IEICE TRANS. COMMUN., VOL.E89–B, NO.9 SEPTEMBER 2006

PAPER Special Section on Networking Technologies for Overlay Networks

AntSearch: An Ant Search Algorithm in Unstructured Peer-to-Peer Networks

Kai-Hsiang YANG[†a)], ***Member***, **Chi-Jen WU**[†], ***and*** **Jan-Ming HO**[†], ***Nonmembers***

SUMMARY The most prevalent peer-to-peer (P2P) application till today is file sharing, and unstructured P2P networks can support inherent heterogeneity of peers, are highly resilient to peers' failures, and incur low overhead at peer arrivals and departures. Dynamic querying (DQ) is a new flooding technique which could estimate a proper time-to-live (TTL) value for a query flooding by estimating the popularity of the searched files, and retrieve sufficient results under controlled flooding range for reducing network traffic. Recent researches show that a large amount of peers in the P2P file sharing system are the free-riders, and queries are seldom hit by those peers. The free-riding problem causes a large amount of redundant messages in the DQ-like search algorithm. In this paper, we proposed a new search algorithm, called "AntSearch," to solve the problem. In AntSearch, each peer maintains its hit rate of previous queries, and records a list of pheromone values of its immediate neighbors. Based on the pheromone values, a query is only flooded to those peers which are not likely to be the free-riders. Our simulation results show that, compared with DQ and its enhanced algorithm DQ+, the AntSearch algorithm averagely reduces 50% network traffic at almost the same search latency as DQ+, while retrieving sufficient results for a query with a given required number of results.

key words: *peer-to-peer network, dynamic querying, flooding, free-riding*

1. Introduction

Peer-to-peer (P2P) networks such as Gnutella, KaZaA, and BitTorrent have emerged as a new Internet computing paradigm over the past few years. The most prevalent P2P application till today is file sharing. In contrast to structured P2P networks, search in unstructured P2P networks is considerably more challenging because of the lack of global routing and directory service. In spite of this apparent limit, unstructured P2P networks have several desirable properties: (1) they support inherent heterogeneity of peers; (2) they are highly resilient to peers' failures, and (3) they incur low overhead at peer arrivals and departures. Most importantly, they are simple to implement and result in virtually no overhead in topology maintenance. Consequently, many real-world large-scale P2P networks are unstructured.

In a Gnutella P2P network, a blind flooding algorithm is used to search results for a query under a time-to-live (TTL) constraint. The biggest problem of the blind flooding algorithm is that a single query may cause a large amount of network traffic, and the second problem is that, the number of search results can not be guaranteed. A good search algorithm should be able to retrieve sufficient (small or no overshooting) results for a query with a given required number of results at low network traffic cost. For this purpose, a new controlled flooding technique, dynamic querying (DQ) [1], is proposed for these requirements. It works as follows. (1) Probe phase: a requester peer (a peer that generates a query) first floods a query towards a few neighbors with a small TTL value for estimating the popularity of the searched items. Then (2) an iterative process takes place. During each of iterations, the requester peer computes the number of peers to be contacted for obtaining the desired number of results; then it chooses a neighbor peer, calculates a TTL for a query flooding to that neighbor, and propagates a query with that TTL to the neighbor peer. This iterative process stops when the desired number of results is returned, or all neighbor peers have been visited. Intuitively, this flooding algorithm is dynamic because the requester peer estimates the item's popularity to adjust a TTL value for each flooding, so that sufficient results can be retrieved at lower network traffic overhead than a blind flooding algorithm.

Jiang et al. [4] evaluated and analyzed the DQ technique and proposed an enhanced DQ technique, DQ+, which can further reduce network traffic cost and shorten search latency. To avoid network traffic cost, the DQ+ technique uses a confidence interval method to provide a safety margin on the estimate of the popularity of the searched item. To achieve the lower search latency, the DQ+ technique uses the greedy strategy in each of iterations where the requester peer expects to find sufficient results from a chosen neighbor. Compared with the DQ technique, a query packet is only flooded to a small amount of the required number of peers, and thus the DQ+ technique is excellent in the performance of search latency. Basically these two algorithms are still based on a flooding technique.

Unfortunately, there is a serious problem, called the free-riding problem, for a flooding technique. Current research papers [2], [3] show that a large amount of peers in P2P file sharing systems are free-riders, which is defined as the peers sharing less than 100 files (about 96% in [2], and 75% in [3]), and queries are seldom hit at these peers. Thus, a query flooding causes a large amount of network traffic for sending queries to those free-riders. For example, the part (a) of Fig. 1 depicts an unstructured P2P network which is formed by eight peers, and each peer has three immediate neighbors. Suppose the peer A is the requester peer, and the target files are located in peers B, E, and H. It is easily to observe that each peer excluding peer A receives three query packets from its immediate neighbors. In this paper, we de-

Manuscript received December 26, 2005.
Manuscript revised April 11, 2006.

