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

Chi-Jen Wu, Kai-Hsiang Yang and Jan-Ming Ho
Institute of Information Science, Academia Sinica, Taiwan
{cjwu,khyang,hoho}@iis.sinica.edu.tw

Abstract

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-riders 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.

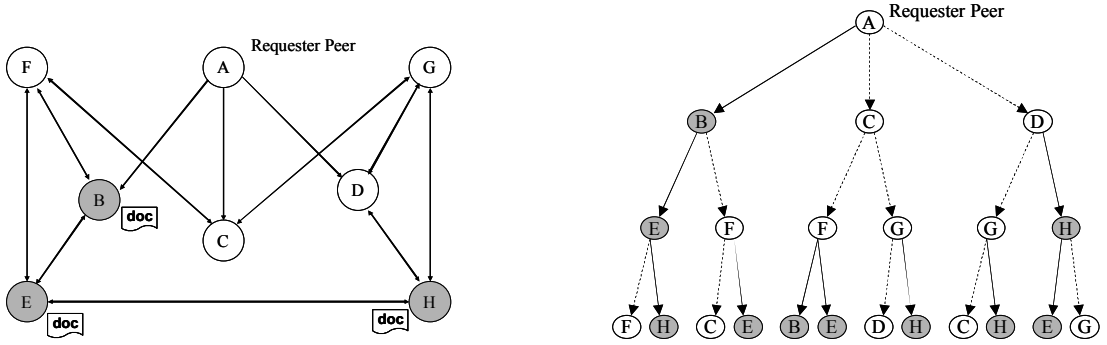
1. Introduction

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 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 a P2P file sharing system 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 Figure 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 search items 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. When multiple query packets are sent to a peer, all but the first messages are considered as redundant and useless messages. In the part (b) of Figure 1, the total 21 packets in this flooding sample consist of 10 solid lines and 11 dotted lines, where a dotted line represents a redundant packet. Hence, a flooding will cause too much



(a) (b)

Figure 1. The excessive traffic overhead in a flooding search algorithm. Part (a) depicts an unstructured P2P network topology, and peers B, E, and H contain the target documents. Part (b) is the 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.

redundant network traffic.

In this paper, we focus on the free-rider 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 present its hit rate of previous received queries, and records a list of pheromone values of its immediate neighbors. Based on these pheromone values, the AntSearch can flood a query only to those peers which are not likely to be free-riders. The main idea of the AntSearch is using pheromone values to identify the free-riders, prevent sending messages to those peer in order to reduce the redundant messages.

We have conducted several experiments to evaluate the network traffic cost and search latency in AntSearch. Compared with DQ and DQ+ search algorithm, 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. The experimental methodology and results are presented in Section 4. Finally, we summarize our results and represent our conclusions in Section 5.

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 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 research [6, 7, 8] has 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 to store the statistics about the query contents for each neighbor, 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].

As mentioned above, 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 every user to the system, and proposed a free-rider control scheme. They focus on modeling the free-riding phenomenon and studing the user behavior in P2P systems. However, our Antsearch algorithm is a feasible solution for solving the free-riding problem.

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 the search algorithm in the AntSearch.

3.1. Overview

The AntSearch algorithm is designed for solving the free-

rider 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 searched 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 obtained, the requester peer has to assign two variables before flooding this query, the first one is the k value, and the other is the TTL value. The k value represents that how much percentage of neighbors should be chosen to flood this query, and the TTL value is a bound of flooding hops. By the probe table, the requester peer estimates the search cost (including search latency and flooding messages) at every k value for choosing a suitable k value. When the 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 chooses 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 AntSearch and the dynamic querying search algorithm [1], DQ+ [4], is that 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. For this goal, each peer maintains its pheromone value and stores a list of pheromone values of its neighbors. By this data structure, a peer only propagates a query to the top k% of its neighbors with higher pheromone values. The pheromone table is used to help a peer identify the neighbors which may contain the searched file. Figure 2 illustrates search efficiency in AntSearch. Compared with the part (b) in Figure 1, each peer only floods the query to several neighbors with higher pheromone values. Intuitively, the pheromone table is a data structure to hint the direction where a searched file is located.

