Towards Minimum Traffic Cost and Minimum Response Latency: A Novel Dynamic Query Protocol in Unstructured P2P Networks

Chen Tian[1]   Hongbo Jiang[1]   Xue Liu[2]   Wenyu Liu[1]   Yi Wang[1]

[1]Department of EIE, Huazhong University of Science and Technology, Wuhan, Hubei, China

[2]School of Computer Science, McGill University, Montreal, Quebec, Canada

[1]{tianchen,hxj,liuwy,ywang}@mail.hust.edu.cn, [2]xueliu@cs.mcgill.ca

Abstract—**Controlled-flooding algorithms are widely used in unstructured networks. Expanding Ring (ER) achieves low response delay, while its traffic cost is huge; Dynamic querying (DQ) is known for its desirable behavior in traffic control, but it achieves lower search cost at the price of an undesirable latency performance; Enhanced dynamic querying (DQ+) can reduce the search latency too, while it is hard to determine a general optimum parameters set. In this paper, a novel algorithm named Selective Dynamic Query (SDQ) is proposed. Unlike previous works that awkwardly processing floating TTL values, SDQ properly select an integer TTL value and a set of neighbors to narrow the scope of next query. Our experiments demonstrate that SDQ provides finer-grained control than other algorithms: its latency is close to the well-known minimum one via ER; in the mean time its traffic cost also close to the minimum. To our best knowledge, this is the first work capable of achieving best performance in terms of both response latency and traffic cost. In addition, our experiments also demonstrate that SDQ works well in various network topologies.**

Index Terms—**Distributed applications, distributed networks, distributed systems, resource discovery, Selective Dynamic Query, peer-to-peer networks, query algorithm**

I. INTRODUCTION

There are three types of architecture for peer-to-peer networks: centralized, decentralized but structured, and decentralized and unstructured [1]. Resource query is the process of searching for one or multiple copies of an item in a large, connected peer-to-peer network. Methods and performance of resources query are greatly different in these architectures. In decentralized and unstructured systems, neither a central server nor any precise control over network topology/resources placement is required. Therefore, the unstructured peer-to-peer networks present considerable challenges to the design of query algorithms. Gnutella [8] and Limewire [9] are examples of such a system. To find a file, the simplest idea is blind flooding: the request node propagates the query to the entire network. This method is un-scalable at all as it could cause huge traffic in large-scale networks like Gnutella. Efficient algorithms should retrieve sufficient results with minimum network traffic and lowest search latency.

Two main categories of enhanced query protocols are developed for unstructured networks. Controlled-flooding based algorithms try to control the flooding process instead of simple flooding: a preset TTL (time-to-live) value is carried in the packet so that the scope of the search is controlled by the TTL values. Controlled-flooding based algorithms are widely used in unstructured networks such as wireless ad hoc networks and sensor networks. Expanding Ring (ER) is the first protocol [3] of this type. Several researchers [1], [2] have compared the performance of ER [3] with other algorithms in the context of peer-to-peer networks; the Gnutella developer community proposed the Dynamic Query (DQ) technique to retrieve sufficient results at a relatively lower traffic [4]; Jiang et al. analyzed the DQ protocol and proposed an enhanced version (DQ+) [5], [13] in unstructured peer-to-peer networks.

Another category of query protocols is random-walk based. The query node sends out a query packet to be forwarded in some random fashion until it finally hits the target [1]. In general, random-walk based algorithms can reduce network traffic and enhance the system scalability; at the same time, they usually result in much longer search latency, and the number of retrieved results varies greatly for different underlying network topologies [1], [2], [6]. For energy-constrained applications such as sensor networks, random-walk based protocols are considered good choices. When adopted in unstructured peer-to-peer networks, their response latencies are too high compared to controlled-flooding based algorithms.

Our work also falls in the category of controlled-flooding. While some works had been done to search single node/objects [3], [14], [15], here we focus on multiple-copy-search algorithms, which are universal in real Internet peer-to-peer applications. In this paper, we propose a novel algorithm that could minimize the cost of search and maximize user perceivable quality of service. This paper makes the following contributions:

- We find the unsatisfactory design of previous dynamic query algorithms by extensive analysis and experiments. Specially, we find the latency using DQ+ [5] is still too high, and the usage of TTL floating value is not practical and efficient. On the other hand, we find that the neighbor heterogeneity can be exploited.
- We present a novel algorithm named Selective Dynamic Query (SDQ) which is capable of achieving almost the same performance as DQ+ in terms of traffic cost, as well as achieving almost the same performance as ER in terms of response latency. We argue that our SDQ is the

best controlled-flooding algorithm for unstructured peer-to-peer networks. Further, unlike DQ+, there is no need for SDQ to modify query message format due to the use of integer TTL values.

- We evaluate searching algorithms using two more topologies besides Gnutella. We show that our SDQ also works well in topologies other than Gnutella. SDQ adapts better than DQ+ to Flat and Power Law topologies since it results in better standard deviations of up to 75 percent.
- We show SDQ can work well when the replica is not uniformly distributed. For a skewed replica distribution, SDQ achieves almost the same performance as for a uniform distribution.