[†]The authors are with Institute of Information Science, Academia Sinica, Taiwan, ROC.
a) E-mail: khyang@iis.sinica.edu.tw

DOI: 10.1093/ietcom/e89–b.9.2300

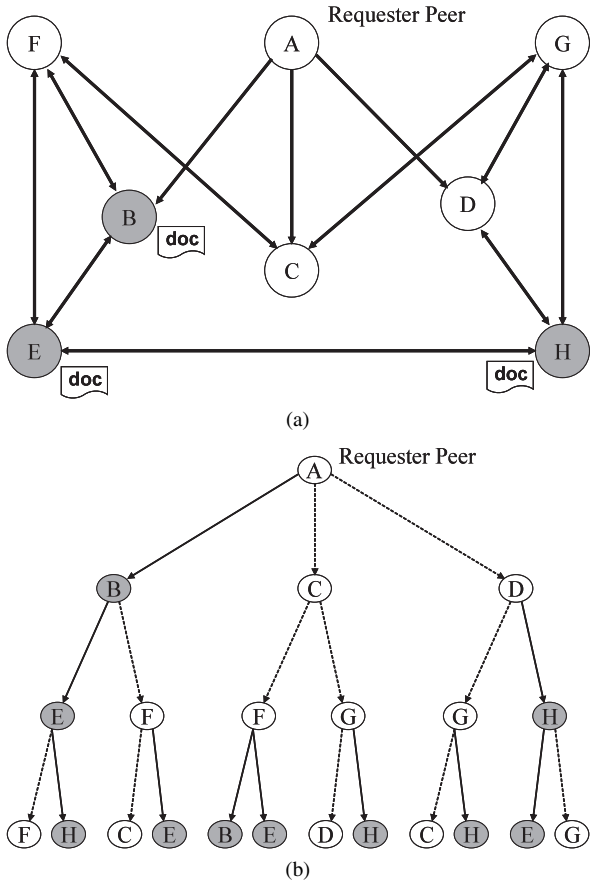

Fig. 1 The excessive traffic overhead in a flooding search algorithm. The part (a) depicts an unstructured P2P network topology, and peers B, E, and H contain the target files. The part (b) shows the total flooding path when peer A floods a query with TTL = 2. A solid line represents a hit message, and a dotted line represents a redundant message.

fine a query packet/message is redundant when it is sent to a peer and does not hit in that peer. In the part (b) of Fig. 1, the total 21 packets generated in a query flooding consist of 10 hit messages (solid lines) and 11 redundant messages (dotted lines). Hence, a query flooding will cause too much redundant network traffic.

In this paper, we focus on the free-riding problem in the DQ-like search algorithm, and propose a new search algorithm, called "AntSearch," to reduce the redundant messages during a query flooding. In AntSearch, each peer maintains a pheromone value to represent its hit rate of previous received queries, and also records a list of pheromone values for its immediate neighbors. Based on these pheromone values, the AntSearch algorithm can flood a query only to those peers which are not likely to be free-riders. The main idea in the AntSearch algorithm is using pheromone values to identify the free-riders, prevent sending messages to those peer in order to reduce the amount of redundant messages.

We have conducted extensive experiments to evaluate the number of searched files, network traffic cost and search latency in AntSearch. Compared with the DQ and DQ+ search algorithms, our simulation results show that the AntSearch algorithm averagely reduces 50% network traffic at almost the same search latency as DQ+, while retrieving sufficient results for a query with a given required number of results.

The remainder of this paper is organized as follows. Section 2 briefly reviews current related works in unstructured P2P networks. Section 3 introduces the system architecture of AntSearch, and its index structures and search algorithm. Section 4 discusses several important issues of AntSearch. The experimental methodology and results are presented in Sect. 5. Finally, we summarize our results and represent our conclusions in Sect. 6.

2. Related Work

In this section, we review previous search algorithms in unstructured P2P networks and describe the free-riding problem in a P2P system. As mentioned above, a flooding search algorithm is blind and expensive, since the network does not provide any clues to facilitate a search. Hence, it is very crucial to reduce network traffic, shorten search latency, and retrieve sufficient results for a query. The Gnutella developer community proposed the DQ technique to guarantee that sufficient results can be retrieved and a research result in [5] indicated that DQ technique could predict a proper TTL value for a query flooding in order to reduce network traffic load. Jiang et al. [4] evaluated and analyzed the DQ technique and proposed an enhanced DQ technique, DQ+. However, the design of DQ and DQ+ techniques are still based on a flooding search algorithm without considering the free-riding problem in the P2P network.

Besides the DQ-like technique, several solutions [6]–[8] have also been proposed to reduce network traffic load during a query flooding. Yang et al. [7] proposed a technique in which each peer only forwards a query to a subset of its neighbors according to statistics of previous query contents. This solution reduces network traffic; however the number of retrieved results may not be able to satisfy the query. Another limitation of this solution is that, each peer has to spend large space for storing the statistics about the query contents, and periodically maintain the statistics. Another type of search algorithm is a well-known random walk technique, which forwards a query to a neighbor at each step until a file is found. Yatin et al. [6] proposed an algorithm called GIA, based on the random walk technique. Each peer maintains an index of files stored in its neighbors and floods a query to those high capacity peers. In general, random walk based algorithm can reduce network traffic and enhance the system scalability; however, it usually results in longer search latency, and the number of retrieved results varies to a great extent for different underlying network topologies [9].

