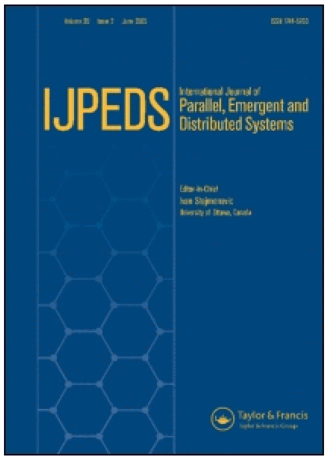


International Journal of Parallel, Emergent and Distributed Systems

Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713729127

A hybrid searching scheme in unstructured P2P networks

Online Publication Date: 01 January 2007
To cite this Article: Li, Xiuqi and Wu, Jie , (2007) 'A hybrid searching scheme in unstructured P2P networks', International Journal of Parallel, Emergent and Distributed Systems, 22:1, 15 - 38
To link to this article: DOI: 10.1080/17445760601029412
URL: http://dx.doi.org/10.1080/17445760601029412

The International Journal of Parallel, Emergent and Distributed Systems,
Vol. 22, No. 1, February 2007, 15–38

A hybrid searching scheme in unstructured P2P networks

XIUQI LI* and JIE WU†

Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA

(Received 9 June 2006; in final form 26 August 2006)

The existing searching schemes in peer-to-peer (P2P) networks are either forwarding-based or non-forwarding based. In forwarding-based schemes, queries are forwarded from the querying source to the query destination nodes. These schemes offer low state maintenance. However, querying sources do not entirely have control over query processing. In non-forwarding based methods, queries are not forwarded and the querying source directly probes its neighbors for the desired files. Non-forwarding searching provides querying sources flexible control over the searching process at the cost of high state maintenance. In this paper, we seek to combine the powers of both forwarding and non-forwarding searching schemes. We propose an approach where the querying source directly probes its own extended neighbors and forwards the query to a subset of its extended neighbors and guides these neighbors to probe their own extended neighbors on its behalf. Our approach can adapt query processing to the popularity of the sought files without having to maintain a large set of neighbors because its neighbors' neighbors are also in the searching scope due to the 1-hop forwarding inherent in our approach. It achieves a higher query efficiency than the forwarding scheme and a better success rate than the non-forwarding approach. To the best of our knowledge, the work in this paper is the first one to combine forwarding and non-forwarding P2P searching schemes. Experimental results demonstrate the effectiveness of our approach.

Keywords: Hybrid; Peer-to-peer networks; Unstructured P2P; Searching scheme

1. Introduction

Peer-to-peer (P2P) networks have been widely used for information sharing. In such systems, all nodes play equal roles and the need of expensive servers is eliminated. P2P networks are overlay networks, where each overlay link is actually a sequence of links in the underlying network. P2P networks are self-organized, distributed and decentralized. In addition, they can gather and harness the tremendous computation and storage resources on computers in the entire network. P2P networks can be classified as *unstructured*, *loosely structured* and *highly structured* based on the control over data location and network topology [1], as shown in figure 1. In this paper, we are concerned with unstructured P2P networks because they are

*Corresponding author. Email: xli@cse.fau.edu
†Email: jie@cse.fau.edu

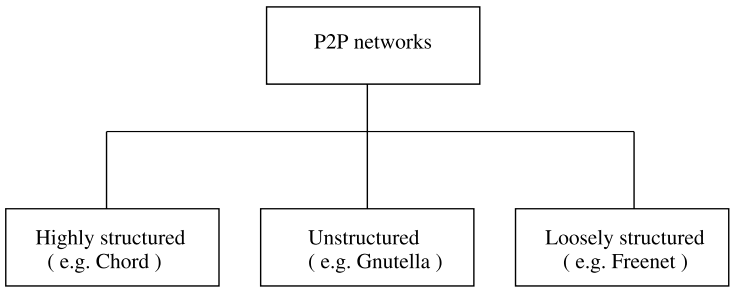

P2P networks
Highly structured (e.g. Chord)
Unstructured (e.g. Gnutella)
Loosely structured (e.g. Freenet)

Figure 1. P2P network classification.

the most widely used systems in practice. In such systems, no rule exists that defines where data is stored and the network topology is arbitrary.

Searching is one of the most important operations in P2P networks. Figure 2 illustrates the classification of existing P2P searching techniques. Most of them are based on *forwarding* [1]. In such schemes, a query is forwarded on the overlay from the querying source toward the querying destinations where the desired data items are located. The query forwarding stops when the termination condition is satisfied. Forwarding schemes offer low state maintenance. Each node only needs to keep a small number of neighbors. However, the querying source has no control over query processing. Once the query is forwarded, the querying source has no influence on the number of nodes that receive the query and in which order these nodes receive the query. Too many nodes are searched for popular data items while not enough nodes are examined for rare ones. Therefore, the forwarding-based approach does not offer query flexibility and has low query efficiency.

Non-forwarding schemes were proposed in Refs. [2,3]. In these approaches, queries are not forwarded. Instead, the querying source directly probes its neighbors for the data items it desires. Thus the querying source has full control over query processing. The extent of a search is determined by the querying source. For popular items, only a small number of nodes need to be searched. For rare items, a large number of nodes are queried. No resource is wasted to search for popular items. However, to find rare items, each node has to maintain (dynamically recruit) a large number of living neighbors because it relies solely on its own

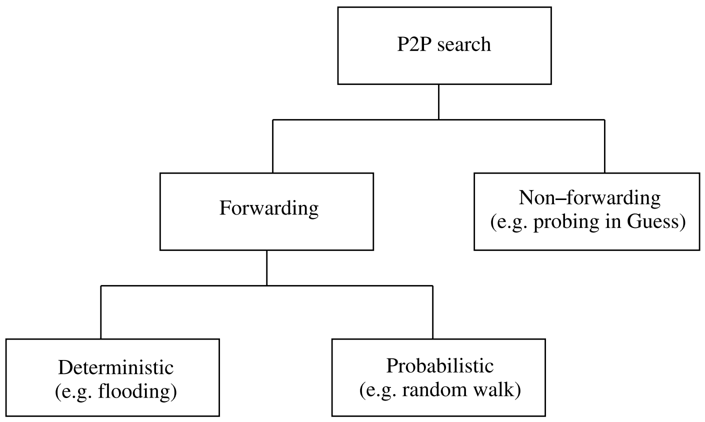

P2P search
Forwarding
Non–forwarding (e.g. probing in Guess)
Deterministic (e.g. flooding)
Probabilistic (e.g. random walk)

Figure 2. P2P search classification.

neighbors for finding a data item. The system has to either carry a large overhead to keep a large number of neighbors alive or leaves queries unsatisfied with a low state maintenance overhead because the number of living neighbors that a node is aware of is not enough for finding rare items.

In this paper, we seek to combine these two schemes to get their advantages while lowering their disadvantages. Our goals are to advocate the integration of both schemes, to explore different methods for integration, and to evaluate the integrated schemes. We propose an approach that is a unification of direct query probing and guided 1-hop forwarding. Given a query, the querying source directly probes its own extended neighbors for the desired files and forwards the query to a selected number of neighbors. These neighbors will probe their own extended neighbors on behalf of and under the guidance of the querying source and will not forward the query further. When the query termination condition is satisfied, the querying source terminates its own probing and the probing of its neighbors.

The main contributions of this paper are the following:

- We identify the necessity to integrate both the forwarding schemes and non-forwarding schemes into one approach.
- We devise a hybrid approach that combines both the forwarding and non-forwarding schemes. This hybrid approach achieves query flexibility, query efficiency and query satisfaction without a large state maintenance overhead. To the best of our knowledge, this work is the first one to combine both schemes.
- We investigate different design tradeoffs in integrating the forwarding and non-forwarding approaches. These choices include constant integration and adaptive integration. We point out their pros and cons and offer some practical advice in applying them to real world systems.
- We put forward two new policies for recruiting new neighbors, called *Most Files Shared in Neighborhood (MFSN)* and *Most Query Results in Neighborhood (MQRN)*. The nodes with more files and more past query results in its neighborhood are recruited first.
- We evaluate our hybrid approach against both the forwarding schemes and non-forwarding schemes and demonstrate the performance improvement in our hybrid approach through simulations.