The remainder of this paper is organized as follows. We present the works of dynamic querying algorithms and discuss their ambiguities in Section II. Section III introduces the intuition behind SDQ algorithm. Section IV gives details of SDQ design. The simulation results and analysis are presented in Section V. Finally, we summarize our results and draw our conclusions in Section VI.

II. DYNAMIC QUERYING ALGORITHMS

A. Backgrounds

We first clarify some definitions. TTL value of a query packet indicates the hops of the farthest reached nodes from the query node. To be convenient, we also use $nTTL$ value, which is TTL minus 1, to denote the hops of the farthest reached nodes from the query node's direct neighbor. Besides, we assume the query node could only get its direct neighbors' degree information (number of direct neighbors), which is likely to be the case in practice.

Assume a neighbor has degree d which can be known, and the average degree of the network is D which can be estimated. As the degree of intermediate nodes are unknown, we adopt the average number of neighbors per node $D = 24$ of Gnutella characterization from [7] as their estimation. Horizon refers to expected number of queried peers. If $nTTL$ is given, the horizon within $nTTL$ hops from this neighbor H can be estimated by

$$H = (d-1) \sum_{i=0}^{nTTL-1} (D-1)^i. \qquad (1)$$

On the other hand, if H is specified, then $nTTL$ values required to reach H via this neighbor can be calculated:

$$nTTL \approx \log_{(D-1)} \frac{H(D-2)}{d-1}. \qquad (2)$$

The number of already visited nodes H_{es} can be estimated by (1). Let R_c be the results already collected, then the estimation of search item popularity P_{es} can be given by

$$P_{es} = R_c/H_{es}. \qquad (3)$$

Expanding Ring (ER) algorithm is the forerunner of controlled flooding query. ER uses successive floods with increasing TTL values: a peer starts a flood with a small TTL value (usually equals 2), and waits to see if the search is successful to retrieve sufficient results; if it is, the process stops; otherwise, the node increases the TTL by 1 and starts another flooding phase. This iterative process repeats until the required number of results is returned. ER algorithm often incurs huge overshooting and returns much more results than necessary [1]. A good search algorithm should be able to retrieve just sufficient (small or no overshooting) results for a query with a given certain required number hence results in low network traffic cost.

B. Dynamic Querying

Dynamic Querying (DQ) [4] is proposed by the Gnutella developer community. Based on the estimated popularity of the searched item, DQ dynamically adjust the scope of search by setting TTL value for next query. DQ works as follows.

(1) *Probe phase*: the query node floods a query towards a few neighbors with a small TTL value to estimate the popularity of the searched items. The search process stops if enough results are retrieved, otherwise it enters iterative flooding phase.

(2) *Iterative flooding phase*: an iterative process takes place to retrieve sufficient results. In each iterative step, the query node first computes H_{es} , then gets H_{ne} (the required number of peers should be contacted in next query) by average H_{es} with remaining connections. Let R_l and C_l denote the numbers of un-retrieved number of results and remaining connections respectively, then H_{ne} for next query could be calculated as

$$H_{ne} = \frac{R_l/P_{es}}{C_l} = \frac{R_l * H_{es}/R_c}{C_l}. \qquad (4)$$

Then DQ chooses another neighbor, calculates the $nTTL$ value for a query flooding to that neighbor by (2) and propagates a query with that TTL value 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 query node dynamically estimates the item's popularity to adjust a TTL value in each iterative flooding. Sufficient results can be retrieved at lower network traffic overhead than a blind flooding algorithm [4].

The problem is: the obtained TTL values are always floating numbers, not integer values consistent with TTL definition. A direct approach is to round this floating number into an integer as DQ works. DQ Specification in [4] proposes to round a floating value towards lower TTL.

C. Enhanced Dynamic Querying

Original dynamic querying algorithm reduces traffic cost at the price of undesirable latency performance. Nevertheless, latency performance is critical to user perceived quality of service in Internet applications. DQ algorithm is too conservative in propagating query packets to the network: when there are many remaining neighbors, a query packet is propagated to only a small fraction of the required number of peers. This method is doomed to have a high latency [5].

2

If in each iterative step the TTL is set to be a larger value and the query packet is propagated to a bigger number of peers, intuitively in just a few iterations, there will be enough returned results [5]. This is the intuition behind DQ+. The main difference between DQ and DQ+ lies in the iterative process. DQ+ iterative process is (1) greedy - in each iterative process, the query node propagates a query packet to a new neighbor hoping to find all the required number of results via this neighbor alone; and also (2) conservative - at the same time to avoid overshooting, the query node uses a Pearson's confidence interval method to provide a safety margin on the estimated popularity of the searched item. Given the required number of query-results R_c and confidential parameter $\delta \geq 0$, conservative estimation R_{es} of the true mean expected number of returned results is obtained by

$$R_{es} = R_c + \delta/2 + \delta\sqrt{R_c + \delta^2/4}, \qquad (5)$$

R_{es} is the upper limit of Pearson's confidence interval. This result reveals: if there is $R_c \geq 0$ returned results, then the expected number of returned results is less than R_{es} with a probability determined by the parameter δ. For example, when $\delta = 1$, the confidence level is about 68% , and if $\delta = 3$, the confidence level is about 99.7%. H_{ne} is calculated by

$$H_{ne} = R_l * H_{es}/R_{es}. \qquad (6)$$

To handle the obtained floating TTL values, DQ+ uses two different approaches: integer and floating. DQ(i)+, the integer version of DQ+, has the same problem as that in DQ, the TTL value is rounded up or down to an integer by a boundary value. The ratio between picking the ceiling and picking the floor is 0.3:0.7 in [5], simply given without any support. DQ+ algorithm also chooses $\delta = 3$ to provide a high confidence level.