Recently, several research papers [12], [13] study the user behavior in P2P systems, and discover the free-riding problem is very serious in a flooding-based algorithm. Feldman et al. [12] present an economic model of user behavior in P2P systems, explore the effect of free-riders, and propose

several research problems for the free-riding phenomenon. Ramaswamy et al. [13] introduce a concept of utility function to measure the usefulness of every user to the system, and proposed a free-rider control scheme. They focus on modeling the free-riding phenomenon and studying the user behavior in P2P systems. However, our proposed AntSearch algorithm is a feasible solution for solving the free-riding problem in an unstructured P2P network.

3. Design of AntSearch

In this section we first give an overview of AntSearch algorithm, and then present the data structures in each peer and detail the search algorithm.

3.1 Overview

The AntSearch algorithm is designed for solving the free-riding problem while searching in unstructured P2P networks. Like the DQ search algorithm [1], AntSearch algorithm comprises two search phases: (1) a probe phase and (2) a flooding phase.

(1) Probe phase: when a requester peer starts to process a query, it first has to flood probe queries to a few neighbors with a small TTL (in general, flooding to three neighbors with TTL=2) for estimating the popularity of target files. When the small-area flooding ends, the requester peer obtains the statistics information about the searched files. By this statistics information, the requester peer can predict how many results could be retrieved when each step only floods the query to k% (k=10, 20..., 100) of immediate neighbors. All these information will be stored into a data structure which is called the "probe table."

(2) Flooding phase: When the probe table is generated, the requester peer has to decide two parameters about a query flooding, the first one is the k value, and the other is the TTL value. The k value represents how much percentage of neighbors should be chosen to flood a query, and the TTL value is an upper bound of flooding hops. According to the probe table, a requester peer can estimate the search cost (including search latency and flooding messages) for each k value, and chooses a suitable k value for the following query flooding. When a k value is assigned, an iterative search process takes place. During each of iterations, (1) the requester peer calculates how many peers should be further contacted, and computes a suitable TTL for a neighbor. (2) The requester peer then propagates the query packet towards a neighbor, and all the following peers only forwards the query to the k% of neighbors with higher pheromone values. This iterative process stops when the desired number of results is returned, or all neighbors have been visited.

The main difference between the AntSearch algorithm and the dynamic querying search algorithm [1], DQ+ [4], is that the AntSearch algorithm improves the search efficiency of a flooding by reducing the number of messages sent by a peer and the number of peers that are queried. According to the table of pheromone values, a peer only propagates a query to top k% of its neighbors with higher pheromone values. The pheromone table is used to help a peer identify the neighbors which may be free-riders. Figure 2 illustrates search efficiency in the AntSearch algorithm. Compared with the part (b) in Fig. 1, each peer only floods the query to fewer neighbors with higher pheromone values. Intuitively, the pheromone table is a data structure to hint the direction where a target file is located.

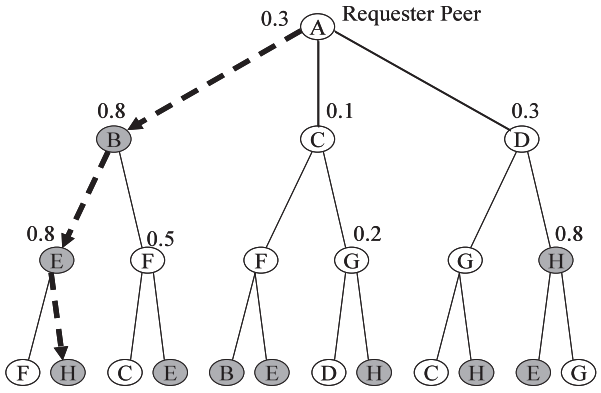

Fig. 2 An example of AntSearch, the number at the peer p_i side is presented the pheromone value of the peer p_i.

3.2 Pheromone Table

The objective of storing a pheromone table in each peer is to record the hit rate of previous queries in each immediate neighbor, so that a peer can choose a neighbor with higher pheromone value to forward a query. A neighbor's pheromone value primarily measures the probability it is a free-rider. Each peer in the system maintains two values: the first one is the number of hit queries N_h, and the other is the number of total processed queries N_q. These two values are permanently stored in a peer when it joins the system in the first time. For a peer q, suppose that the d_q is the degree of peer q (which means peer q has d_q number of immediate neighbors), the pheromone value of peer q, (pv_q), can be computed as follows:

$$pv_q = \frac{N_h}{N_q} \times \alpha + \frac{\sum_{i=1}^{d_q} pv_i}{d_q}(1-\alpha), \quad (1)$$

where the first part of the formula, N_h/N_q, is the hit rate of previous queries in peer q, and the second part is the average pheromone values in immediate neighbors of peer q, which represents the effect of neighbors. The value α (between 0 and 1) is a parameter to adjust the weights between the hit rate of peer q and the average pheromone value of the neighbors of peer q. Another benefit of using the value α is to prevent a pheromone value from being zero. A peer still has certain probability to be searched when it first joins into the network. If a peer q is a free-rider, the ratio of N_h/N_q will be very small. On the other hand, if the peer q is a peer sharing a lot of popular files, the ratio of N_h/N_q will increase

rapidly. Note that the N_h value will be added by one when a query is hit. These two values are continually updated both in probe phase and flooding phase. The pheromone table stored in a peer is only updated under the following two situations. (1) When a peer joins into the system, it first collects the pheromone values of its immediate neighbors. The pheromone value is sent within PING and PONG messages in the Gnutella protocol. (2) When a peer receives a query, it then updates each record of the pheromone table. When a neighbor leaves the network, a peer immediately removes the pheromone value of the neighbor from its pheromone table. No other action is required in the AntSearch system, and it is clear to observe that the maintenance cost for a pheromone table is very limited.