This paper is organized as follows. In Section 2, the forwarding and non-forwarding searching schemes in unstructured P2P networks are reviewed. In Section 3, the proposed hybrid approach is overviewed and contrasted with the forwarding and non-forwarding schemes. In Section 4, the details about the hybrid approach, such as action queue (AQ) computation, different integration design choices including constant integration and adaptive integration, and state maintenance are discussed. In Section 5, the experimental setup and results are described. At the end, our work is summarized and a future plan is identified.

2. Related work

Most searching schemes in unstructured P2P networks are forwarding-based and are different variations of flooding. They can be classified as deterministic or probabilistic [1]. In a deterministic approach, each node forwards a query to a deterministic number of neighbors when it receives a query. The iterative deepening [4] and local indices [4] belong to this category. In a probabilistic approach, each node that receives a query forwards that query to a subset of its neighbors randomly, probabilistically, or based on ranking. This category includes *k*-walker random walk [5], modified random BFS [6], directed BFS [4], adaptive probabilistic search [7], dominating set based search [8], RNG-based search [9], GES [10], scalable query routing (SQR) [11], DiCAS [12], the approaches in Refs. [13,14].

Forwarding schemes in unstructured P2Ps can also be classified as blind search or informed search [1]. In blind searches, such as *k*-walker random walk and modified random BFS, nodes do not keep any information about the data location. In informed searches, for example, directed BFS and SQR, nodes store some hints that facilitate the search. The directed BFS utilizes simple hints while SQR takes advantage of complicated hints. Forwarding schemes can also be classified as regular-grained (e.g. SQR) or coarse-grained (e.g. dominating set based search and RNG-based search) based on whether some nodes in the network are shielded from the query forwarding.

In contrast, there are only two non-forwarding schemes for searching unstructured P2Ps in the research literature. The non-forwarding concept was first proposed in GUESS [2]. In this approach, each node fully controls the entire process of its own queries. Each node directly probes its own neighbors in a sequential order until the query is satisfied or until all neighbors have been probed. The query fails in the latter case. Each node uses a *link cache* to keep the information about neighbors, which includes the IP, the time stamp, the number of files shared, and the number of results from the most recent query. There is one entry for each neighbor in the link cache. These link cache entries are refreshed through periodic pings. In addition, to add new neighbors into the link cache, each node also requests that its neighbors select a certain number of their own link cache entries and return them in the pongs during the periodic pings.

Because of the overhead of link cache maintenance, the link cache size cannot be too large. To accommodate this problem, when a neighbor is probed during the query processing, it also returns some of its own link cache entries in a separate query pong message. These link cache entries are stored in another cache, called *query cache*. Each entry in the query cache has the similar content to that in the link cache. Some entries in the query cache may be moved to the link cache. However, the entries in the query cache are not maintained.

The performance of GUESS is improved by in Ref. [3], which emphasizes the impacts of different design choices called policies in non-forwarding schemes. The policies are classified into five types: QueryProbe, QueryPong, PingProbe, PingPong and CacheReplacement. The QueryProbe and PingProbe stipulate the order in which neighbors are probed during the query processing and the periodic state maintenance respectively. The QueryPong and PingPong specify the preference for selecting link cache entries in response to a query and a periodic ping correspondingly. For each policy type, many specific policies may be adopted. Five common policies, which include random (RAN), most recently used (MRU), least recently used (LRU), most files shared (MFS) and most results (MR), are proposed for these policy types. The parallel probing of *k* neighbors is also briefly mentioned.

It should be noted that the non-forwarding concept has been proposed for structured P2Ps as well. In Ref. [15], a 1-hop lookup querying scheme is designed to remove the overhead of query message forwarding in structured P2Ps. However, due to the basic topology differences between unstructured P2Ps and structured P2Ps, the research issues related to the non-forwarding searching for these two types of P2Ps vary dramatically.

3. Outline of the hybrid search

Figure 3 illustrates the differences between the three types of searching approaches, forwarding based, non-forwarding based and hybrid. In the figure, a node's children refer to some or all its neighbors in the P2P overlay. Forwarding-based searching can be regarded as a *D*-level tree rooted at the querying source as shown in figure 3(a). *D* refers to the maximum TTL value. The querying source, denoted by a triangle, checks its local datastore and forwards the query to its children nodes. These children, denoted by solid squares, look up their local datastores and forward the query to their own children. This process continues until the search terminates successfully at a leaf node that is not at Level-*D* or the search fails at a leaf node that is at Level-*D*. It is observed that once the query is forwarded, the querying source cannot control how the nodes on this tree process the query. Each node just needs to maintain a small number of neighbors because nodes within *D* hops of the querying source are potentially in the searching scope.

Non-forwarding based searching is shown in figure 3(b). It is a 1-level tree rooted at the querying source. The querying source directly probes its child nodes for the desired files. These children only search their local datastores and do not send the query further. The querying source terminates the search when the query is satisfied or when all its neighbors are probed. Only the querying source and its direct neighbors are involved in the processing of a particular query. Therefore, each node must maintain a sufficient number of live neighbors. These neighbors are dynamically recruited and updated via periodical ping-probes and ping-pongs.

The hybrid searching is illustrated in figure 3(c). It is a 2-level tree rooted at the querying source. The querying source directly probes the nodes at Level-1 of the tree. In the mean time, it also forwards the query to the internal nodes at Level-1 and guides these nodes to probe the nodes at Level-2 on its behalf. The querying source terminates the search when the

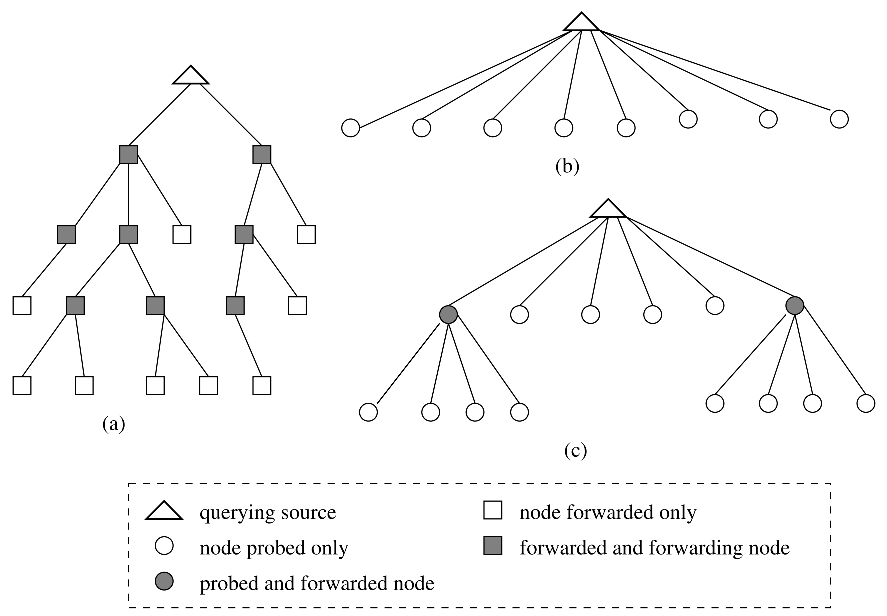


Figure 3. The three types of P2P searches: (a) forwarding based; (b) non-forwarding based; (c) hybrid.

query is satisfied or when all its neighbors and its neighbors' neighbors are probed. The maximum searching scope for a query in this approach is the 2-hop neighborhood of the querying source.