In DQ(f)+, the floating version of DQ+, floating TTL value is supported by modify peers' forwarding algorithm. At the last hop, a relay peer only propagates query packets to a subset of its neighbors by utilizing the decimal fraction of the TTL value. Let $frac$ denotes the fraction part of TTL, the forwarding probability should be equal to $(d^{frac}-1)/(d-1)$.

D. Unsatisfactory design of dynamic query family

Dynamic query family makes a great progress in the research of query protocols in unstructured peer-to-peer networks. But its design, which is focused on float TTL values calculation and processing, is unsatisfactory.

As mentioned above, there are two approaches to handle floating TTL value. Let's discuss the simple rounding approach first. Both DQ and DQ(i)+ adjust the next TTL value by rounding the floating values. The difference between one integer TTL and the next is so great in terms of the number of peers reached that this TTL rounding calculation should be carefully dealt with. DQ Specification in [4] proposes to round a floating value towards lower TTL. A set of simulations, which will be presented in Section V, suggest that this rule is inefficient because of the one by one neighbor query nature of DQ: be more aggressive from the very beginning could

make the popularity estimation converge faster hence reduce variance in later queries. In DQ(i)+, the TTL value is rounded up or down to an integer by a boundary value. The ratio between picking the ceiling and picking the floor is 0.3:0.7 in [5], which means the boundary to ceiling is 0.3 and only values with decimal fraction bigger than 0.7 could be ceiling. It is found that an optimized boundary may relate to the item popularity, suggest that this ratio is somehow an experimental optimization more than a result of theoretical analysis.

Adjust the next query scope of search by rounding TTL floating value has two serious problems. First, it is difficult to determine a general round boundary. This boundary is neither a result of theoretical analysis nor an experience value, but more a conservative suggestion as in [4] or an experimental optimization as in [5]. A boundary value can't be optimized before we know the popularity-which is supposed to be estimated during the query process itself. Second, a fixed boundary is not optimized for individual query. For example, if we pick the ceiling and flooring as 0.3:0.7 [5], then two floating $nTTL$ values 2.695 and 2.705 would be rounded to 2 and 3 respectively. In Gnutella the former would reach only hundreds of peers and the latter would reach thousands, nearly tens of times than the former. While in DQ+ popularity calculations, their real horizon $H2$ times $H1$ should be equal to

$$(D-1)^{nTTL_2 - nTTL_1} = (D-1)^{0.01} \approx 1.032. \qquad (7)$$

This situation is obviously unreasonable, inappropriate, and risky of overshooting or latency variance.

The second approach of handling floating TTL values requires complicated implementation and deployment issues: the vast amount of existing peers should be modified to support this approach; otherwise its advantage for the request node is not obvious. In a real Internet world, an important issue related to deployment is the incentives for users to adopt the new enhanced protocol. We argue that incentives for adopting the floating version of dynamic querying algorithm are unclear, especially at the startup deployment stage. Those nodes first adopting the protocol do not benefit from the adoption of this algorithm. If such a node initiates a query, support of the last hop peers is not guaranteed, which results in longer latency and bias estimation of popularity. In one word, a node maybe make modification and relay a floating TTL value packet to represent someone else's benefits instead of its own. Therefore, realistic users in the current networks may lack the incentives to support floating versions of dynamic query algorithms.

III. INTUITION OF OUR ALGORITHM

Before delving into the details, we present observations first. In previous dynamic query algorithms DQ/DQ+ [5], when there are many remaining neighbors, in each iterative process only one query packet is propagated to only one neighbor trying to retrieve all the remaining results. Assume in a Gnutella scenario (average degree is 24) there is one iteration of DQ+ with next neighbor degree 11, the expected horizon for next query H_{ne} is estimated to be 8324. In this case,
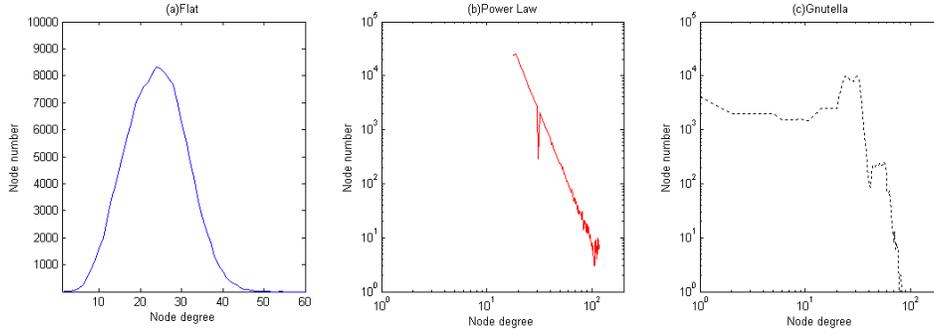
3

Fig. 1. Node degree distribution in (a) Flat topology (b) Power Laws topology and (c) Gnutella

$nTTL = 3.10$ should be used. Also assume there are other 26 neighbors left with diverse degrees, and 10 of them have total degrees sum up to 549. This implies that this query may be completed by this group of neighbors together by a group of $nTTL = 2$ packets. It is clear that the optimal utilization of neighbor heterogeneity is not achieved yet.