3.3 AntSearch Algorithm

The AntSearch algorithm is a controlled flooding technique to search results for a query with a specified required number of results, denoted by N. The search process comprises two phases: (1) the probe phase and (2) the flooding phase.

(1) Probe phase. When a requester peer produces a query with a required number of results, N, it first floods a "probe query" to a few neighbors with a small TTL (in general, flooding to three neighbors with TTL=2). After the flooding of the probe query finishes, the requester peer collects the statistics about the numbers of searched files and total searched peers. A probe table is then generated to summarize the numbers of the searched files and the searched peers when only flooding different k% of neighbors in each step. Table 1 gives an example of a probe table, which consists of three columns: the k value, the number of searched files n_k, and the estimated number of searched peers, h_k (also called the search horizon in this paper).

In this case when k is 10%, each peer only forwards a query to 10% of its neighbors with higher pheromone values, and the requester peer can search 1 file while the flooding averagely visits to 15.84 number of peers. In this paper, we assume the degree d of a neighbor can be known, and the average degree of network is D which can be estimated. For each k, the search horizon, h_k, is calculated by the following formula, where TTL = 2 in our experiments.

$$h_k = \sum_{i=1}^{3}(d_i k)\sum_{j=0}^{TTL-1}(Dk-1)^j = \sum_{i=1}^{3}(d_i k)(Dk)^{2-1} = \sum_{i=1}^{3} d_i Dk^2 \quad (2)$$

In order to gather the number of searched files, n_k, for each k during a single flooding to small area, we designed a probe flooding mechanism (PFM) to obtain all the needed information for a probe table. In the PFM, each query packet is sent with a mark to identify which k it belongs to during the search steps. Figure 3 shows an example of the PFM. Step 1 represents that a requester peer, A, forwards a query to its neighbor, B, which belongs to the top 10% neighbors

Table 1 The probe table.

k	n_k	h_k
10%	1	15.84
20%	3	63.36
30%	5	142.56
40%	8	253.44
50%	12	396.00
60%	12	570.24
70%	13	776.16
80%	15	1013.76
90%	16	1283.04
100%	17	1584.00

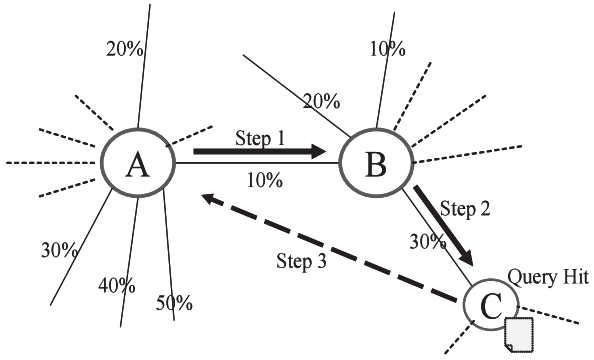

Fig. 3 An example of a probe phase in the AntSearch system. Peer A is the requester peer, and peer B belongs to the top 10% peers in the pheronome table of peer A. Peer C belongs to the top 30% peers in the pheronome table of peer B, and it contains a target file.

with higher pheromone values. In Step 2, the peer B forwards the query to its neighbor, C, which belongs to the top 30% neighbors with higher pheromone values. Before forwarding the query to peer C, peer B has to rewrite the new k value into the mark in the query from 10% to 30%. In Step 3, when a query is hit, the peer C returns a hit message with the mark k = 30% to the requester peer A. Thus, the requester peer can calculate the number of searched files for each different k after the flooding of a probe query finishes, and then generates a probe table.

(2) Flooding phase: Since a probe table is already generated, the first step in the flooding phase is to choose a proper k and compute a TTL value for the query. Figure 4 illustrates the pseudo code of the AntSearch algorithm. For each k, we can easily calculate how many peers the flooding has to further search for retrieving a required number of results, N. Suppose H_k denote the further search horizon for a given k, and it could be estimated by $h_k(N - n_k)/n_k$. When the H_k is computed, we can estimate a proper TTL for a flooding with a given k to reach the search horizon by the following formula (2) in [4].

$$TTL_K \approx \log_{(Dk-1)} \frac{H_k(Dk-2)}{dk-1} \quad (3)$$

Here we start to introduce the method for choosing a proper k value. Figure 5 shows the pseudo code of the function "Choosing_K." As shown in Fig. 5, a requester peer has

```
1) AntSearch ( desired number of result N )
2) Begin
3) Probe_Table= Probe()
4) If returned_Results >= N then return Results
5) Else
6)    K = Choosing_K(Probe_Table, MAX_TTL)
7)    While returned_Results <= N
8)       Neighbor = randomly choosing a unvisited neighbor
9)       d = degree(Neighbor)
10)      TTL = Calculating _TTL(K, d)
11)      Forwarding query to the Neighbor with K and TTL
12)   End while
13) End if
14) return returned_Results
15) End begin
```