Like the non-forwarding approach, a node in the hybrid approach maintains an extended neighbor set and dynamically recruits and updates this neighbor set via periodic ping-probes and ping-pongs. However, the hybrid approach can achieve the same or higher query satisfaction with less neighbors per node. Compared to the forwarding-based approach, the querying source in the hybrid approach can control the extent of the searching.

To combine the forwarding and non-forwarding smoothly, the hybrid search is implemented as follows. It considers three types of actions, *probing only*, *forwarding only*, *probing and forwarding*. *Probing only* means that the querying source probes its neighbors and these neighbors look up their local datastores. *Forwarding only* means that the querying source does not probe its neighbors but guides its neighbors to probe their own neighbors on its behalf. *Probing and forwarding* means the combination of the first two actions.

When processing a query, the querying source first ranks these three types of actions if performed on all its neighbors and organizes these actions into an *action queue*. Two examples of AQs are shown in figure 4(b). The final AQ contains six actions listed in the descending order of their ranks, probe node B_1, probe node B_4, probe and forward to node B_3, forward to node B_4, probe and forward to node B_2, and forward to node B_1. The querying source then takes actions in this queue in order. It can take actions at a constant rate of k_1 actions at once, which is called *constant integration*. It can also take actions at a variable rate depending on the rareness of the sought files, which is referred to as *adaptive integration*. The querying source terminates the entire searching process when the query is satisfied or when all actions in the queue have been taken.

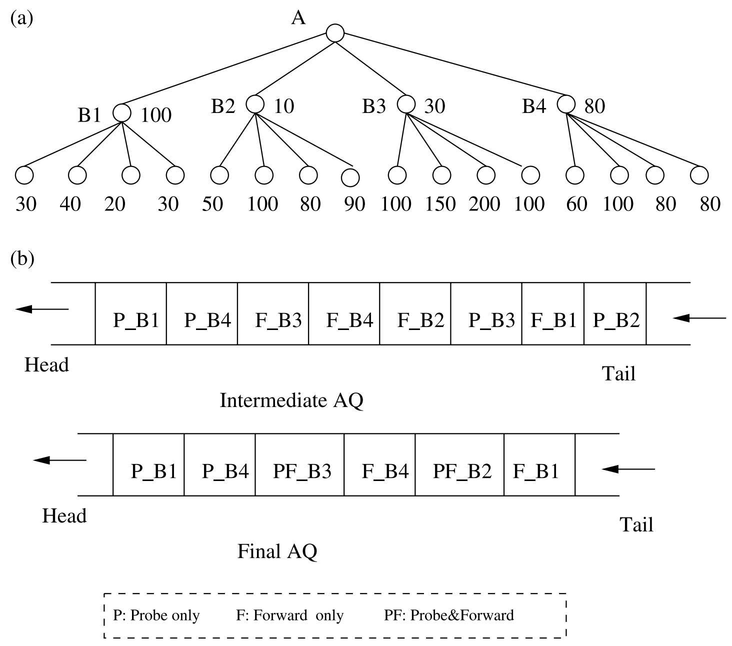


Figure 4. An example of AQ computation: (a) the querying source A and its 2-hop neighborhood, and the file distribution; (b) the computed AQ (the intermediate and final results).

The action ranking considers both the costs and gains of actions. The cost of an action is the time (in terms of the number of overlay hops) it takes for that action to be completed. The gain of an action is the estimated probability of that action for returning query results, which are determined by the system policies. These policies can also be used by the neighbors of the querying source for probing their own neighbors on behalf of the querying source .

To keep information about neighbors, each node actively maintains a *link cache*. There is one entry per neighbor. These entries are periodically updated (deleting dead entries, replacing existing entries using new entries) according to system policies. We propose two new policies, *Most Files Shared On Neighborhood (MFSN)* and *Most Query Results on Neighborhood (MQRN)*.

4. The hybrid search

The hybrid search involves the querying source and its neighbors. The processing at these nodes is shown in Algorithms 1 and 2. Given a query q, the querying source s first computes the AQ based on the discussion in Section 4.1. If constant integration is adopted, s takes the first k_1 actions in AQ at the same time. k_1 is a system parameter. P, F, or PF messsages are sent to the intended neighbors according to the action types. When v receives P or PF messages, it looks up its datastore and returns the query results if there are any. When v receives F or PF messages, it probes its own neighbors on behalf of s with k_2 neighbors per

Algorithm 1 The hybrid search at the querying source s

```
1: Compute the action queue AQ for the query $q$ based on the description in Section 4.1;
2: if the integration design is constant then
3:   while $q$ is not satisfied AND AQ is not empty do
4:     remove the first $k_1$ actions from AQ and store them in the array $ACT_k$;
5:     for $i = 0$ to $k_1 - 1$ do
6:       if $ACT_k[i]$ is ProbeOnly then
7:         send $P$ message to the intended node;
8:       else if $ACT_k[i]$ is ForwardOnly then
9:         send $F$ message to the intended node;
10:        add this node to the set: FWDed;
11:      else
12:        send $PF$ message to the intended node;
13:        add this node to the set: FWDed;
14:      end if
15:    end for
16:    if $s$ receives query results from a neighbor $v$ then
17:      store the query results in the array QRes;
18:      if $v$ has probed all its neighbors then
19:        remove $v$ from the set FWDed;
20:      end if
21:    end if
22:  end while
23: else
24:   call the algorithm adaptive_integration_search in Section 4.2;
25: end if
26: if $q$ is satisfied then
27:   Order each node in FWDed to stop probing on behalf of $s$;
28: end if
```

Algorithm 2 The hybrid search at the querying source s's neighbor v

```
1: if $v$ receives a $P$ message then
2:    $v$ checks its local datastore and returns a query result to $s$ if the result is found;
3: else if $v$ receives a $F$ message then
4:    $v$ probes its own neighbors on behalf of $s$ at the rate of $k_2$ nodes per probe;
5: else
6:    $v$ checks its local datastore and returns a query result to $s$ if the result is found;
7:    $v$ probes its own neighbors on behalf of $s$ at the rate of $k_2$ nodes per probe;
8: end if
```

probe. k_2 is also a system parameter. If s receives any query result from a neighbor v, s stores that result. If adaptive integration is employed, follow the detailed algorithm in Section 4.2. When q is satisfied, s stops its own probing and the probing performed by its neighbors on its behalf.

4.1 Action queue computation

The AQ is computed based on the gain/cost ratios of the actions if they are performed on the querying source's neighbors. We intend to use the number of query results per hop as the gain/cost ratio. The cost of an action is the time (in terms of the number of overlay hops) taken for that action to be completed. The gain of an action is the estimated probability of that action for returning query results. This probability is computed based on the system policy on estimating nodes' query-answering ability. Possible policies are RAN, MRU, most files (MF) and most query results (MR). The AQ computation algorithm varies according to the chosen system policy.

If the system policy is random, the AQ is a random sequence of *ProbeOnly* actions on all neighbors of the querying source s followed by a random sequence of *ForwardOnly* actions on those neighbors. If the system policy is MRU, the AQ is a sequence of *ProbeOnly* actions on s's neighbors, followed by a sequence of *FowardOnly* actions on those neighbors. Both sequences are sorted in the descending order of the timestamp when s interacted with these neighbors regardless of which party initiated the interation. No *Probe&Forward* action is involved in these two policies to reduce the query traffic.