Furthermore, the results already involving R_c is extremely varying in the startup phase when the number of contacted peers is not large enough. If the estimation of search item popularity R_{es} is not properly deduced, the risk of overshooting exists. That is why DQ+ needs a very conservative estimation of the popularity of the searched item. Such a conservative estimation of popularity has little chance to complete the query in the first one or two iterations, hence it is difficult to further reduce response latency. Here the questions are: *why do we have to undertake the high risk of TTL choice in a TTL rounding scenario? Why do we have upgrade millions of reluctant users to support floating TTL value?*

To explore the degree heterogeneity of all remaining neighbors, the main idea behind Selective Dynamic Query (SDQ) is: rather than adjusting the scope of search based on processing the floating TTL value, SDQ dynamically selects a proper integer TTL value and the corresponding next query neighbors group based on the estimated popularity.

We emphasize that SDQ is (1) *well-planned* - it always tries to finish a query in a small TTL value, so it can limit the range of flooding hence the chance of overshooting, as well as reduce latency in each round; (2) *greedy* - in each iteration, the source peer propagates query packets to a set of neighbors by a selected TTL, to find the required number of results via these neighbors at one time, and (3) *safe* - after fixing on a low integer TTL value, SDQ focuses on neighbor selection. Take $TTL = 3/nTTL = 2$ situation as an example, one more neighbor would incur no more than hundreds of transmitted packets and a few overshooting results, while in DQ+ an aggressive TTL value selection may bother thousands of peers.

If in each iterative step the TTL value is selected *wisely* and the query packet is propagated to the *right* number of

neighbors, then it is expected that within only one or two iterations, there will be enough returned results and the cost and latency could be minimized. This is the intuition behind our approach: the source peer always tries to explore the degree heterogeneity of the whole remaining neighbors group. Authors of [12] also try to improve performance of random walk based protocols by exploiting neighbor heterogeneity. However, its design purpose, applicable areas and details are completely different from ours.

As mentioned above, DQ(f)+ also avoid TTL value rounding, while users are unlikely to have incentives to upgrade. SDQ algorithm does not require any upgrade in other peers except the query node itself, hence a flag day for transition is not necessary. SDQ benefits users immediately, and those who adopt the SDQ earlier benefit earlier. In one word, its design satisfied both altruism and self-interest.

IV. ALGORITHM DESIGN

A. Overview

Like DQ/DQ+, SDQ algorithm comprises two search phases: a probe phase and an iterative flooding phase.

(1) *Probe phase*: This phase is identical with DQ+ in [5].

(2) *Iterative Flooding phase*: Based on the estimated horizon of next query and the total remained degrees of unused neighbors, SDQ selects a proper integer $nTTL$ value and the number of required degree for next query; after that, a proper set of neighbors is picked out by the number of required degree; query packets with this $nTTL$ value are then propagated via these neighbors. The iteration process stops if the desired number of results are returned; otherwise a new estimated horizon is obtained and iterated to select another $nTTL$ value and another set of neighbors for next query. This process continues until the desired number of results are obtained or all neighbors are used. Hereby are the pseudo codes of iterative phase algorithm.

1: $R_l \leftarrow results\_need - results\_received$ {results number remains to be retrieved }
2: $H_{es} \leftarrow horizon\_esimated$ {estimated number of touched nodes}
3: $D_l \leftarrow degree\_remain$ { total degrees of available neighbors}
4: $H_{ne} \leftarrow Estimation(R_l, H_{es})$ {estimation of next horizon }

4

5: $D_{ne}, nTTL \leftarrow NextQueryTTL(H_{ne}, D_l)$ { calculate proper nTTL and required degree for next query}
6: $SelectQuerySet(D_{ne}, nTTL)$ {select a proper set of neighbors}

B. Select next TTL

In the previous algorithm, the number of all the remaining neighbors' degrees D_l is critical here. Consistent with [5], we deduce equation (8) to calculate $nTTL$ values and how many neighbors d should be covered in next given horizon.