Fig. 4 Pseudo code of the AntSearch algorithm.

```
1) Choosing_K ( Probe_Table, MAX_TTL)
2) Begin
3)    For (k=0.1 to 1.0 )
4)       H_k = h_k(N − n_k) / n_k
5)       TTL = log_(Dk−1) H_k(Dk − 2) / (dk − 1)
6)       if TTL <= MAX_TTL then break
7)    End for
8)    return k
9) End begin
```

Fig. 5 The pseudo code of choosing k rate.

to calculate the required TTL value for each k, and finds a minimum k with a TTL less than or equal to a TTL threshold, the *MAX_TTL*. (The *MAX_TTL* is generally set to 4).

After a proper k is chosen, the requester peer starts an iterative process to search the results (from line 7 to line 12 in Fig. 4). During each of iteration, the requester peer randomly chooses a neighbor and calculates a proper TTL for the neighbor. It then sends out a query with the chosen k and the calculated TTL to the neighbor (at line 10 in Fig. 4, and the function **Calculating_TTL** is implemented by the formula 3 to calculate a TTL value for a given k value). The iteration continues until the required number of files is obtained or all neighbors are visited.

There is a tradeoff between choosing a larger k value and choosing a smaller one. A flooding with a larger k value results in more query messages. For example, when k = 100%, all the neighbors will be searched in each step of a flooding. Hence, it will cause too much network traffic. On the other hand, choosing a smaller k value will result in a larger TTL value to be calculated to reach the search horizon, so the flooding will take longer to retrieve the required number of files.

Recall that in the original DQ and DQ+ techniques, a query packet is propagated to all the peers that can be reached within the TTL constraint. They do not consider the free-riding problem in a real P2P file sharing system. The AntSearch algorithm uses the pheromone table to prevent flooding a query to a free-rider, in order to reduce redundant network traffic.

4. Discussion

In this section, we discuss several important issues in terms of scalability and load balancing, convergence property of pheromone table, and file diversity of the AntSearch algorithm.

4.1 Scalability and Load Balancing

It is well known that Gnutella systems are not scalable due to their widespread use of flooding. To solve this problem, Jiang et al. [4] proposed a Dynamic Query algorithm to averagely reduce 50% network traffic. Hence, the DQ-like systems are more scalable than the Gnutella systems. In this paper, our AntSearch algorithm averagely reduces 50% network traffic in the DQ-like systems. Therefore, the AntSearch system is more scalable than the DQ-like systems.

To consider the load balancing issue, the non free-rider nodes may suffer from heavier network traffic load than the free-rider nodes do in our AntSearch network. This is because our AntSearch algorithm is designed to reduce the flooding messages which will be sent to free-rider nodes. However, in a flooding-based network, a query will be flooded to all nodes in a flooding region with a TTL constraint. Hence, load in a non free-rider node in the AntSearch network is equivalent to load in "anyone" node in a flooding-based P2P network.

Many research papers have addressed the load imbalance problem in unstructured P2P systems. Recent work [9], [15], [16] considers applying a static replication mechanism on the Gnutella network. The authors show that replicating objects proportionally to their population will achieve optimal load balance, and shorten average search latency. In [17], the authors proposed a distributed caching protocol for Gnutella-like system, which distributes index cache among nodes and divides the searching space into multiple layers. This protocol can significantly balance network traffic load for a query, and we believe that our AntSearch algorithm can collaborate well with these algorithms.

4.2 Convergence Property of Pheromone Table

In this section, we discuss about the convergence property of pheromone table in the AntSearch algorithm, and also show that the pheromone tables only need to locally converge. Recall that in the design of the AntSearch algorithm, each node only needs to know which of its immediate neighbors are likely to be the free-rider nodes before it forwards a query message, and does not send any query message to those neighbors. Indeed, when the pheromone tables only locally

converge, the flooding paths of the same queries issued at different time are probably different, but the AntSearch algorithm still can find efficient results for a query. It is the reason why our AntSearch system can work well when each pheromone table converges locally.

Moreover, another reason to our design is that nodes in the P2P systems are extremely transient, so that it is impossible to reach a globally convergence state. Some analyses [3] for the Gnutella and Napster systems indicate that the average online time of a node is around 60 minutes. For a large P2P system of 100,000 nodes, this implies a high churn rate of over 1600 nodes joining and leaving the system per minute. This high churn rate makes the pheromone tables difficult to globally converge. Our AntSearch algorithm, therefore, is very suitable and easy to implement for a real P2P system.

4.3 File Diversity

In this section, we discuss and show that the proposed AntSearch algorithm has the same ability to search the non-popular files as a flooding-based algorithm does.

First the search region in the AntSearch algorithm is limited and bounded by that in a flooding-based algorithm under the same TTL constraint. If a file is needed by minorities or it is created recently, the AntSearch algorithm will realize this during the probe phase of the search algorithm. Recall that the probe phase is to flood probe queries to a small region (usually, three neighbors with TTL=2) for estimating the popularity of target files. After the probe phase finishes, the number of searched files, n_k, in a probe table will be very small, so that the AntSearch algorithm selects a larger k value to find sufficient results. When k is chosen to a maximal value (1.0), all the neighbors are flooded for this query, and the search region is equivalent to that in a flooding-based algorithm.

