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
{xli, jie}@cse.fau.edu

Abstract

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.

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 [7]. In this paper, we are concerned with unstructured P2Ps because they are 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. Most existing P2P searching techniques are based on *forwarding* [7]. 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.

Recently, *non-forwarding* schemes were proposed in [2] [12]. 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

*This work was supported in part by NSF grants CCR 9900646, CCR 0329741, ANI 0073736, and EIA 0130806.

on its own 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 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, including iterative deepening [11], local indices [11], k-walker random walk [8], modified random BFS [6], two-level k-walker random walk [5], directed BFS [11], intelligent search [6], routing indices based search [3], adaptive probabilistic search [9], and dominating set based search [13]. These schemes are different variations of flooding used in Gnutella [1]. They can be classified as deterministic or probabilistic [7].

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 information about its 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 processing of a query, 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 is not maintained.

The performance of GUESS is improved in [12], 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. For each policy type, many specific policies may be adopted. Five common policies, which include random (RAN), most recently used

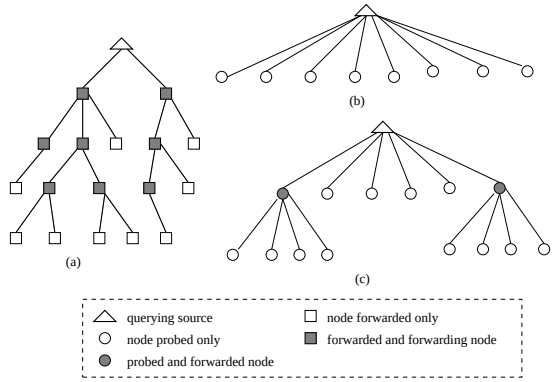

Figure 1. The three types of P2P searches. (a) forwarding based. (b) non-forwarding based. (c) hybrid.

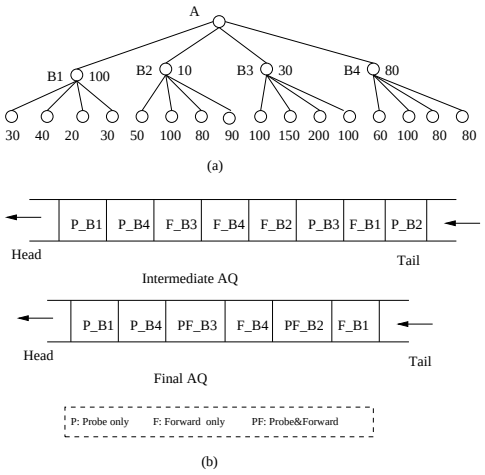

Figure 2. An example of action queue computation. (a) the querying source A and its 2-hop neighborhood, and the file distribution. (b) the computed action queue (the intermediate and final results).

(MRU), least recently used (LRU), most files shared (MFS), and most results (MR), are proposed for these policy types.

3 Outline of the hybrid search

Figure 1 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 1(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 1(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 1(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 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 action queues are shown in Figure 2(b). The final AQ(Action Queue) 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.

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 Algorithm 1 and Algorithm 2. Given a query q, the querying source s first computes the action queue 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 is any. When v receives F or PF messages, it probes its own neighbors on behalf of s with k_2 neighbors per 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 action queue is computed based on the gain/cost ratios of the actions if they are performed on the querying

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
```

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 random (RAN), most recently used (MRU), most files (MF), and most query results (MR). The action queue computation algorithm varies according to the chosen system policy.

If the system policy is random, the action queue 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 most recently used, the action queue 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 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 most files, the action queue 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. $NumF_v$ represents the gain of the action. It is the number of files on node v. 2 is the cost of this action, 2 overlay hops.

$$PGCR_v = \frac{NumF_v}{2}$$

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. $NumF_u$ 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.

$$FGCR_v = \frac{\sum_{u \in NB_v} NumF_u}{\gamma(1 + 2d_v/k_2 + d_v/k_2)}$$

If the system policy is most query results, the action queue computation is similar to that of most files. 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.

Algorithm 3 The action queue 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**

An example of action queue computation is shown in Figure 2 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 2(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 most files, $k_2 = 2$, 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

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

Then we sort these actions in the descending order of their gain/cost ratios and get the intermediate action queue as shown in Figure 2(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 2(b).

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 action queue 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.

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

Algorithm 4 The $adaptive\_integration\_search$ at the querying source s (called by Algorithm 1)

```
1:  Initialize α;
2:  while q is not satisfied AND AQ is not empty do
3:     remove the first α actions from AQ and store them
       in the array ACT_k;
4:     for i = 0 to α − 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 α arrives then
22:       update α 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 action queue computation. This subsection dicusses 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 action queue 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 most recently used, 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 most files or most query results, v chooses k_2 unprobed neighbors that have the top number of files or top number of query results for the most recent query.