$$d \approx \frac{H(D-2)}{(D-1)^{nTTL}} + 1. \tag{8}$$

Starting from a low $nTTL$ value (in general 1 or 2), we iteratively calculate the required number of selected neighbors' set D_{ne}. If D_{ne} is less than current total remaining degrees, this $nTTL$ value can be selected as the next query $nTTL$ candidate. For example, if $D_l = 389$ and $D_{ne} = 380$ under $nTTL = 2$ condition, then next query would select almost all remaining neighbors into query set. If this iteration can not complete the query, there is a high risk of query failure because almost all neighbors are used. We provide a safety margin by limiting used degrees in one round to be no more than two third of all remain degrees. Hereby are the pseudo codes of next TTL calculation.

```
1: Dl ← degree_remain {total degrees of available neighbors}
2: Hne ← Estimation(Rl, Hes) {estimation of next horizon}
3: for nTTL = 1 to MAX_TTL_ALLOWED do
4:     Dne ← nTTL*(AVER_DEGR-2)/pow(AVERE_DEGR-1,nTTL) + 1
5:     if Dne ≥ 2/3 Dl then
6:         continue
7:     else
8:         return   Dne, nTTL
9:     end if
10: end for
```

C. Calculating next query sets

Selecting optimum subset from the neighbor set to match D_{ne} can be solved by mathematical programming. Even with a smaller confidence level than DQ+, popularity and horizon estimations are still conservative in SDQ. Here in neighbors' selection we could deliberately introduce a little overhead to seek balance. Let n denote neighbor group index, and $A = \{a_i, 1 \leq i \leq n\}$ to be the group of neighbors' degree. We need to solve the following integer programming.

$$\begin{cases} \text{Get} : x_i = 0 || 1, 1 \leq i \leq n \\ \text{Target} : \min(\sum_{i=1}^{n} a_i x_i) \\ \text{Constraint:} \sum_{i=1}^{n} a_i x_i \geq D_{ne} \end{cases}, \tag{9}$$

By using iterative Knapsack programming, we can solve the above integer programming problem although it is computationally heavy. We could also choose a simple calculation to approximate it: before each iteration, all remain neighbors with their degree numbers are randomly organized to a list; if D_{ne} is larger than zero, we select the list head and subtract its degree from D_{ne}; the loop continues until D_{ne} value is smaller than zero; at last the total degrees of all selected neighbors should be a little more than required. As simulation results of two calculations are similar, we use the simple one as our standard implementation.

V. Evaluation

A. Evaluation methodology

We have implemented all algorithms in an event-driven simulator. We have followed the protocol specifications [1], [3]–[5]. Except for expanding ring, a node with degree at least 15 is picked to manage a search process. There is no restriction on the degree of peers which forward queries. We use the approach described in [4], [5] to estimate theoretical horizon and the average popularity of the searched item. The timeout interval is set to TTL times 2.4 seconds as recommended.

We test the performance in three different topologies: (1) in Flat topology model designed by Waxman [11], where the nodes are randomly placed on an Euclidean plane; (2) the Power Laws topology generated using [10]; (3) a snapshot of the Gnutella network topology on Feb 2, 2005 [7]. For each topology, the mean node degree is 24 and Figure 1 shows the node degree distributions. 8 different objects are located in 160K peers. Each object with replication ratio p is distributed randomly. A common probe phase suggested by [4] is used: the query is propagated down three neighbors in the neighbor list with $nTTL = 1$. Replication ratio and required number of results are specified.

The used evaluation metrics include: (1) *Response latency*: the search latency is defined as the total time needed for complete one query process and is the most important; (2) *Number of returned results*: for a query with a required number N of results, a good search algorithm should retrieve the number of results close to or only a little more than N; (3) *Number of transmitted packets*: the number of query messages is defined as the total amount of query messages generated during the flooding process; (4) *Success Ratio*: a query retrieved enough or more results than needed is considered as successful, so success ratio indicates the stability of a algorithm.

B. DQ boundary value probe

In this subsection, we try to probe the best boundary values for DQ, (the only strategic parameter). The calculated floating TTL value in DQ is rounded up or down to an integer value based on the boundary. That means if the boundary to ceiling is 0.3, only values with decimal fraction bigger than 0.7 could be ceiling, others should be flooring. Each search targets for 50 results, and four replication popularity scenarios - 0.005, 0.01, 0.02 and 0.03, are evaluated. In each case, 11 simulations are performed in total: boundary values increase from 0 to 1 with an incremental step 0.1. The bigger the boundary value is, the bigger chance of ceiling the TTL value hence bigger number of returned results in each iteration. We expect that the success ratio and number of return results may increase with the increase of boundary value, and the latency may decrease at the same time.

Fig.2 shows the results in Gnutella topology. We can see in Fig.2(a) that despite the variety of replication ratios, all results are close to 100% success (Success Ratio=1) when the boundary value is bigger than 0.9. Whatever the boundary is, DQ could control the traffic well as shown in Fig.2(b).
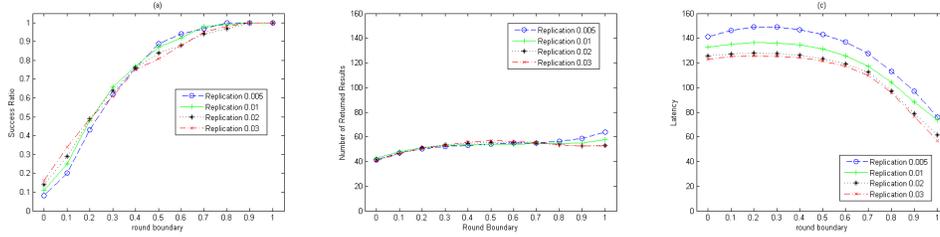
5

