Revisiting Dynamic Query Protocols in Unstructured Peer-to-Peer Networks

Chen Tian, Hongbo Jiang, *Member*, *IEEE*, Xue Liu, *Member*, *IEEE*, and Wenyu Liu, *Member*, *IEEE*

Abstract—In unstructured peer-to-peer networks, the average response latency and traffic cost of a query are two main performance metrics. Controlled-flooding resource query algorithms are widely used in unstructured networks such as peer-to-peer networks. In this paper, we propose a novel algorithm named Selective Dynamic Query (SDQ). Based on mathematical programming, SDQ calculates the optimal combination of an integer TTL value and a set of neighbors to control the scope of the next query. Our results demonstrate that SDQ provides finer grained control than other algorithms: its response latency is close to the well-known minimum one via Expanding Ring; in the mean time, its traffic cost is also close to the minimum. To our best knowledge, this is the first work capable of achieving a best trade-off between response latency and traffic cost.

Index Terms—Peer-to-peer networks, query algorithm, selective dynamic query.

1 Introduction

Peer-to-peer overlay networks, running at the application layer, perform scheduling and routing without any knowledge of the underlying physical networks. Various peer-to-peer systems have became the most popular Internet applications and a major portion of the Internet traffic is attributed to them.

Topologies of peer-to-peer systems can be divided into three different categories: centralized, decentralized but structured, and decentralized and unstructured. Napster-like [15] centralized systems have their resource directories hosted at some central servers. A centralized topology scales poorly and suffers from the single-point-of-failure problem. Chord [25] and Tapestry [30] are decentralized, but their network topologies are highly structured and their resources are placed by distributed-hash-table algorithms. It is not surprising that these topologies are sensitive to the extremely transient join/leave/failure behaviors of peers, which is, unfortunately, an intrinsic characteristic of Internet peer-to-peer applications.

Nowadays, the unstructured topology is a popular model in some peer-to-peer systems since: 1) the unstructured peer-to-peer systems are highly resilient to peers' failures and incur a very low overhead at peer arrivals and departures; 2) they are simple to be implemented and have little overhead in topology maintenance. Gnutella [9] and Limewire [19] are examples of such file-sharing systems. In decentralized and unstructured systems, neither central servers nor any precise managements over network topology/resources placement are required [4].

In this paper, we consider the problem of a resource query in unstructured peer-to-peer networks [12], [17], [18]. That is, *given an overlay network $G = (V, E)$, m copies of a data item published in V, and a predefined parameter $N (m \geq N)$, the goal of the study is to enable any inquiry peer in V to discover at least N identical data copies published in G at the expense of the minimum traffic cost and response latency*. Here, the traffic cost is defined as the number of messages required to complete a query, and the response latency is the time spent to collect at least N identical data copies. Since unstructured networks do not offer any clue to facilitate a resource query, researchers are facing considerable challenges when designing the query algorithms in those networks.

A novel algorithm named Selective Dynamic Query (SDQ) is proposed in this paper. By solving a Knapsack programming problem, SDQ calculates the optimal combination of a group of neighbors with a proper integer TTL value for the next query round. Our extensive simulation results (with different network topologies, and with various object replica scenarios etc.) show that SDQ exhibits superior performance over existing protocols.

The remainder of this paper is organized as follows: we summarize related work in Section 2. We present some background in Section 3. Section 4 introduces the intuition behind the proposed SDQ algorithm. Section 5 presents the design details of SDQ. The simulation results and analysis are presented in Section 6. Finally, we summarize our results and draw our conclusions in Section 7.

2 Related Work

To allow resource querying in unstructured peer-to-peer networks, two main categories of query protocols are developed. Controlled-flooding-based algorithms, as the name suggests, control the iterative flooding process: instead of blind flooding, an integer TTL value is carried in each packet of an individual query round; the scope of the

• C. Tian, H. Jiang, and W. Liu are with the Wuhan National Laboratory for Optoelectronics, the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan 430074, P.R. China. E-mail: hongbojiang2004@gmail.com.
• X. Liu is with the University of Nebraska-Lincoln, 104 Schorr Center, Lincoln, NE 68588.