3.2. Pheromone Table

The objective of storing a pheromone table in each pear is to record the hit rate of previous queries in each immediate neighbor for directing a query to a neighbor with higher pheromone value. A neighbor's pheromone value represents the probability of that the neighbor is chosen to be searched. 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

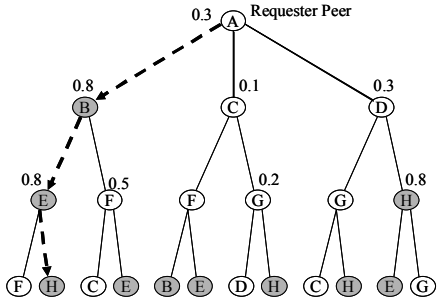


Figure 2. An example of AntSearch, the number at the peer p_i side is presented the pheromone value of the peer p_i.

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 the d_q is the degree of peer q (which means peer q has d_q number of 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 N_h/N_q is the hit rate of previous queries in peer q, and the second part of the formula is the average pheromone values of neighbors of peer q, which represents that a peer q shares few files, but its neighbors share a large amount of popular files. The value α (between 0 and 1) is used to adjust the weights between the hit rate of peer q and the average pheromone value of the neighbors of peer q. If the peer q is a free-rider, the ratio of N_h/N_q will be very small. 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 is only increased 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 peers is updated in two situations. (1) When a peer joins into the unstructured P2P system, it collects the pheromone values of its immediate neighbors. The pheromone value is sent within a 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 disconnects form the network, a peer immediately removes the pheromone value of the neighbor from the pheromone table. No other action is required in the AntSearch protocol, and it is easily to observe that the maintenance cost for a pheromone table is very limited.

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

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 process of it 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). When the flooding ends, the requester peer receives the statistics about the searched files, and it can generate the probe table, to summarize the popularity of the searched files and the results when only flooding *k*% of neighbors in each step. Table 1 shows a probe table, which consists of three columns: the *k* value, the number of searched results n_k, and the estimated number of searched peers, h_k (also called the search horizon).

In this case when *k* is 10%, each peer only forwards the query to 10% of its neighbors with higher pheromone value, and the requester peer can receive 1 results 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:

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

In order to gather the number of searched results, N_k, for each *k* during a flooding to small area, we designed a probe flooding mechanism (PFM) to obtain the 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 shows that a requester peer, A, forwards a query to its neighbor, B, which belongs to the top 10% neighbors 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, and then the peer B has to rewrite the marked *k* in the query that is forwarded to the peer C form 10% to 30%. In Step 3, when a query is hit, the peer C returns the query with marked *k* = 30% to requester peer A. Thus, the requester peer can calculate the number of searched results for each *k* in a flooding, and then generates a probe table.

(2) Flooding phase: Since a probe table is generated in the probe phase, the first step in the flooding phase is to choose a proper *k* and compute a *TTL* value for next neighbor. The Figure 4 illustrates the pseudo code of 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 should be equal to $h_k(\mathbf{N}\text{-}n_k)/n_k$. When H_k is computed, we can estimate a proper *TTL* for a flooding with a given *k* to reach the search horizon by the formula (2) in [4] as follows.

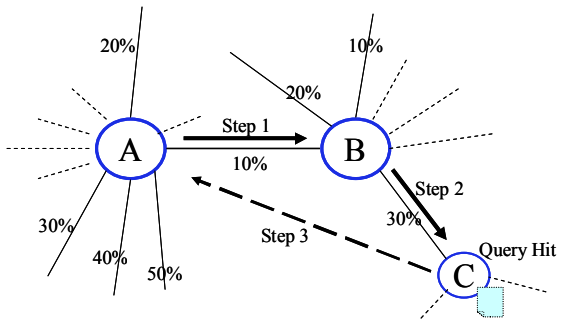


Figure. 3. An example of probe phase in AntSearch. We give a simplification of probe in the figure, the peer A, B, C should use the flooding technique to forward query. In this scenario, the peer B is the top 10% peer in the pheronome table of the peer A, the peer C is the top 30% peer in the pheronome table of the peer B. And the peer C has the queried item.

$$TTL_k \approx \log_{(Dk-1)} \frac{H_k(Dk-2)}{dk-1}$$

As the pseudo code shown in Figure 5, the requester peer has to calculate the required *TTL* value for each *k*, and finds a minimum *k* with a *TTL* less or equal to a *TTL* threshold, *MAX_TTL*. The *MAX_TTL* is generally set to 4. The *k* is then chosen and used in the following flooding iterations. After a proper *k* is chosen, the requester peer starts an iterative process to search the results (from line 7 to line 12 in Figure 4). During each iteration, the requester peer randomly chooses a neighbor, calculates a proper *TTL* to the neighbor, and then sends the query message with the chosen *k* and the calculated *TTL* to the neighbor (at line 10 in Figure 4, and the function **calculate_TTL** is implemented by the formula 3 to calculate a *TTL* value). The iteration continues until the required number of results is obtained or all neighbors are visited.