Fig. 2. Rounding Boundary impact to DQ

The response delay decreases dramatically when the boundary value increases. It is clear that in software implementations, simply ceiling the TTL value is the best choice: we will have the lowest latency while reduce traffic cost. If we conform to the specifications in [4] to round a floating TTL towards lower value (boundary=0.0), the results would be pretty inefficient(at least in object popularity less than 0.03 situations which is verified by our study). After tracing the simulation, we found this is because the hop- by-hop queries nature of DQ: *a more aggressive start would retrieve more results without taking the risk of overshooting; more contacted horizon also simplifies popularity estimation and reduces variance in later iterations.*

C. DQ(i)+ and DQ(f)+ parameters prob

DQ(i)+ strategy probe Next we probe the best boundary for DQ(i)+ and again use the same settings for DQ: search for 50 results, replication value set to 4 scenarios, and eleven boundary values are evaluated. Three simulations are performed for different confidential parameters $\delta = 1, 2, 3$ respectively. Fig.3, 4 and 5 give the results.

We can find that on average, each confidential parameter can find a proper boundary: (0.3,0.4,0.5) for $\delta = 1, 2, 3$ respectively. However, we observed that small δ results in big corresponding standard deviations, which implies algorithm instability. As stated before, a boundary could be neither a result of theoretical analysis nor an experience value, but a conservative suggestion or an experimental optimization.
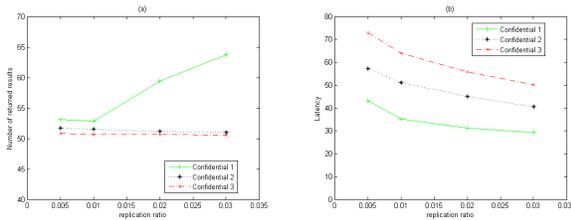

Fig. 6. DQ(f)+: Confidential Parameter impact

DQ(f)+ confidential parameter probe Although not bothered by boundary selection, DQ(f)+ still needs to fix its confidential level. Again, each search targets for 50 results and replication value set to 4 scenarios. As success ratios are always nearly 1 in DQ(f)+, other two metrics are shown in Fig.6. Replication ratio are used as x-axis. Obviously, $\delta = 1$ is not applicable. When the ratio increases, the traffics cost could be huge.

TABLE I
GNUTELLA TOPOLOGY RESULTS

algo	$\bar{R}$	σ_R	$\bar{P}$	σ_P	$\bar{L}$	σ_L
ER	218	140.88	28551	24981	15.44	3.02
DQ	57.58	26.74	6374.90	3614.37	73.47	31.59
DQ(f)+1	52.89	5.78	5815.30	956.52	35.15	14.04
DQ(f)+2	51.50	6.45	5657.99	1074.07	51.11	15.95
DQ(f)+3	50.69	1.63	5571.28	786.36	63.91	15.72
DQ(i)+1	63.44	29.13	7032.84	3599.27	54.94	23.43
DQ(i)+2	54.28	11.68	5957.17	1577.93	65.02	22.18
DQ(i)+3	51.87	4.40	5668.01	902.98	73.44	24.41
SDQ+1	54.28	7.99	5961.17	1219.00	23.27	7.16
SDQ	56.16	13.11	6172.79	1785.72	19.51	5.63

TABLE II
POWER LAW TOPOLOGY RESULTS

algo	$\bar{R}$	σ_R	$\bar{P}$	σ_P	$\bar{L}$	σ_L
ER	156.58	84.57	20517.58	26205.81	14.39	0.22
DQ	64.32	47.86	7293.81	11069.89	72.26	33.45
DQ(f)	54.12	28.21	6019.59	5230.43	36.44	13.46
DQ(i)	70.77	33.08	7874.77	4182.59	60.16	27.84
SDQ	55.87	12.95	6112.67	1767.18	23.98	6.74

TABLE III
FLAT TOPOLOGY RESULTS

algo	$\bar{R}$	σ_R	$\bar{P}$	σ_P	$\bar{L}$	σ_L
ER	160.55	94.71	21873.81	21287.84	14.76	1.81
DQ	60.24	26.73	6777.20	3419.23	73.09	29.49
DQ(f)	54.50	28.43	6188.98	7579.10	36.14	14.71
DQ(i)	62.91	27.46	7142.53	3559.30	61.39	28.12
SDQ	55.71	14.21	6214.30	1962.09	25.26	8.02

TABLE IV
SKEWED REPLICA DISTRIBUTION

algo	$\bar{R}$	σ_R	$\bar{P}$	σ_P	$\bar{L}$	σ_L
ER	194.03	132.55	37268.52	57148.25	15.45	3.00
DQ	63.54	49.41	8627.66	16968.56	70.31	33.36
DQ(f)	60.65	55.30	8059.83	15511.83	36.02	15.99
DQ(i)	67.53	51.76	9257.11	17472.37	54.79	27.31
SDQ	60.48	23.47	7731.30	5343.57	24.85	8.64