If the system policy is MF, the AQ is computed according to Algorithm 3. The gain/cost ratio of a *ProbeOnly* action on a neighbor v, denoted by PGCR_v, is computed using the following formula. $\mathrm{Num}\,F_v$ represents the gain of the action. It is the number of files on

Algorithm 3 The AQ computation at the querying source s for policies MF and MR

```
1: compute the gain/cost ratios of the actions ProbeOnly and ForwardOnly if performed on each neighbor $v$;
2: sort these actions in the descending order of their gain/cost ratios and store the result in the linked list $AQ$.
3: if a node $v$ exists such that the action $ForwardOnly\ \ to\ \ v$ precedes action $Probe\ \ v\ \ Only$ in $AQ$ then
4:    replace the action $FowardOnly\ \ to\ \ v$ by $Probe\ \ v\ \ and\ \ Forward\ \ to\ \ v$;
5:    remove the action $Probe\ \ v\ \ Only$ from $AQ$;
6: end if
```

node v. 2 is the cost of this action, two overlay hops.

$$\mathrm{PGCR}_v = \frac{\mathrm{Num}\, F_v}{2}$$

The gain/cost ratio of a *ForwardOnly* action on a neighbor v, denoted by FGCR_v, is calculated according to the following formula. NB_v refers to the set of neighbors of node v. $\mathrm{Num\,F}_u$ refers to the number of files on u. d_v represents the degree of node v. k_2 is the system parameter mentioned earlier. The gain of this action is the total number of files on v's neighbors. The cost of this action is the denominator where 1 means that it takes one hop for the querying source s to send a F message to v, $2d_v/k_2$ represents the time taken for v to finish probing all its neighbors at the rate of k_2 nodes per probe, d_v/k_2 denotes the time taken for v to return all query results found on its neighbors to s, and γ refers to the penalty weighting factor because probing and forwarding are considered together in action ranking.

$$\mathrm{FGCR}_v = \frac{\sum_{u \in \mathrm{NB}_v} \mathrm{Num\,F}_u}{\gamma(1 + 2d_v/k_2 + d_v/k_2)}$$

If the system policy is most query results, the AQ computation is similar to that of MF. The only difference is that the number of files on node u and v are replaced by the number of query results for the most recent query on u and v, respectively.

An example of AQ computation is shown in figure 4 and table 1. Suppose that the querying source A, its neighbors B_1, B_2, B_3, B_4, and its neighbors' neighbors are the same as that in figure 4(a). The numbers next to each node refers to the number of files on that node. Assume that the system policy for estimating nodes' query-answering ability is MF, $k_2 = 2$, and $\gamma = 2$. We first consider the *ProbeOnly* and *ForwardOnly* actions if performed on each neighbor of A. The gain/cost ratios of these actions are illustrated in table 1. Take node B_4 as an example. The gain/cost ratio of the action *Probe B_4 only* is $80/2 = 40$. The gain/cost ratio of the action *Forward to B_4 only* is

$$\frac{60 + 100 + 80 + 80}{2\left(1 + \frac{2\times 4}{2} + \frac{4}{2}\right)} \doteq 23.$$

Then we sort these actions in the descending order of their gain/cost ratios and get the intermediate AQ as shown in figure 4(b). Because *Forward to B_3 only* (*F_B_3*) action appears before *Probe B_3 only* (*P_B_3*) action in the intermediate AQ, they are combined into one action *Probe B_3 and Forward to B_3* (PF_B_3). Similarly the actions *F_B_2* and *P_B_2* are combined into the action PF_B_2. The final AQ is shown in figure 4(b).

Table 1. The gain/cost ratios of *ProbeOnly* and *ForwardOnly* actions if performed on *A*'s neighbors.

Node	*ProbeOnly*	*ForwardOnly*
B1	50	8.5
B2	5	23
B3	15	40
B4	40	23

4.2 Integration design

We consider two ways to integrate forwarding and probing, *constant integration* and *adaptive integration*. In constant integration, the querying source s takes actions in the AQ at a constant speed (k_1 actions each time where k_1 is determined experimentally). In adaptive integration, s adjusts its action-taking progress according to the rareness of the sought files. The rarer, the more progressive. There are many options for adaptive integration. One simple example is to adjust the progress according to the following formula. α denotes the number of actions taken by s each time. α is initialized to α_0 and is increased by β actions for every NumN nodes that have been searched since last update. NumN serves as an update interval. NumNSoFar is the total number of nodes that have been searched since the beginning of the query processing. α_0 and β will be determined experimentally. The neighbors of the querying source s must report their probing progress to s. The hybrid search in the case of adaptive integration is shown in Algorithm 4. The main difference is that s must initialize α before processing a query q and update α while processing q.

$$\alpha = \alpha_0 + \lfloor \frac{\text{NumNSoFar}}{\text{NumN}} \rfloor \beta$$

Algorithm 4 The *adaptive_integration_search* at the querying source s (called by Algorithm 1)

```
1: Initialize $\alpha$;
2: while $q$ is not satisfied AND $AQ$ is not empty do
3:   remove the first $\alpha$ actions from $AQ$ and store them in the array $ACT_k$;
4:   for $i = 0$ to $\alpha - 1$ do
5:     if $ACT_k[i]$ is ProbeOnly then
6:       send $P$ message to the intended node;
7:     else if $ACT_k[i]$ is ForwardOnly then
8:       send $F$ message to the intended node;
9:       add this node to the set: FWDed;
10:    else
11:      send $PF$ message to the intended node;
12:      add this node to the set: FWDed;
13:    end if
14:  end for
15:  if $s$ receives query results from a neighbor $v$ then
16:    store the query results in the array QRes;
17:    if $v$ has probed all its neighbors then
18:      remove $v$ from the set FWDed;
19:    end if
20:  end if
21:  if the interval for updating $\alpha$ arrives then
22:    update $\alpha$ accordingly;
23:  end if
24: end while
```

4.3 Query probing

Both the querying source s and its neighbors perform probing during the processing of a query. The probing performed by s is considered together with forwarding in the AQ computation. This subsection discusses the probing performed by s's neighbors on its behalf as a result of query forwarding. This probing is at the rate of k_2 nodes per probe. It is guided by the same system policy for estimating nodes' query-answering ability that was chosen in AQ computation.

Suppose that v is a neighbor of s. If the system policy is random, v randomly chooses k_2 of its own neighbors that have not been probed and probes these neighbors concurrently. If the system policy is MRU, v selects k_2 of its own neighbors that have not been probed and have the latest timestamps among all of its unprobed neighbors. If the system policy is MF or MR, v chooses k_2 unprobed neighbors that have the top number of files or top number of query results for the most recent query.

4.4 The state maintenance

Like the non-forwarding based searching, each node uses a link cache to maintain information about neighbors. However, link cache entries in the hybrid approach have different content because a node needs to know the information about a neighbor and this neighbor's neighbors. Table 2 shows the data structure of the link cache entry for neighbor *B* at node *A* in the hybrid approach. It should be noted that the *TS* is updated no matter which party, *A* or *B*, initiates the interaction and what type of interaction it is.

The link cache is refreshed and updated through periodic pings. Each node periodically selects some of its neighbors and sends Ping messages to these neighbors. These neighbors reply with Pong messages that include the latest information about themselves and a selected number of entries in their own link caches. The ping interval is a system parameter. There are three types of system policies that specify how the periodic pings are conducted. They are PingProbe policy, PingPong policy and CacheReplacement policy. The PingProbe policy specifies the neighbor selection rule for sending Pings. The PingPong policy is used to select neighbors to be included in the Pong when responding to a Ping. The CacheReplacement policy determines the rule for replacing existing entries by the new entries.

For each policy type, one of the seven specific policies may be chosen, RAN, MRU, LRU, MFS on neighbor, most query results on neighbor (MR), MFSN and MQRN. The RAN, MRU, LRU, MFS and MR are similar to those in the non-forwarding approach. The MFSN and MQRN are new policies proposed in this paper. The MFSN selects the neighbor that has the most shared files in its 1-hop neighborhood including that neighbor itself. The MQRN chooses the neighbor that returns the most query results for the last query, which counts the results found on that neighbor and the results found on that neighbor's neighbors.

Table 2. The data structure of a link cache entry at node *A* for neighbor *B*.