Manuscript received 7 June 2010; revised 12 Jan. 2011; accepted 14 Feb. 2011; published online 17 Mar. 2011.
Recommended for acceptance by L. Xiao.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-06-0335.
Digital Object Identifier no. 10.1109/TPDS.2011.111.

flooding can then be controlled. Controlled-flooding-based algorithms are widely used in unstructured networks such as wireless ad hoc networks [5], [6]. Expanding Ring (ER) is the first such protocol [5]. Several researchers [14], [20] have compared the performance of ER [5] with other algorithms in peer-to-peer networks. The Gnutella developer community proposed the Dynamic Query (DQ) technique to retrieve enough results at a low traffic cost [12]; Jiang and Jin analyzed the DQ protocol and then proposed an enhanced version (DQ+) [17], [18] to reduce response latency.

The second category of query protocols is random-walk-based. The query node sends out a query packet, which is then forwarded in a random fashion until it finally hits the target [20]. Lv et al. [21] strived to improve the performance of random-walk-based protocols by exploiting neighbor heterogeneity. In biased random walks [2], [31], a node has statistical preference to forward the walker toward the target, so as to reduce the excepted number of steps before the target is reached. In general, random-walk-based algorithms can reduce network traffic and enhance the system scalability. On the downside, they usually result in much longer search latency, and the number of returned results can greatly vary in different underlying network topologies [13], [14], [20]. For energy-constrained applications, random-walk-based protocols are considered as the good choices. While in unstructured peer-to-peer networks, their response latencies are too high to be acceptable for users.

In addition to direct query, some orthogonal studies have been developed for unstructured networks to achieve low search latency at relatively low traffic cost. Crespo and Garcia-Molina [10] proposed routing indices to find documents with content of interest across potential peer-to-peer sources, by having each node to maintain the indices. Doulkeridis et al. [11] extended this scheme to enable multidimensional routing indices. Apart from highly popular files, Puttaswamy et al. [24] proposed to use index replication [8] to find "rare" objects. With this scheme, every node stores just the metadata of its data on all of its one/multihop neighbors. Ioannidis [16] proposed to treat the failed query as "evidence of absence" of the file when a node fails to locate a file. This information is then stored and shared with other peers in order to quickly stop the search and inform the inquiry node that the file is unavailable. By doing so, the proposed scheme can reduce the traffic cost and search delay significantly. In [29], network traffic in a query flooding can be reduced by only sending queries to nodes that are not likely to be free riders [1]. Chawathe et al. [7] proposed to direct queries to high-capacity nodes, and hence increase the chance of finding the request item. GES [32] combined techniques from information retrieval (IR) to enhance search performance in terms of search efficiency and quality. Some researchers attempt to improve search efficiency by exploiting the geographical and temporal locality [3].

3 Backgrounds

We first state some definitions used in this paper. In each query packet, the TTL value indicates the hops from the farthest reached node to the inquiry node. For simplicity, we also use $nTTL$ value ($nTTL = TTL - 1$) to denote the hops from the farthest reached node to one of the inquiry node's direct neighbor which is passed by the packet. Consistent with the state of the art, we assume that the inquiry node could (only) know its direct neighbors' degree information (number of direct neighbors), which is likely to be the case in practice. The average degree of the network is D, which can be estimated.

As the degrees of intermediate nodes are unknown, the inquiry node can adopt the average value D as their estimation. Horizon H refers to the expected number of queried peers in a query round. If the $nTTL$ value and a neighbor's degree d are given, the horizon H within $nTTL$ hops from this neighbor can be estimated by

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

On the contrary, if H is given, then the $nTTL$ value required to reach H via this neighbor can be calculated by

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

The number of all visited nodes after a query round H_{es} can be calculated by the summation of the estimation of all finished rounds; let R_c denote the number of results that already collected; then the estimated popularity of the targeted item P_{es} can be given by

$$P_{es} = R_c / H_{es}. \tag{3}$$

4 Intuition of Our Algorithm

Before delving into the details, we present our intuition first. In DQ+ [17], when there are many residual neighbors at one round, only one query packet is propagated to only one neighbor, trying to retrieve all the residual results. In a Gnutella system with the average degree $D = 24$, assume that there is one iteration of DQ+ with next neighbor degree $d = 12$, and the expected horizon $H_{ne} = 8{,}324$. In this case, $nTTL = 3.10$ should be used according to (2). We also assume that there are other 26 neighbors left with different degrees; 10 peers among them have the total degrees of 357. The observation is that the query could be completed within one iteration, by sending these 10 neighbors together with $nTTL = 2$ query packets (we will provide a detailed analysis on how to calculate this value in Section 5.3). It is obvious that the optimal utilization of neighbor heterogeneity is not achieved yet.