D. Performance comparison

In this subsection we make comparison among expanding ring (ER), dynamic query (DQ), enhanced dynamic query
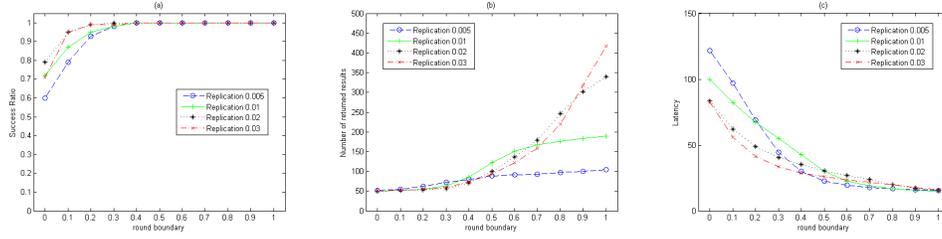
6

Fig. 3. Rounding Boundary impact to DQ(i)+: Confidential Parameter 1
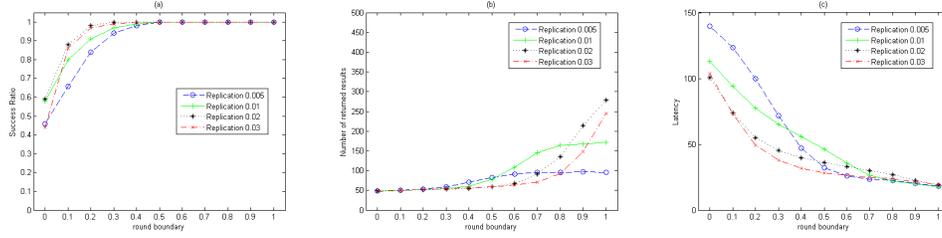

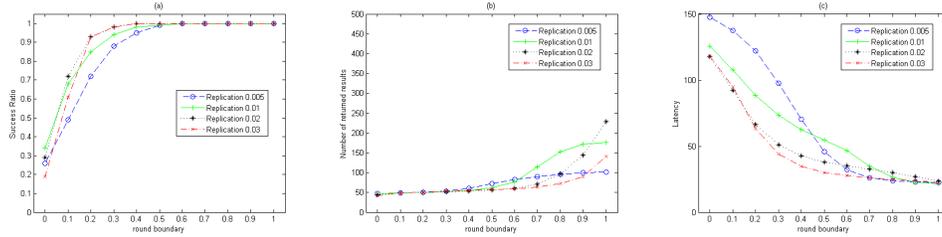Fig. 4. Rounding Boundary impact to DQ(i)+: Confidential Parameter 2

Fig. 5. Rounding Boundary impact to DQ(i)+: Confidential Parameter 3

DQ(i)+/DQ(f)+ and Selective Dynamic Query (SDQ) using three aforementioned network topologies. Each search is for 50 results, and replication value set to 0.01. DQ will round all its obtained floating TTL value to upper ceil; DQ(i)+ use 0.3 as the boundary value. Average outputs of 1000 runs are given together with their standard deviations.

We refer $\bar{R}/\bar{P}/\bar{L}$ as the mean value of Results obtained, Packets transmitted, Response Latency respectively, and σ_R, σ_P, σ_L as their corresponding standard deviation. The number after algorithm name denotes confidential parameter δ for DQ(f)+/DQ(i)+/SDQ respectively, for example, DQ(f)+3 means $\delta = 3$ and SDQ means no conservative estimation. Due to space limit, we only provide results with varying δ values in Table I. To achieve fair comparison, for the rest of the simulations we uniformly set $\delta = 1$ for those algorithms where this parameter is required.

Table I, Table II and Table III show the results in three network topologies. The performance differences of the algorithms are significant. In terms of transmitted packets, expanding ring has almost 3-4 times of traffic cost compared with others. DQ(f)+ has the minimum number of results returned and the minimum number of transmitted packets, which reflects its effect of fine-grain control in last forwarding

hop. With regard to latency, DQ has the most undesirable characteristic including average value and variance. This is the consequence of its conservative connection by connection query nature. DQ(i)+ and DQ(f)+ fall into the same level. We argue that SDQ is the best one here: its performance is close to ER in terms of delay, and is close to DQ algorithms in terms of transmitted packets. Also, SDQ exhibits a good performance in all three network topologies. In Table II and Table III, we found that SDQ results in much smaller number of transmitted packet ($\bar{P}$) and much smaller latency than other DQ algorithms. Also, in Flat and Power Law topologies, SDQ is significantly stable with a small variation compared with other DQ algorithms. Next, we turn to investigation of the impact of δ. Table I shows that the larger δ is, the better traffic controlled and the larger response latency are. This characteristics are exhibited by both DQ(f)+/DQ(i)+/SDQ. However, latency of DQ(i) increases quickly with δ increases. We trace the simulations and find that, when there are only 1 or 2 results still need to be retrieved, too conservative estimation makes DQ(i) staying in $nTTL = 1(TTL = 2)$ state continuously, and thus obstructs finishing the search.
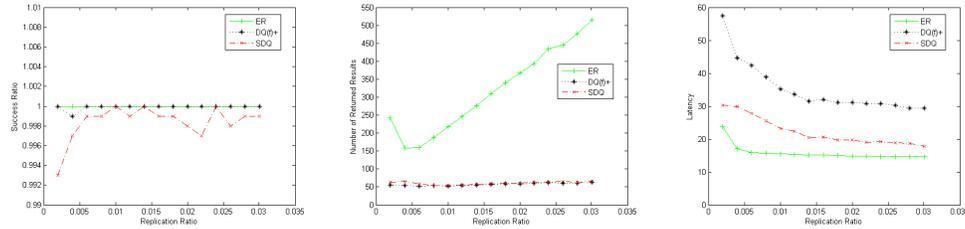
7