Notation	*Definition*
IP	The IP address of *B*
TS	The last time when *A* and *B* interacts with each other
$NumFiles_P$	The number of files on *B*
$NumRes_P$	The number of query results for the last query found on *B*
$NumFiles_F$	The total number of files on *B*'s neighbors
$NumRes_F$	The total number of query results for the last query found on *B*'s neighbors

5. Experimental results

In this section, we describe the simulation setup, investigate the system configuration of the hybrid search, and evaluate the performance of the hybrid approach against existing searches in unstructured P2P networks.

5.1 Simulation setup

We created a network of NumNodes nodes. Each node's link cache is seeded with CacheSeedSize $= 5\%$ of *LinkCacheSize* neighbors. Then the neighbors are dynamically extended/updated based on the *PingProbePolicy*, *PingPongPolicy* and *CacheReplacement-Policy*. In the hybrid approach, the AQ is computed based on the *QueryAnswerAbilityPolicy*, which is used to estimate the ability of a neighbor for returning query results. Both the document replication distribution and the query distribution are zipfian distributions. As suggested in Ref. [16], we let 10 percent of the documents have around 30 percent of the total stored copies and receive around 30 percent of total query requests.

To simulate the dynamic network, we let *PctNodesChanged* nodes die periodically. It is assumed that when a node dies, another new node is born and the dead node does not return to the system. Therefore the number of living nodes in the system remains the same. We use the *random friend seeding policy* [17] to initialize the link cache of the new node. The new node introduces itself to nodes in its link cache at probability $IntroProb = 0.1$. Each node pings *PingInterval* number of neighbors in its link cache periodically. The pinged neighbor returns *PongSize* number of its own neighbors in response.

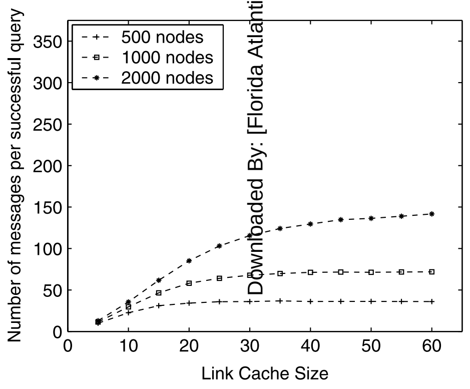
The performance measures are the average query success rate (or query unsuccess rate) and the average number of query messages per successful query. A query is a search for a single document based on the document ID. A query is considered successful if at least *NumDesiredResults* copies of the sought document are found. Table 3 lists the major system parameters and their default values.

5.2 State maintenance

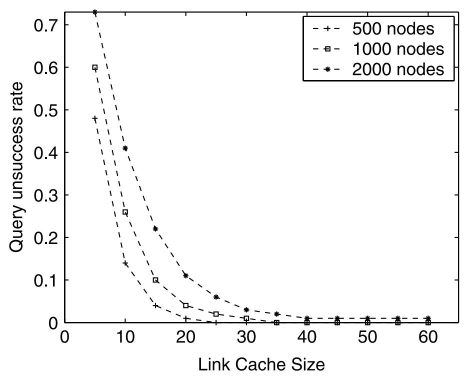
5.2.1 Varying link cache size. Figure 5 illustrates the impact of the different link cache sizes in networks of different scales for the hybrid approach. The network sizes are 500, 1000 and 2000 nodes. To isolate the effect of the link cache, we did not implement the query cache. As seen in figure 5(a), the average number of query messages per query increases as the link

Table 3. The major system parameters and their default values.

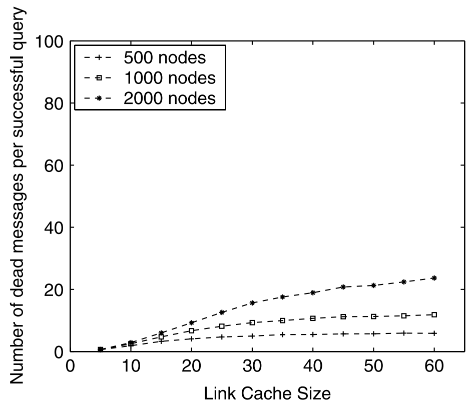
Parameter	*Default value*
NumNodes	1000
NumDesiredResults	1
PctNodesChanged	5%
LinkCacheSize	15
PingInterval	5
PongSize	5
QueryAnswerAbilityPolicy	Random
CacheReplacementPolicy	Random
PingProbePolicy	Random
PingPongPolicy	Random



(a) Number of query messages.

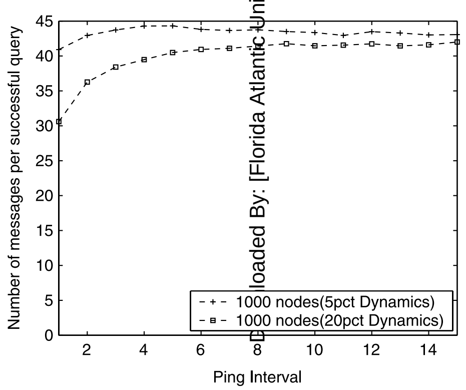


(b) Query unsuccess rate.

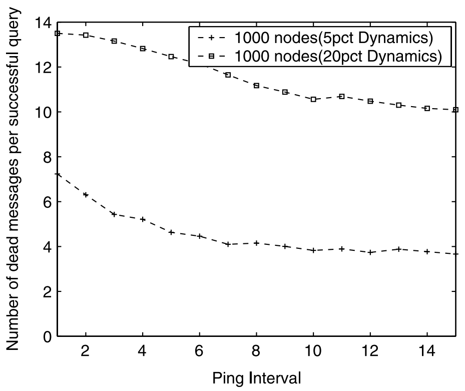


(c) Number of dead query messages.

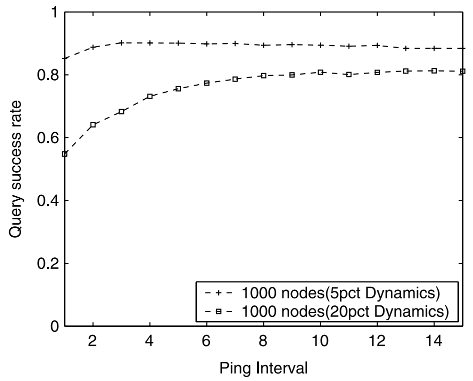
Figure 5. Varying link cache size.



(a) Number of query messages.



(b) Number of dead query messages.



(c) Query success rate.

Figure 6. Varying ping interval; Link cache size is 15; 1000 nodes with 5% and 20% of network dynamics.

cache gets larger. The query unsuccess rate drops quickly when the link cache size increases as shown in figure 5(b). When the link cache size is more than 30, the query unsuccess rate does not change much. Figure 5(c) explains the reason. More messages are sent to dead neighbors when the link cache size is larger. The networks at different scales show similar trends as the link cache size changes. These figures suggest that the appropriate values for the link cache are in the range of 15–30. The figures also imply that the hybrid approach can achieve a fairly good query performance with link caches alone. Each node does not have to maintain an extra query cache.

5.2.2 Varying ping interval. Each node pings *PingInterval* number of nodes in its link cache periodically. Larger *PingInterval* values means higher pinging frequency. Ping interval is another important factor in state maintenance. Figure 6 shows the impact of the different ping intervals on the query performance of the hybrid approach. The system settings are default except the ping interval (in the range of [1,15]) and the percentage of nodes that join or leave (5% and 20%).

Figure 6(a) and (c) indicate that when the ping interval is small (≤ 6), as *PingInterval* increases, there is an apparent increase in the query success rate and the average number of query messages per query, and a dramatic decrease in the average number of dead query messages per query. The change can be explained by figure 7. In the small ping interval setup, the number of live link cache entries increases as the nodes are pinged more frequently. Therefore there are more live candidates to search for and more queries are satisfied.