Unfortunately, even though using a flooding-based algorithm to search, paper [14] tells us that some queries still can not return results. They measured the traffic characteristics of the Gnutella network from multiple vantage points in the Planet Lab, and showed that average 18% queries can not find results, and among two-third of which, there are results available in the network. The result implies that our AntSearch algorithm suffers from the same problem in a flooding-based algorithm. The paper [14] had proposed a hybrid search algorithm to increase search region for solving this problem, and we believe that our AntSearch algorithm can collaborate well with their solutions.

5. Performance Evaluation

In this section, we use three metrics to measure the performances of the AntSearch, DQ and DQ+ search algorithms. The simulation model is first described and then our simulation results are presented. Our simulator is based on that used in [4], which runs on a real Gnutella network topology on February 2, 2005 [10], and simulates 160,000 peers in this network. The average number of neighbors per peer is close to 24, and more detail information is provided in the technical report [11]. In the probe phase of the AntSearch algorithm, a query is propagated to random three neighbors with TTL=2, and in our experiments the default maximum TTL value is 4 seconds, and the timeout is set to 2.4 times TTL seconds. All the experimental setting is the same as that in [4].

During each simulation run, 1,000 different objects are located over 160,000 peers, and each object has 1,600 replicas in the P2P network. The placement policy of replicas follows the 80/20 distribution [2] to simulate a large number of free-riders existing in the P2P file-sharing system. Average 20% peers contain the 80% replicas, and the other 20% replicas are randomly located in the rest 80% peers. All the queries are uniformly distributed to the network (randomly choosing a peer without the searched file as the source peer to issue a query), and each query aims to retrieve 100 results (N=100). In formula (1) of pheromone value, the parameter α is set to 0.7. According to our extensive experiments, the parameter α does not significantly affect the performance of the AntSearch algorithm when it is set between 0.3 and 0.8.

The evaluation metrics used in our experiments include the followings. (i) Number of searched files: for a query with a required number of results, N, a good search algorithm should retrieve the number of results over but close to N. (ii) Per result cost: we define the per result cost as the total amount of query messages divided by the number of searched results. This metric measures how many average query messages generated to gain a result. (iii) Search latency: the search latency is defined as total time for the query process.

Figure 6 shows the number of searched results in the Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and the AntSearch (DQ-Ants) algorithms at 100 simulation runs. It is clearly to observe that, the number of searched results in DQ is less than the required number of results (N=100), because the DQ search algorithm

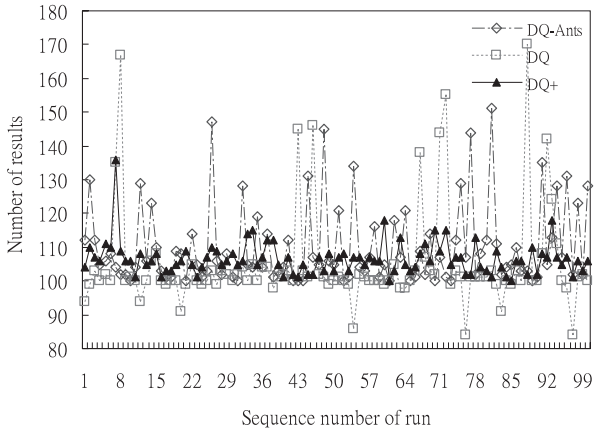

Fig. 6 The performance comparison of Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants) algorithms in the number of searched results.

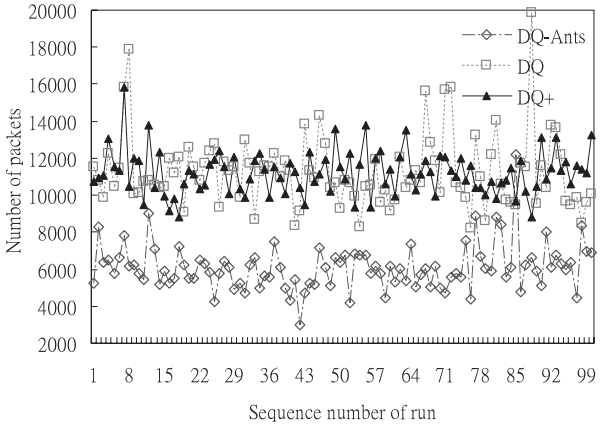

Fig. 7 The performance comparison of Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants) algorithms in the number of query messages.