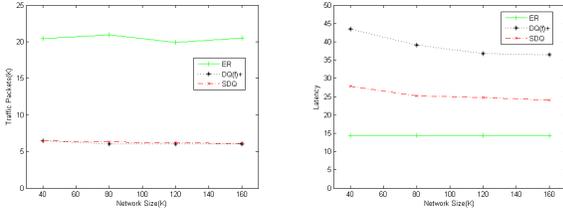Fig. 8. Performance comparison for replica ratio impact: (1)success ratio (2)number of results and (3)latency.

Fig. 7. Performance comparison with a variety of network sizes: (1)number of transmitted packets and (2)latency.

E. Sensibility to network configuration

Skewed replica distribution Shown in Table IV, we turn to study skewed replica distribution instead of uniform one with Flat network topology which is the only case that incorporates Euclidean proximity. 80% replica are put on the left half of area, and the rest 20% replica on the right. Also, SDQ shows a good performance, resulting in a small traffic cost and small latency. Furthermore, Figure 7 shows the results in different network scales. Again, SDQ archives good performance in terms of both traffic cost and latency.

Sensibility to replication ratio To extensively evaluate algorithm performance under different conditions, we study ER/DQ(f)+/SDQ in a broader range. First we set the number of returned results to 50 and evaluate algorithm performance in different replication values. The replication values are increased from 0.004 to 0.03 by a 0.002 step. Totally 14 scenarios are scheduled, each with 1000 runs. The average success ratio, returned results and latency are shown in Figure 8 respectively. We omit results of transmitted packets here because of its high relevance with return results. Success ratio metric is added to illustrate the algorithm stability.

VI. CONCLUSION

In this paper, we propose a novel searching protocol: Selective Dynamic Query (SDQ) in unstructured P2P networks. Rather than adjusting the scope of search based on the floating TTL value processing in DQ/DQ+, SDQ dynamically selects the next query neighbors group based on the fine-grain estimated popularity and a proper TTL value. Our experiment results show that compared with previous two versions of DQ+ search algorithm, the SDQ algorithm on average reduces 50% latency with almost the same traffic cost. The latency and the traffic of SDQ are all close to minimum, while it still retrieves sufficient results for a query. We will further explore SDQ for a wider range of applications, in particular, other unstructured networks such as wireless ad hoc network and sensor networks.

REFERENCES

[1] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. "Search and replication in unstructured peer-to-peer networks". In Proceedings of International Conference on Supercomputing, November 2002.
[2] C.Gkantsidis, M. Mihail, and A. Saberi. "Hybrid search schemes for unstructured peer-to-peer networks". IEEE INFOCOM 2005.
[3] N. Chang and M. Liu. "Revisiting the TTL-based controlled flooding search: Optimality and randomization". In Proceedings of ACM Mobi-Com, September 2004.
[4] A. Fisk, "Gnutella dynamic query protocol v0.1," May 2003, http://www9.limewire.com/develop-r/dynamic_query.html.
[5] H. Jiang and S. 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.
[6] C. Gkantsidis, M. Mihail, and A. Saberi. "Random walks in peer-to-peer networks". In Proceedings of IEEE INFOCOM, March 2004.
[7] D. Stutzbach and R. Rejaie. "Characterizing the two-tier Gnutella topology". In Proceedings of ACM SIGMETRICS (Poster), June 2005.
[8] Open Source Community. Gnutella. In http://gnutella.wego.com/, 2001.
[9] Limewire. http://www.limewire.com/
[10] C. Palmer and G. Steffan: "Generating network topologies that obey power laws", in Proc. IEEE Globecom 2000.
[11] B. Waxman: "Routing of multipoint connections", IEEE Journal on Selected Areas in Communications, vol. 6, no. 9, pp. 1617C1622, December 1988.
[12] Q. Lv, S. Ratnasamy, and S. Shenker. "Can heterogeneity make Gnutella scalable", In proceedings of first international workshop on peer-to-peer systems (IPTPS), 2002
[13] S. Jin and H. Jiang. "Novel Approaches to Efficient Flooding Search in Peer-to-Peer Networks", Computer Networks, Vol.51(10), 2007.
[14] N. Chang and M. Liu, "Controlled flooding search with delay constraints", in IEEE INFOCOM , April 2006, Barcelona, Spain
[15] N. Chang and M. Liu, "Optimal controlled flooding search in a large wireless network", in Proc. 3rd International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt'05), April 2005, Trentino, Italy