There is a tradeoff between choosing a larger *k* value and choosing a smaller *k* value. A flooding with a larger *k* value results in more query messages. For example, when *k* = 100%, it means that all the neighbors will be flooded in each step of search. Hence, it will cause much network traffic. On the other hand, choosing a smaller *k* causes a larger *TTL* value to be calculated for a flooding, which means that the flooding will take longer to retrieve the required number of results.

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

4. 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 experimental results are presented. Our simulation is processed by a simulation program which is based on that

```
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
```

Figure 4. AntSearch algorithm.

1) **Choosing_K** (*Probe_Table*, *MAX_TTL*)
2) *Begin*
3) **For** *(k=0.1* **to** *1.0)*
4) $H_k = \frac{h_k(N - n_k)}{n_k}$
5) $TTL = \log_{(Dk-1)} \frac{H_k(Dk-2)}{dk-1}$
6) **if** *TTL* <= *MAX_TTL* **then break**
7) **End for**
8) **return** *k*
9) *End begin*

Figure 5. The pseudo code of Choosing *k* rate.

used in [4]. This simulator runs on a real Gnutella network topology on February 2, 2005 [10], and simulates 160,000 peers in the network topology. 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 AntSearch, a query is propagated to random three neighbors with TTL=2. Besides, in our experiments the default maximum TTL value is 4, and the timeout interval is set to the 2.4 times TTL seconds. All the experimental setting is the same as paper [4].

During each simulation, 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 replica 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 target file), and each search aims to retrieve 100 results (N=100). For the computation formula of pheromone value, the parameter α is set to 0.7 in our experiments. According to our experiments, the parameter α does not significantly affect the performance of search 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 close to or over N. (ii) Number

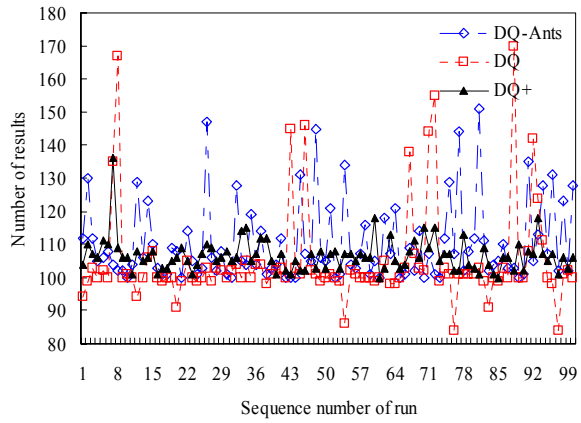


Figure 6. The performance comparison of Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants) in the number of returned results.

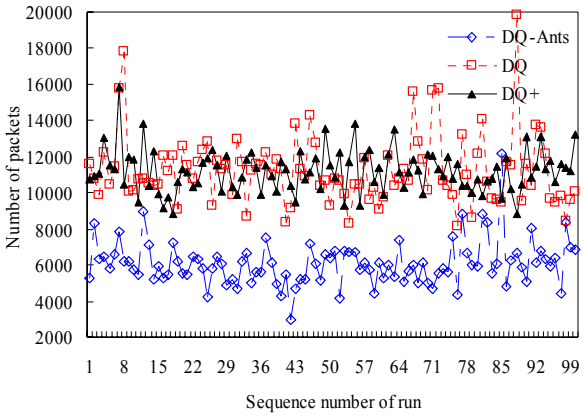


Figure 7. The performance comparison of Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants) in the number of transmitted packets.

of query messages: the number of query messages is defined as the total amount of query messages generated during the flooding process. (iii) Search latency: the search latency is defined as the total time for the flooding process.

Figure 6 shows the numbers of searched files in Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and the AntSearch (DQ-Ants) at 100 simulation runs. It is clearly to observe that, the number of searched files in DQ is less than the required number of results, because the DQ search algorithm is designed to use a very conservative approach to flooding queries. On the other hand, 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 physical number of searched peers is larger than the estimated number of peers.

Figure 7 depicts the numbers of query messages generated in the three algorithms at 100 simulation runs. First, we can observer that the total query message in AntSearch is much

smaller than those in DQ and DQ+ algorithms. The total number of query message in AntSearch is around 6,000, and that in DQ and DQ+ is approximately 12,000. In other words, AntSearch algorithm average reduces 50% of query messages during a flooding. Recall that a query is propagated to all neighbors in DQ and DQ+ algorithms, and a query is only propagated to top k% of neighbors with higher pheromone values.