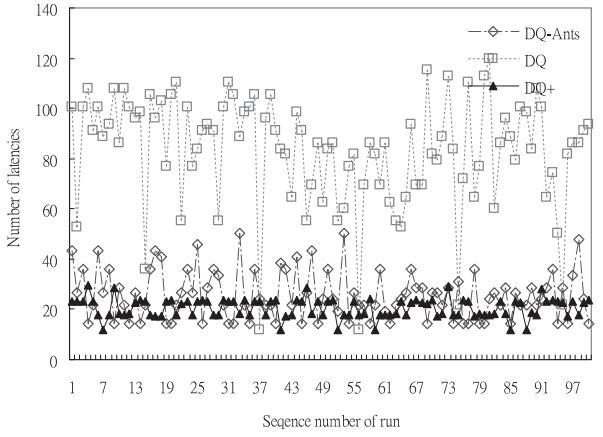

Fig. 8 The performance comparison of Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants) algorithms in search latency.

is designed to use a very conservative approach to flooding queries. On the other hand, the DQ+ and AntSearch algorithms always can retrieve the desired number of results. The most difference between them is that, the AntSearch sometimes retrieves a larger number of results than that in DQ+, and this overshooting problem is caused by the misestimate of search horizon, which means the actual number of searched peers is larger than the estimated one.

Figure 7 depicts the total number of query messages generated in the three algorithms at 100 simulation runs. First, we can observe that the total number of query messages in the AntSearch algorithm is much smaller than those in the DQ and DQ+ algorithms. The total number of query messages in the AntSearch algorithm is around 6,000, while those in the DQ and DQ+ algorithms are both approximately 12,000. In other words, the AntSearch algorithm average reduces 50% of query messages during a query. Recall that a query is propagated to all neighbors in the DQ and DQ+ algorithms, but it is only propagated to top $k\%$ of neighbors with higher pheromone values.

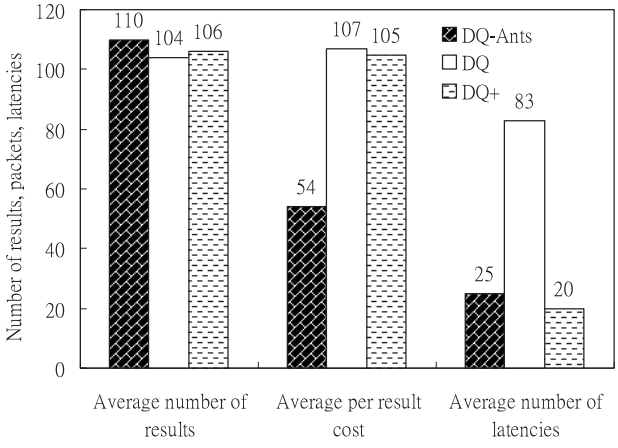

Fig. 9 The overall performance comparison of Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants) algorithms.

Figure 8 shows search latency in the three search algorithms at 100 simulation runs. Search latency is an important factor for the P2P file sharing system. We can observe that search latency in the DQ algorithm is longest, approximately 100 seconds at each simulation run. The DQ+ algorithm has the shortest search latency due to a greedy strategy applied in its iterative process. During each of iteration, the requester peer estimates to retrieve sufficient results from a selected neighbor. Besides, search latency in the AntSearch algorithm is a little higher but very close to that in the DQ+ algorithm, because a query is only propagated to top $k\%$ of neighbors with higher pheromone values, and in some cases the query needs more iterations to gain enough results.

The overall performance comparison of the three search algorithms are listed in Fig. 9. Averagely speaking, the per result cost (the number of query messages generated by a result) is about 105 in the DQ+, 107 in the DQ, and 54 in the AntSearch algorithm. It is clearly that the AntSearch algorithm did reduce approximately 50% network traffic for a query, and search latency in it is a little longer than that in the DQ+ algorithm by 5 seconds.

Figure 10 illustrates the performance metrics of the AntSearch, and DQ+ algorithms under different number of replicas. We can draw several conclusions by this result. First, the AntSearch algorithm always has smaller per result cost than that in the DQ+ algorithm under all different number of replicas. Second, search latency in the AntSearch algorithm is very similar to that in the DQ+ algorithm. Both of them produce short search latency. Third, for the number of searched results, the AntSearch algorithm can retrieve more results than the DQ+ algorithm does, especially when more replicas exist in the P2P system. According to these performance metrics, it is clearly that the AntSearch algorithm can reduce a large amount of network traffic at an acceptable cost of search latency, while receiving sufficient results for a query.

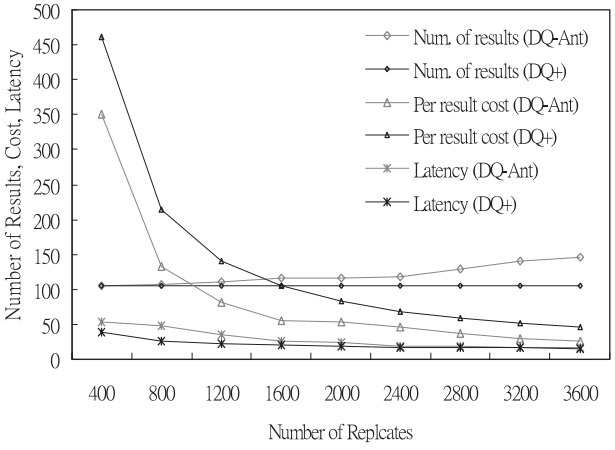

Fig. 10 The performance comparison of the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants) algorithms under different number of replicas.

