of three query routing strategies become more and more closer, which deviates from our goal—achieving high query hit rate with low bandwidth consumption. Network topologies can affect the query routing performance to a certain extent. PQR benefits most from small-world network because of its characteristics of small path length and high clustering coefficient. Parameters' values are fixed in the simulation of network churn because their effects on the three query routing strategies are the same. The space cost of *QueryTable* and *PeerTable* is not evaluated in this paper because both *QueryTable* and *PeerTable* are two vectors with linear length, which are negligibly small compared with that of TGM at each peer. The length of *PeerTable* does not exceed the length of neighbor table and the length of *QueryTable* is also not large because it records only the queries with different query objects set.

From the extensive experiments carried out in this paper, we can see that PQR is more capable to reduce the consumption of messages for query answering in different networks compared with the other two compared strategies. In other words, the number of query processing messages of the proposed method is smaller than that of the other two ones. The reason lies in two aspects. First, successful query history information recorded in TGMs can help to improve query hit rate. If a query request does not hit, it will produce TTL query processing messages; otherwise, it will produce less query processing messages unless the query request hits at the last node. PQR achieves higher query hit rate than the other two strategies. These successful queries normally produce less query processing messages than those produced by unsuccessful queries in the other two strategies. Second, with the increase of TTL value, many overlapping edges in different query hit paths emerge in our algorithm. Thus, some queries may have higher probability of choosing shorter paths to the other target peers.

5. Conclusion

In this paper, we propose a novel query routing mechanism for improving query performance in unstructured P2P networks. We design a data structure called traceable gain matrix (TGM) that records every query's gain at each peer along the query hit path, and allows for optimizing query routing decision effectively. When a peer receives a query message, it calculates the gain values of its neighbors to which the query will be forwarded and terminated with query hit. The neighbor with the highest gain value of current query will be selected as candidate with the highest probability to forward the query message. Experimental results show that our query routing mechanism can achieve relatively high query hit rate with low bandwidth consumption in different types of network topologies under static and dynamic network conditions.

Acknowledgments

This work was supported by National Natural Science Foundation of China under Grant nos. 60873040 and 60873070. Shuigeng Zhou was also supported by Shanghai Leading Academic Discipline Project No. B114 and the Open Research Program of Shanghai Key Lab of Intelligent Information Processing. Jihong Guan was also supported by the Program for New Century Excellent Talents at the University of China (NCET-06-0376) and the "Shu Guang" Program of Shanghai Municipal Education Commission and Shanghai Education Development Foundation.

References

Albert R, Barabási AL. Statistical mechanics of complex networks. Reviews of Modern Physics 2002;74:47–97.
Almeida V, Bestavros A, Crovella A, Oliveira AM. Characterizing reference locality in the WWW. In: Proceedings of 4th international conference on parallel and distributed information systems, 1996. p. 92–103.
Barabási AL, Albert R. Emergence of scaling in random networks. Science 1999;286:509–12.
Bisnik N, Abouzeid A. Optimizing random walk search algorithms in P2P networks. Computer Networks 2007;51(6):1499–514.
Chawathe Y, Ratnasamy S, Breslau L, Lanham N, Shenker S. Making gnutella-like p2p systems scalable. In: Proceedings of SIGCOMM, 2003. p. 407–18.
Clarke I, Sandberg O, Wiley B, Hong TW. Freenet: a distributed anonymous information storage and retrieval system. In: Workshop on design issues in anonymity and unobservability, 2000. p. 311–20.
Cohen E, Fiat A, Kaplan H. Associative search in peer to peer networks: harnessing latent semantics. Computer Networks 2007;51(8):1861–81.
Connelly BD, Bowron CW, Xiao L, Tan PN, Wang C. Adaptively routing P2P queries using association analysis. In: Proceedings of the 35th international conference on parallel processing, 2006. p. 281–8.
Crespo A, Molina HG. Routing indices for peer-to-peer systems. In: Proceedings of the 22nd international conference on distributed computing systems, 2002. p. 23–34.
Erdös P, Rényi A. On random graphs. Publicationes Mathematicae 1959;6:290–7.
Erdös P, Rényi A. On the evolution of random graphs. Publications of the Mathematical Institute of the Hungarian Academy of Sciences 1960;5:17–61.
Erdös P, Rényi A. On the strength of connectedness of a random graph. Acta Mathematica Scientia Hungary 1961;12:261–7.
Frankel J, Pepper T. Gnutella, 2000. Available online at 〈http://www.gnutella.com〉.
Gkantsidis C, Mihail M, Saberi A. Random walks in peer-to-peer networks. In: Proceedings of IEEE conference on computer communications, 2004. p. 7–11.
Gkantsidis C, Mihail M, Saberi A. Hybrid search schemes for unstructured peer-to-peer netfworks. In: Proceedings of IEEE conference on computer communications, 2005. p. 1526–37.