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

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

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

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

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 is.

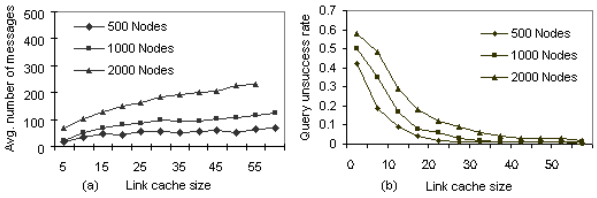

Figure 3. (a) The number of query messages per query. (b) The unsuccess rate in terms of the link cache size for the hybrid search.

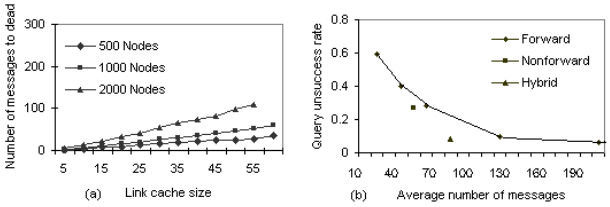

Figure 4. (a) The unsuccess rate in terms of the link cache size. (b) The query unsuccess rate in terms of the average number of messages per query of three approaches

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 cache. 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 PringProbe 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, random (RAN), most recently used (MRU), least recently used (LRU), most files shared on neighbor (MFS), most query results on neighbor (MR), most files shared in neighborhood (MFSN), and most query results in neighborhood (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 neighobr 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.

5 Experimental Results

The performance of the hybrid approach is evaluated experimentally against the forwarding-based scheme and non-forwarding searching. Only the base-line policy *RAN* is implemented because of the time limitation. The performance measures are the average query success rate and the average number of query messages per 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.

We created a network of $numNodes$ nodes. The overlay for the forwarding approach is random graph with an average node degree of 4. For the non-forwarding and hybrid approach, each node's link cache is seeded with *cacheSeedSize* neighbors. Then the neighbors are dynamically extended/updated based on the *PingProbePolicy*, *PingPongPolicy*, and *CacheReplacementPolicy*. Both the document replication distribution and the query distribution is zipfian distribution. As suggested in [10], 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 nodes in the system remains the same. We use the *random friend seeding policy* [4] 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 a fixed number of neighbors in the link cache at constant speed.

Figure 3 illustrates the impact of the different link cache sizes in networks of different scales for the hybrid approach. To isolate the effect of the link cache, we did not implement the query cache. As seen in Figure 3(a), the number of query messages per query increases as the link cache gets larger. The query unsuccess rate drops quickly as the link cache size increases as shown in Figure 3(b). When the link cache size is more than 30, the query unsuccess rate does not change much. Figure 4(a) 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 to 30.

In Figure 4(b), we compare the hybrid approach to the forwarding and non-forwarding approach using the query

unsuccess rate per average query cost (the number of query messages per query) in the network of 1000 nodes. The link cache sizes of the non-forwarding and hybrid approach are 100 and 20 respectively. The ping intervals are the same for both the non-forwarding and hybrid approaches. 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 similar (ping at the same speed and pong size is the same), the hybrid approach can achieve a higher success rate than the non-forwarding approach. This is due to the 1-hop forwarding inherent in the hybrid approach. The searching scope of the hybrid approach includes more living neighbors. 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. 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* which 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 more complex state. To the best of our knowledge, this is the first work to combine the forwarding and non-forwarding schemes.

In the future, we plan to do more experiments to evaluate different system policies and adaptive integration. 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 connects 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.

References

[1] The gnutella protocol specification v0.4. Clip2 distributed search solutions, http://www.clip2.com.

[2] Guess protocol specification v0.1. http://groups.yahoo.com /group /the_gdf /files /Proposals /GUESS /guess_o1.txt.

[3] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing (IEEE ICDCS'02)*, 2002.

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

[5] I. Jawhar and J. Wu. A two-level random walk search protocol for peer-to-peer networks. In *Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, 2004.

[6] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-yazti. A local search mechanism for peer-to-peer networks. In *Proceedings of the 11th ACM Conference on Information and Knowledge Management (ACM CIKM'02)*, 2002.

[7] X. Li and J. Wu. *Searching techniques in peer-to-peer networks*. in Handbook of Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, Edited by J. Wu, CRC Press, 2005.

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

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

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

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

[12] B. Yang, P. Vinograd, and H. Garcia-Molina. Evaluating guess and non-forwarding peer-to-peer search. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (IEEE ICDCS'04)*, 2004.

[13] C. Yang and J. Wu. A dominating-set-based routing in peer-to-peer networks. In *Proceedings of the 2nd International Workshop on Grid and Cooperative Computing Workshop (GCC'03)*, 2003.