6. Conclusions

This paper was motivated by the need of a search algorithm for an unstructured P2P system that can provide better search performance in terms of network traffic cost and search latency. The main contribution of our work is that we propose a search algorithm, called "AntSearch," which can greatly reduce network traffic in a query by only sending queries to those peers which are not likely to be free-riders. By allowing for a small space cost (the pheromone values in each peer), our simulation results show that AntSearch substantially reduces network traffic which is caused by sending queries to the free-riders, and completes a query at almost the same search latency as DQ+ search algorithm. Most importantly, AntSearch is scalable, simple and easy to implement into a real system.

Acknowledgments

The authors would like to thank Mr. Hongbo Jiang for the support of simulator used in his work [4].

References

[1] A. Fisk, "Gnutella dynamic query protocol v0.1," May 2003. http://www9.limewire.com/developr/dynamic_q-uery.html
[2] E. Adar and B.A. Huberman, "Free riding on Gnutella," Technical Report, Xerox PARC, 10 Aug. 2000.
[3] S. Saroiu, P.K. Gummadi, and S.D. Gribble, "A measurement study of peer-to-peer file sharing systems," Proc. Multimedia Computing and Networking (MMCN), pp.156–170, Jan. 2002.
[4] H. Jiang and S. Jin, "Exploiting dynamic querying like flooding techniques in unstructured peer-to-peer networks," Proc. IEEE Internet Conference on Network Protocol (ICNP), pp.122–131, Oct. 2005.
[5] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern P2P file-sharing systems," Proc. Internet Measurement Conference (IMC), pp.49–62, Oct. 2005.
[6] Y. Chawathe, S. Ratnasamy, L. Breslau, and S. Shenker, "Making Gnutella-like P2P systems scalable," Proc. ACM SIGCOMM, pp.407–418, 2003.
[7] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," Proc. 22nd International Conference on Distributed Computing Systems (ICDCS), pp.5–14, 2002.
[8] V. Cholvi, P.A. Felber, and E.W. Biersack, "Efficient search in unstructured peer-to-peer networks," European Trans. Telecommun., vol.15, no.6, pp.535–548, 2004.
[9] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," Proc. 16th International Conference on Supercomputing (ICS), pp.84–95, 2002.
[10] D. Shakkottai and R. Rejaie, "Characterizing the two-tier Gnutella topology," Proc. ACM SIGMETRICS (Poster), pp.402–403, June 2005.
[11] D. Shakkottai and R. Rejaie, "Characterizing today's Gnutella topology," Technique Report, CIS-TR-04-02, CIS, University of Oregon, Nov. 2004.
[12] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and whitewashing in peer-to-peer systems," Proc. ACM SIGCOMM'04 Workshop on Practice and Theory of Incentives in Networked Systems (PINS), pp.228–236, Aug. 2004.
[13] L. Ramaswamy and L. Liu, "Free riding: A new challenge to peer-to-peer file sharing systems," Peer-to-Peer Computing Track, Hawaii International Conference on System Sciences, HICSS-2003, pp.220–222, Jan. 2003.
[14] B. Loo, R. Huebsch, I. Stoica, and J. Hellerstein, "The case for a hybrid P2P search infrastructure," Proc. 3nd International workshop on Peer-to-Peer Systems, (IPTPS'04), pp.141–150, Feb. 2005.
[15] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," Proc. ACM SIGCOMM Conference, pp.177–190, 2002.
[16] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher, "Adaptive replication in peer-to-peer systems," Proc. 24th IEEE International Conference on Distributed Computing Systems, pp.360–369, 2004.
[17] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed caching and adaptive search in multilayer P2P networks," Proc. 24th IEEE international conference on distributed computing systems, pp.219–226, 2004.

Kai-Hsiang Yang received a B.A. degree in Department of Mathematics from National Taiwan University and his Ph.D. degree in Department of Computer Science and Information Engineering from National Taiwan University. He is currently a Postdoctoral Fellow in the Computer Systems and Communication Lab, Institute of Information Science, Academia Sinica. His research interests include Web mining, Peer-to-Peer computing, search engine technique, network protocols and architecture, network security, and information retrieval.

Chi-Jen Wu received the M.S. degrees in Communication Engineering from National Chung Cheng University, Taiwan in 2004. He is currently a research assistant in the Institute of Information Science, Academia Sinica. His research interests include Overlay network, Peer-to-Peer network and Ad hoc network.

Jan-Ming Ho received his Ph.D. in electrical engineering and computer science from Northwestern University in 1989. He received his B.S. in electrical engineering from National Cheng Kung University in 1978 and his M.S. from the Institute of Electronics, National Chiao Tung University in 1980. He joined the Institute of Information Science, Academia Sinica, Taiwan, R.O.C. as a associate research fellow in 1989, and was promoted to research fellow in 1994. He was deputy director of the Institute from 2000 to 2003. He visited the IBM T.J. Watson Research Center in the summers of 1987 and 1988, the Leonardo Fibonacci Institute for the Foundations of Computer Science, Italy in summer 1992. He is Associate Editor of IEEE Transaction on Multimedia. He is a member of IEEE and ACM. He was Program Chair of Symposium on Real-time Media Systems, Taipei, 1994–1998, General Co-Chair of International Symposium on Multi-Technology Information Processing, 1997 and will be General Co-Chair of IEEE RTAS 2001. His research interests target the integration of theory and application research, and include digital archive technology, web services, information extraction and knowledge management, content network and continuous video streaming, and combinatorial optimization.