What's more, the number of already returned results R_c is variable in the startup phase due to the random nature of item replication, as the number of visited peers is not large enough yet. If the estimation of item popularity P_{es} is not properly deduced, the risk of overshooting takes place. That is why DQ+ needs a very conservative estimation of the returned copies R_{es}. Such a conservative estimation has little chance to complete the query in the first one or two iterations; hence, it is difficult to further reduce the response latency. Here, the questions are: *why do we have to undertake the high risk of TTL rounding? Why do we have to upgrade millions of reluctant users to support floating TTL values?*

The main idea behind SDQ is: rather than processing the floating TTL value, SDQ dynamically calculates an optimal combination of a proper integer TTL value and the corresponding group of neighbors via mathematical programming.

We emphasize that SDQ is 1) *well planned*—every query round always tries to finish the query in a small TTL value, so that the inquiry node can limit the range of flooding and the chance of overshooting, and reduce perspective response latency; 2) *greedy*—in each iteration, the inquiry node propagates the query packets to the calculated subset of neighbors with the calculated TTL value, expecting to obtain the required number of results via these neighbors at one time; and 3) *safe*—with relatively low integer TTL values, SDQ can be more aggressive on neighbor selection. Take $TTL = 3/nTTL = 2$ as an example: selecting one more neighbor would incur just hundreds of visited peers and a few overshooting results; in DQ+, an aggressive TTL value may bother thousands of peers.

SDQ is in fact an optimization process with two objectives: minimizing both response latency and traffic cost. If in each iterative round the TTL value is selected *wisely* and the query packets are propagated to the *right* number of neighbors, it can be expected that within only one or two iterations, there would be enough returned results and the cost and latency could be minimized. This is the intuition behind our approach: the inquiry node always tries to exploit the degree heterogeneity of all residual neighbors. As mentioned above, DQ(f)+ also avoids TTL value rounding, while users are unlikely to have incentives to upgrade. The SDQ algorithm does not require any upgrade in other peers except the inquiry node itself; hence, the flag data for transition are not necessary.

5 Algorithm Design

5.1 Overview

The SDQ algorithm is comprised of two query phases: a probe phase and an iterative flooding phase.

1. *Probe phase*. This phase is similar to $DQ+$.
2. *Iterative flooding phase*. Based on the estimated horizon of next query H_{ne} and the total residual degrees of all unused neighbors, SDQ calculates a proper integer $nTTL$ value and the number of required degree; after that, a subset of the neighbors are chosen according to the number of required degrees; query packets with this $nTTL$ value are then propagated via these neighbors. The iteration process stops if enough results are returned; otherwise, a new query round is initiated. Hereby are the pseudocodes of an 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}
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}
```

5.2 Estimating Item Popularity and Next Horizon

To avoid overshooting in flooding phase, like DQ+, we use the Pearson's confidence interval to provide a safety margin on the estimation of the targeted item's popularity. Compared with $DQ+$, SDQ is less likely to have a big overshooting with a relatively smaller perspective TTL value. Therefore, we could be more aggressive than DQ+ in estimation of the popularity (results of $\delta = 1$ and $\delta = 0$ are both presented in simulations). Hereby are the pseudocodes of popularity and horizon estimation.