When the ping interval is large (>6) and the pinging frequency goes up, the query success rate and the number of query messages does not increase dramatically; the average number of dead query messages still drops slightly. This is caused by the fact, as shown in figure 7, that further increase in ping interval values does not bring more live cache entries. Each pinged node pongs back some of its own link cache entries. The P2P overlay is constantly changing.

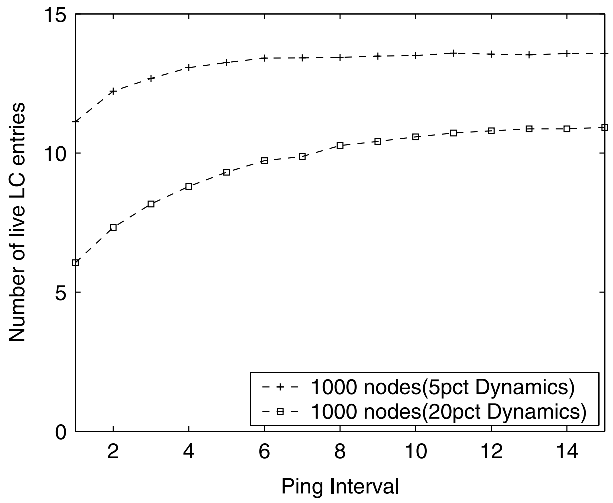


Figure 7. The number of live link cache entries with varying ping intervals. Link cache size is 15. 1000 nodes with 5% and 20% of network dynamics.

Even if all neighbors are pinged, there may still be some dead nodes ponged back. It is also observed that the ping interval has a larger influence on the query performance of more dynamic networks.

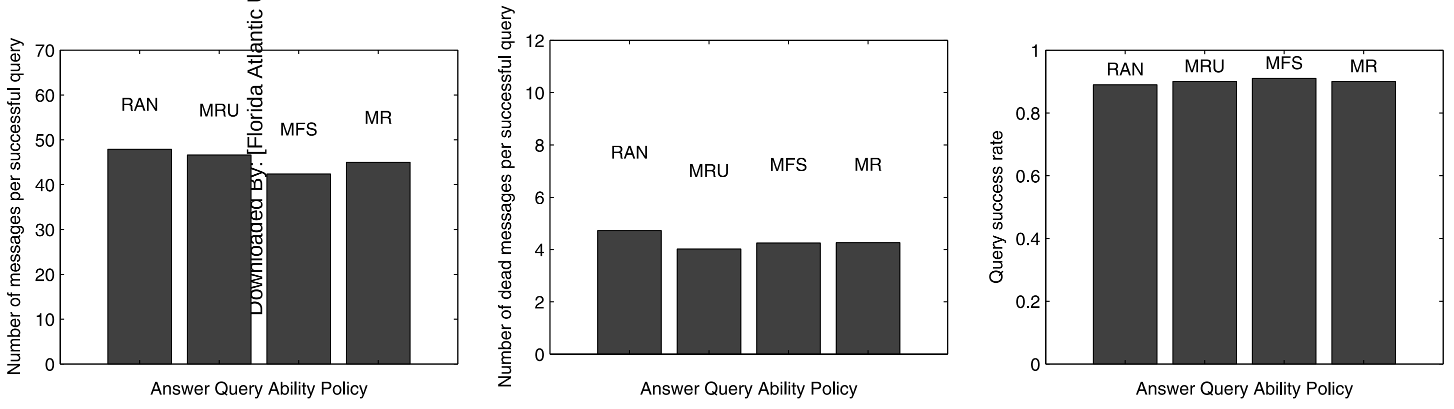
5.3 Varying system policies

In this subsection, we investigate how the query performance is affected by different system policies. There are four types of policies: query answer ability policy, cache replacement policy, ping probe policy, and ping pong policy. To find out the real influence of a policy type alone, when we check one policy type, we leave all other policy types to be *Random*. All other system settings are default values as listed in table 3.

5.3.1 Query answer ability policy. The query answer ability policy specifies how to estimate the likelihood of a node for answering a query. The policy may be set to one of the four values, *RAN*, *MRU*, *MFS* and *MR*. Their impacts on the average number of query messages and dead query messages per query, and query success rate, are shown in figure 8. The first subfigure indicates that *MFS* leads to the smallest query cost and *MR* has the second smallest query cost. *RAN* and *MRU* cause about the same query cost. The last subfigure shows that the query success rate is not heavily affected by different query answer ability policies though *MFS* satisfies the largest number of queries. Therefore, *MFS* is the best value for this policy type. *MR* is the second best choice.

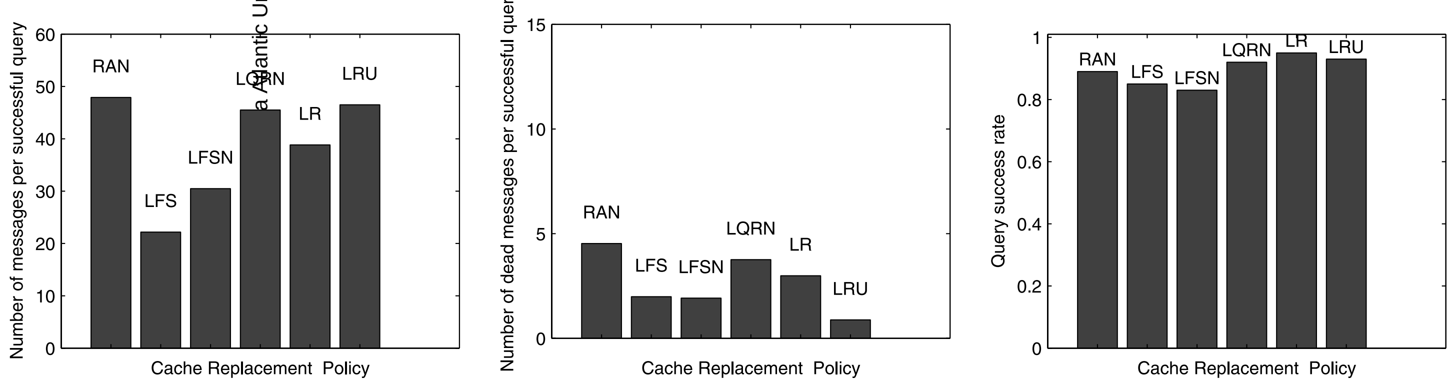
5.3.2 Cache replacement policy. Cache replacement policy is used to decide the order in which cache entries are replaced by new entries. It can take one of the six values, *RAN*, *LFS*, *LR*, *LRU*, *LFSN* and *LQRN*. Figure 9 contrasts their impacts on the query performance. The first subfigure clearly indicates that the average query cost (number of query messages per successful query) is dramatically influenced by different cache replacement policy values. *LFS* incurs the fewest average number of query messages, which is half of the costs if the policy is *RAN*, *LRU*, and *LQRN*. *LFSN* and *LR* cause the second, and the third smallest average query costs respectively. The second subfigure shows that *RAN* causes the most dead messages while *LRU* causes the least. In the third subfigure, *LR* takes the first lead in query success rate. *LR*, *LRU* and *RAN* satisfy the similar number of queries. *LFS* and *LFSN* have approximately the same query success rate with *LFSN*'s rate being the lowest but still practically acceptable. In conclusion, considering both the average query cost and the query success rate, *LFS* is the best choice for cache replacement policy. *LR* is the second best choice. These observations are consistent with the results for the query answer ability policy.