Heinla A, Kasesalu P, Tallinn J. KaZaA, 2001. Available online at 〈http://www.kazaa.com〉.
Heinla A, Kasesalu P, Tallinn J. Skype, 2003. Available online at 〈http://www.skype.com〉.
Jesi GP. PeerSim HOWTO: build a new protocol for the PeerSim 1.0 simulator, 2003. Available at 〈http://peersim.sourceforge.net〉.
Jiang S, Guo L, Zhang X, Wang HD. LightFlood: minimizing redundant messages and maximizing the scope of peer-to-peer search. IEEE Transactions on Parallel and Distributed Systems 2008;19(5):601–14.
Kleinberg J. The small-world phenomenon: an algorithmic perspective. Cornell Computer Science Technical Report, TR99-1776; 2000.
Kumar A, Xu J, Zegura EW. Efficient and scalable query routing for unstructured peer-to-peer networks. In: Proceedings of IEEE conference on computer communications, 2005. p. 1162–73.
Lv Q, Cao P, Cohen E, Li K, Shenker S. Search and replication in unstructured peer-to-peer networks. In: Proceedings of the 16th international conference on supercomputing, 2002. p. 84–95.
Michlmayr E. Ant algorithms for search in unstructured peer-to-peer networks. In: Proceedings of the 22nd international conference on data engineering workshops, 2006. p. 142.
Morselli R, Bhattacharjee B, Srinivasan A, Marsh MA. Efficient lookup on unstructured topologies. In: Proceedings of the 24th annual ACM symposium on principles of distributed computing, 2005. p. 77–86.
Palmer CR, Steffan JG. Generating network topologies that obey power laws. In: Proceedings of IEEE GLOBECOM, 2000. p. 434–8.
Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. In: Proceedings of ACM SIGCOMM, 2001. p. 161–72.
Rowstron A, Druschel P. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM international conference on distributed systems platforms, 2001. p. 329–50.
Sarshar N, Boykin PO, Roychowdhury VP. Percolation search in power law networks: making unstructured peer-to-peer networks scalable. In: Proceedings of IEEE peer-to-peer computing, 2004. p. 2–9.
Stoica I, Morris R, Karger D, Kaashoek F, Balakrishnan H. Chord: a scalable peer-to-peer lookup service for Internet applications. In: Proceedings of ACM SIGCOMM, 2001. p. 149–60.
Tsoumakos D, Roussopoulos N. Adaptive probabilistic search for peer-to-peer networks. In: Proceedings of 3rd international conference on peer-to-peer computing, 2003. p. 102–9.
Watts D, Strogatz S. Collective dynamics of 'small-world' networks. Nature 1998;393:440–2.
Zhang R, Hu YC. Assisted peer-to-peer search with partial indexing. IEEE Transactions on Parallel and Distributed Systems 2007;18(8):1146–58.
Zhao BY, Kubiatowicz J, Joseph A. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. U.C. Berkeley Technical Report UCB/CSD-01-1141; 2001.