Figure 8 depicts the search latency in the three search algorithm at 100 simulation runs. The search latency is an important factor for the P2P file sharing system. We can observe that the search latency in DQ algorithm is the longest, approximately 100 seconds in 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 all the required number of results from a selected neighbor. Besides, the latency in AntSearch is higher but very close to that in DQ+ algorithm, because a query is only propagated to top k% of neighbors with higher pheromone values, and in some case the search needs more iterations to gain enough searched results.

The overall comparison of the three performance metrics are listed in Figure 9. Average speaking, the number of query messages generated by a result is about 105 in DQ+, 107 in DQ, and 54 in AntSearch. It is clearly that the AntSearch did reduce approximately 50% network traffic for a query flooding, and the search latency in AntSearch is longer than that in the DQ+ algorithm by 5 seconds. Our experiments show that AntSearch can reduce a large amount of network traffic at an acceptable cost of search latency.

5. Conclusions and Future works

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 user 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 flooding 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 experimented 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 simple and easy to implement into a real system.

References

[1] A. Fisk, “Gnutella dynamic query protocol v0.1,” May 2003, Http://www9.limewire.com/develop-r/dynamic_query.html.

[2] Eytan Adar and Bernardo A. Huberman. "Free riding on gnutella," in Technical report, Xerox PARC, 10 Aug. 2000.

[3] Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems,” in Proceedingns of the Multimedia Computing and Networking (MMCN), January, 2002.

[4] Hongbo Jiang and Shuding Jin, ”Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-peer Networks,” in Proceedingns of IEEE Internet Conference on Network Protocol (ICNP), October, 2005.

[5] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. “Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems”, in Proceedingns of Internet Measurement Conference (IMC), October, 2005.

[6] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Scott Shenker, “Making Gnutella-like P2P Systems Scalable ,” In Proceedings of ACM SIGCOMM 2003

[7] B. Yang and H. Garcia-Molina,”Improving search in Peer-to-Peer networks,” In Proceedingns of the 22nd International Conference on Distributed Computing Systems (ICDCS), 2002.

[8] V. Cholvi, P. A. Felber, and E. W. Biersack, “Efficient Search in Unstructured Peer-to-Peer Networks,” in European Transactions on Telecommunications, 15(6) , 2004.

[9] Qin Lv, Pei Cao, Edith Cohen, Kai Li and Scott Shenker, “Search and Replication in Unstructured Peer-to-Peer Networks,” in Proceedings of the 16th international conference on Supercomputing (ICS) 2002.

[10] D. Shakkottai and R. Rejaie, “Characterizing the two-tier Gnutella topology,” in Proceedings of the ACM SIGMETRICS (Poster), June 2005.

[11] D. Shakkottai and R. Rejaie, “Characterizing today’s Gnutella topology,” Technique Report CIS-TR-04-02, CIS, University of Oregon, November 2004.

[12] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, “Free-Riding and Whitewashing in Peer-to-Peer Systems,” in Proceedings of the ACM SIGCOMM'04 Workshop on Practice and Theory of Incentives in Networked Systems (PINS), August 2004.

[13] Lakshmish Ramaswamy and Ling 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), January 2003.

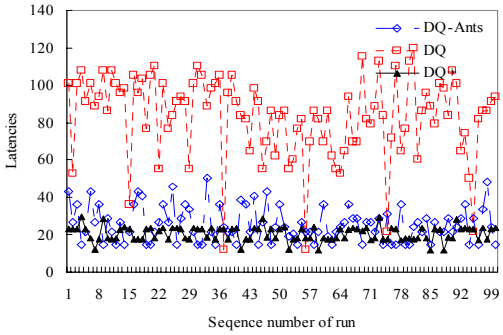


Figure 8. The performance comparison of Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants) in the number of latency.

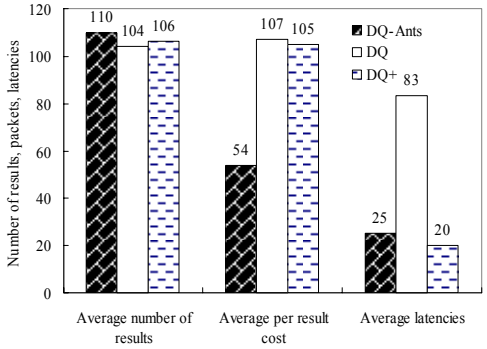


Figure 9. The comparison of Dynamic Querying (DQ), the enhanced version of Dynamic Querying (DQ+), and our AntSearch (DQ-Ants).