5.3.3 Ping probe policy. Ping probe policy is used for a node to determine which neighbors in its link cache should be probed first in order to recruit new neighbors. There are six possible policy values, *RAN*, *MFS*, *MR*, *MRU*, *MQRN* and *MFSN*. Their performances are shown in figure 10. The first subfigure clearly shows that *MFSN* has the smallest average query cost and the difference from the costs of other policy values are not trivial. *MFS* has the second smallest query cost, just slightly smaller than that of *MR*, *MRU*, or *MQRN*. *RAN* causes the highest query cost. In the third subfigure *MFS* has the lowest query success rate



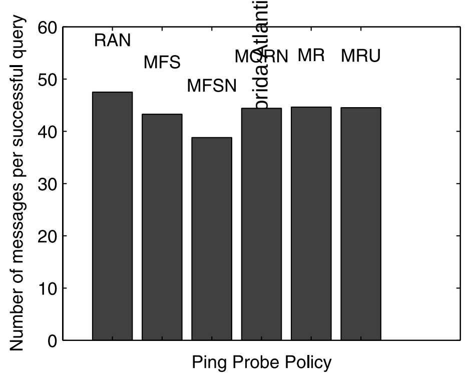
(a) Number of query messages. (b) Number of dead query messages. (c) Query success rate.

Figure 8. Varying answer query ability policies (link cache size = 15); 1000 nodes.

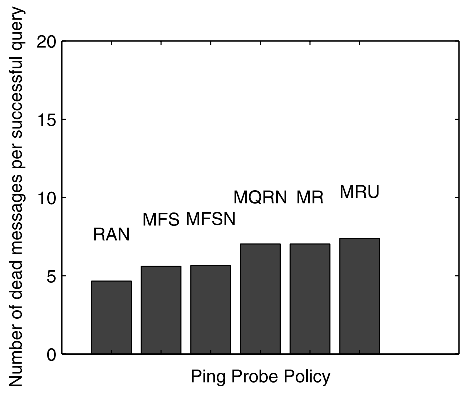


(a) Number of query messages. (b) Number of dead query messages. (c) Query success rate.

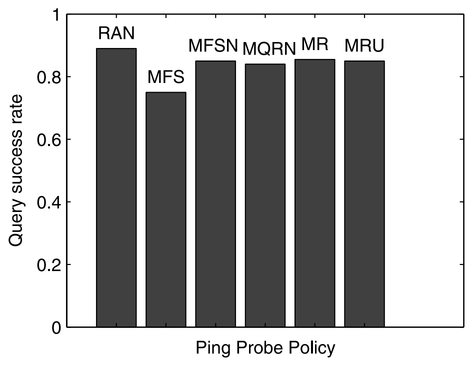
Figure 9. Varying cache replacement policies (link cache size = 15); 1000 nodes.



(a) Number of query messages.



(b) Number of dead query messages.



(c) Query success rate.

Figure 10. Varying ping probe policies (LC = 15); 1000 nodes.

(lower than 0.80). *RAN* has the highest query success rate. The other policies satisfy about the same number of queries. Therefore, *MFSN* is the best choice for ping probe policy, and *MFS* is the worst choice.

5.3.4 Ping pong policy. Ping pong policy is used for a node to determine which neighbors in its link cache should be ponged back to the pinging node as new neighbors in response to a ping probe message. There are six possible policy values, *RAN*, *MFS*, *MR*, *MRU*, *MQRN* and *MFSN*. Their performances are shown in figure 11. The first subfigure clearly shows that *MFS* has the smallest average query cost and the difference from the costs of other policy values are non-trivial. The second and third smallest average query cost are obtained by choosing *MFSN* and *MR*. This result corresponds to that in the query answer ability policy. *MQRN* and *MRU* lead to similar query costs. The average query cost for choosing *RAN* is slightly higher than that of *MQRN* and *MRU*. In the third figure, *RAN* earns the highest query success rate, which is just slightly higher than the rates achieved by *MFS*, *MR* and *MRU*. *MFSN* satisfies the least but an acceptable percentage of queries. The query success rate of *MQRN* is slightly higher than *MFSN* and lower than *MFS*, *MR* and *MRU*. Taking into consideration both the average query cost and the query success rate, the best choice for ping pong policy is *MFS*, the second best is *MR*, and the third best is *MFSN* or *RAN*.

In summary, different choices about system policies do have a major influence on the query performance of the entire system. Care must be taken when making these choices. The wrong decision, such as selecting *MFS* as ping probe policy, can make the system perform poorly. The simulation results show that *MFS* and *MR* is the best and the second best for answer query ability policy and ping pong policy. *LFS* and *LR* is the best and the second best for cache replacement policy. *MFSN* is the best ping probe policy.

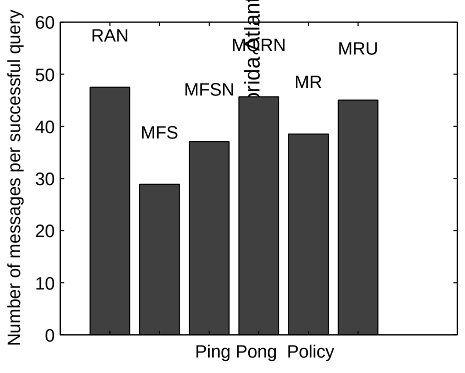
5.4 Evaluation against existing approaches

The hybrid approach is evaluated against existing forwarding and non-forwarding schemes. The forwarding approach is Gnutella-like flooding. All three approaches are simulated in the network of 1000 nodes. The overlay for the forwarding approach is random graph with an average node degree of 4. The percentage of nodes in the overlay that die/(or are born)is the same as those in the hybrid and forwarding approach.

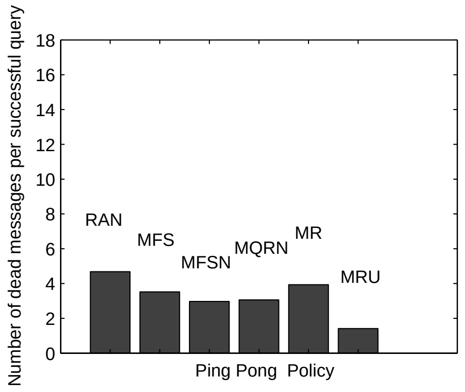
In the non-forwarding approach, a query source probes its neighbors according to the *QueryProbePolicy*, which determines the order in which neighbors are probed. Each node's link cache is seeded with the same number of neighbors as the hybrid approach. Link caches are managed based on *PingProbePolicy*, *PingPongPolicy* and *CacheReplacementPolicy*. We simulated the base line non-forwarding approach, where all policies are *Random*. The query cache is not implemented for the purpose of fair comparison because the hybrid approach does not need query caches.

In the hybrid approach, the link cache size is 15, the ping interval is 5, and the pong size is 5. Two configurations of the hybrid approach are used in the comparison, the base line with all policies being *Random*, and the configuration in which the ping pong policy is *MFS* and all other policies are *Random*.

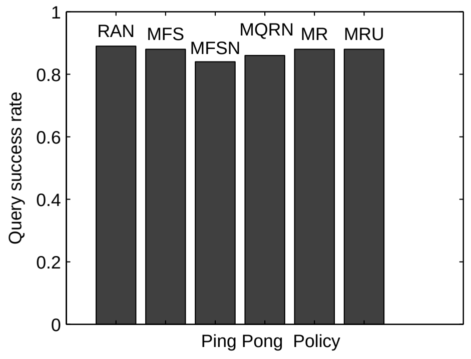
In the non-forwarding scheme, the pong size is the same as the hybrid approach. We simulated the base line non-forwarding with two settings of *LinkCacheSize* and *PingInterval*. The first setting is *LinkCacheSize* $= 15$ and *PingInterval* $= 5$. It is for the non-forwarding to



(a) Number of query messages.



(b) Number of dead query messages.



(c) Query success rate.

Figure 11. Varying ping pong policies (LC = 15); 1000 nodes.

maintain the same state as the hybrid approach. The second setting is *LinkCacheSize* = 200 and *PingInterval* = 12. This setup is for the non-forwarding approach to achieve a sufficiently high query success rate.

Figure 12 contrasts the performances of the forwarding, non-forwarding and the hybrid approach. The *X* and *Y* axises are the average query cost (the average number of query messages per successful query), and the query unsuccess rate respectively. The line for the forwarding approach is generated by varying the maximum number of query messages per query. The *non-forwarding (base line, same state)* and *non-forwarding (base line, larger state)* refer to the first and second settings of link cache sizes and ping intervals.

The forwarding approach has a fixed searching extent; the query unsuccess rate increases dramatically when the query cost is restricted. Both the hybrid approach and non-forwarding approach have smaller unsuccess rates than the forwarding approach at the same query cost due to query flexibility.

When the state maintenance overhead is the same (the same link cache size, ping interval and pong size), the base line hybrid approach (represented by the circle) can achieve a significantly higher query success rate than the base line non-forwarding approach (denoted by the square). In fact, the percentage of unsuccessful queries in the non-forwarding scheme is too large for practical usage. This performance difference is due to the 1-hop forwarding inherent in the hybrid approach. With the absence of query caches, the non-forwarding approach relies solely on nodes that are in the link cache of a query source and are alive. The searching scope of the hybrid approach includes more living nodes: those that are in the link caches of neighbors of a query source. It should be noted that the higher success rate of the hybrid approach is achieved at a query cost higher than the non-forwarding approach but lower than the forwarding approach.

The base line non-forwarding search can achieve the same query success rate as the base line hybrid search. This scenario is illustrated in the figure as the left triangle. The average query cost of the base line non-forwarding is even slightly smaller than the base line hybrid. However, the improvement in the query success rate and the query cost comes with a

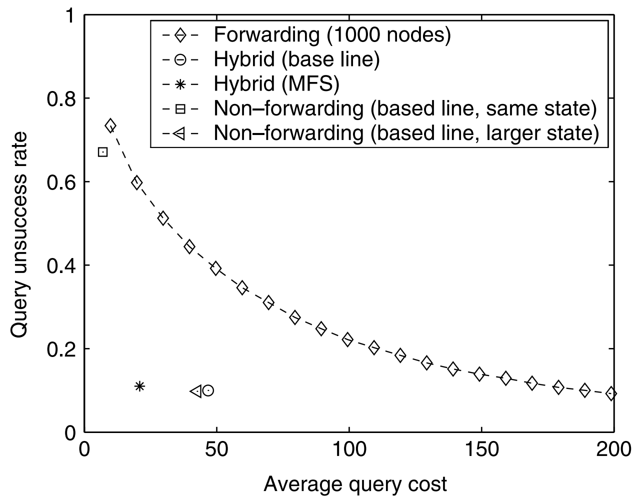


Figure 12. The query unsuccess rate in terms of the average query cost in three approaches.

dramatically higher state maintenance. Each node in this non-forwarding setting has to keep a link cache that is thirteen times as large as that in the base line hybrid. In addition, each node has to ping the link cache entries more than twice as frequently as a node in the base line hybrid. In summary, the experimental results demonstrate that the hybrid approach combines the advantages of both the forwarding and non-forwarding approaches.

6. Conclusions

In this paper, we propose a hybrid searching scheme in unstructured P2P networks. It is a combination of probing and guided 1-hop forwarding. Given a query, the querying source probes its neighbors and forwards the query to its neighbors. These neighbors probe their own neighbors on behalf of and under the guidance of the querying source as a result of query forwarding. When the query is satisfied, the querying source terminates its own probing and the probing performed by its neighbors. To integrate the probing and forwarding smoothly, we compute an *action queue* that consists of *ProbeOnly*, *ForwardOnly* and *Probe&Forward* actions sorted in the descending order of their gain/cost ratios. The querying source just takes actions in this queue at a constant rate or a variable rate that is adapted to the rareness of the sought data. We also propose two new policies for recruiting new neighbors, *Most Files Shared on Neighborhood (MFSN)* and *Most Query Results on Neighborhood (MQRN)*.

Compared to the forwarding-based scheme, hybrid searching is more flexible. It adapts the query processing to the popularity of sought files and does not waste resources when searching for popular files. Therefore hybrid searching has a higher query efficiency. Compared to the non-forwarding scheme, hybrid searching accomplishes a better query success rate without maintaining a larger state. To the best of our knowledge, this is the first work to combine the forwarding and non-forwarding schemes in unstructured P2P networks.

The hybrid search in this paper is applied to flat P2P overlays. When the P2P network is very large, this could lead to a scalability problem. One solution is to designate some peers as superpeers, each of which processes queries for other regular peers that connect to this superpeer. All superpeers form an unstructured P2P sub-overlay. Hybrid searching can be applied to this P2P sub-overlay. We will evaluate this approach in the future.

Acknowledgement

This work was supported in part by NSF grants ANI 0073736, CCR 0329741, CNS 0422762, CNS 0434533, EIA 0130806 and CNS 0531410.

References

[1] Li, X. and Wu, J., 2005, Searching techniques in peer-to-peer networks. In: J. Wu (Ed.) *Handbook of Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks* (Boca Raton, FL: CRC Press).
[2] Guess protocol specification v0.1, http://groups.yahoo.com/group/the_gdf/files/Proposals/GUESS/guess_o1.txt.
[3] Yang, B., Vinograd, P. and Garcia-Molina, H., 2004, Evaluating guess and non-forwarding peer-to-peer search. *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (IEEE ICDCS'04)*.

[4] Yang, B. and Garcia-Molina, H., 2002, Improving search in peer-to-peer networks. *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (IEEE ICDCS'02).*

[5] Lv, Q., Cao, P., Cohen, E., Li, K. and Shenker, S., 2002, Search and replication in unstructured peer-to-peer networks. *Proceedings of the 16th ACM International Conference on Supercomputing (ACM ICS'02).*

[6] Kalogeraki, V., Gunopulos, D. and Zeinalipour-yazti, D., 2002, A local search mechanism for peer-to-peer networks. *Proceedings of the ACM Conference on Information and Knowledge Management (ACM CIKM'02).*

[7] Tsoumakos, D. and Roussopoulos, N., 2003, Adaptive probabilistic search in peer-to-peer networks. *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03).*

[8] Yang, C., Li, X. and Wu, J., 2005, Dominating-set-based searching in peer-to-peer networks, *International Journal of High Performance Computing and Networking*, **4**(3), 205–210.

[9] Escalante, O., Perez, T., Solano, J. and Stojmenovic, I., 2005, RNG-based searching and broadcasting algorithms over internet graphs and peer-to-peer computing systems. *Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'05).*

[10] Zhu, Y., Yang, X. and Hu, Y., 2005, Making search efficient on gnutella-like p2p systems. *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS'05).*

[11] Kumar, A., Xu, J. and Zegura, E., 2005, Efficient and scalable query routing for unstructured peer-to-peer networks. *Proceedings of IEEE INFOCOM'05.*

[12] Wang, C., Xiao, L., Liu, Y. and Zheng, P., 2004, Distributed caching and adaptive search in multilayer P2P networks. *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04).*

[13] Liu, Y., Xiao, L., Liu, X., Ni, L.M. and Zhang, X., 2005, Location awareness in unstructured peer-to-peer systems, *IEEE Transactions on Parallel and Distributed Systems*, **16**(2), 163–174, Feb..

[14] Fei, T., Tao, S., Gao, L. and Guerin, R., 2006, How to select a good alternative path in large peer-to-peer systems. *Proceedings of IEEE INFOCOM'06.*

[15] Rodruigues, R., Gupta, A. and Liskov, B., 2003, One-hop lookups for peer-to-peer overlays. *Proceedings of the 11th Workshop on Hot Topics in Operating Systems (HotOS'03).*

[16] Tsoumakos, D. and Roussopoulos, N., 2003, A comparison of peer-to-peer search methods. *Proceedings of the International Workshop on the Web and Databases (WebDB'03).*

[17] Daswani, N. and Garcia-Molina, H., 2003, Pong cache poisoning in guess, *Technical report,* Stanford University.