```
1: $Initialization$
2: $R_c \leftarrow results\_received$ {results retrieved}
3: $H_{es} \leftarrow horizon\_esimated$ {estimation number of touched nodes}
4: $P_{es} \leftarrow estimation\_popularity$ {estimated popularity}
5: $R_{es} \leftarrow 0$ {conservative estimation of results}
6: $R_l \leftarrow results\_total - results\_received$ {results left to be retrieved}
7: $R_{es} \leftarrow R_c + \delta/2 + \sqrt{R_c + \delta^2/4}$
8: $P_{es} \leftarrow R_{es}/H_{es}$
9: $H_{ne} \leftarrow R_l/P_{es}$
```

5.3 Selecting Next TTL

It is well known that the most multiobjective optimization problem is hard to find an optimal solution. Fortunately, we can exploit the integrity of the $nTTL$ values. The number of all the residual neighbors' degrees D_l is the key here. For each possible $nTTL$ value, we can use (4) to calculate the number of neighbors' degrees d which should be covered; given the estimated horizon of the next round, an optimal $nTTL$ value can then be derived

$$d \approx \frac{H(D-2)}{(D-1)^{nTTL}} + 1. \tag{4}$$

Starting from a small $nTTL$ value (in general 1 or 2), we iteratively calculate the required number of selected neighbors' degrees D_{ne}. If D_{ne} is less than the current residual degrees D_l, this $nTTL$ value can be selected as a candidate for the next query. Let's assume an example where $D_l = 389$ and $D_{ne} = 380$ with $nTTL = 2$, and then the next query round would select almost all residual neighbors into the query set; if this round eventually cannot complete the query, there is a high risk of query failure because almost all neighbors are already used. Here, another safety margin is needed to limit the total degrees used in a single round. We set this limitation to be no more than two-third of all residual degrees; this is an experience value (our experimental results show no significant difference for this value from 0.5 to 0.8). Hereby are the pseudocodes of next TTL value calculation.

```
1: $D_l \leftarrow degree\_remain$ {total degrees of available neighbors}
2: $H_{ne} \leftarrow Estimation(R_l, H_{es})$ {estimation of next horizon}
3: for $nTTL = 1$ to $MAX\_TTL\_ALLOWED$ do
4:     $D_{ne} \leftarrow \frac{nTTL*(AVER\_DEGR-2)}{pow(AVERE\_DEGR-1, nTTL)} + 1$
```

```
5: if $D_{ne} \geq \frac{2}{3} D_l$ then.
6: continue
7: else
8: return $D_{ne}, nTTL$
9: end if
10: end for
```

5.4 Calculating Next Query Set

Selecting the optimal subset of residual neighbors to reach D_{ne} can be solved by mathematical programming. Even with a smaller confidence level than DQ+, the popularity and horizon estimations are still conservative in SDQ. Here, in neighbors' selection we can deliberately introduce a little remedy to achieve balance. Let n denote the number of residual neighbor and i denote the neighbor index. $A = \{a_i, 1 \leq i \leq n\}$ is the set of neighbors' degrees. We define the following decision variable:

$$x_i = \begin{cases} 1, & \text{if neighbor } i \text{ is chosen for next query,} \\ 0, & \text{otherwise,} \end{cases}$$

and the integer programming formulation

$$\begin{cases} \text{Get} : x_i = 0 \| 1, 1 \leq i \leq n, \\ \text{Target} : \min\left(\sum_{i=1}^{n} a_i x_i\right), \\ \text{Constraint} : \sum_{i=1}^{n} a_i x_i \geq D_{ne}. \end{cases} \tag{5}$$

Generally speaking, it is difficult to solve integer optimization problems because of their inherent combinatorial complexity. Fortunately, we can use variable replacement to translate (5) to an equivalent Knapsack programming problem. With

$$\begin{cases} y_i = \overline{x_i}, 1 \leq i \leq n, \\ D_f = D_l - D_{ne}, \end{cases} \tag{6}$$

we have

$$\begin{cases} \text{Get} : y_i = 0 \| 1, 1 \leq i \leq n, \\ \text{Target} : \max\left(\sum_{i=1}^{n} a_i y_i\right), \\ \text{Constraint} : \sum_{i=1}^{n} a_i y_i \leq D_f. \end{cases} \tag{7}$$

This is a Knapsack problem, and the optimal solution can be solved by mature algorithms [22].

However, the exact Knapsack algorithm has high computational complexity. Due to the approximation nature of our algorithm, an exact optimal solution is unnecessary. We develop a polynomial-time heuristic algorithm: before each iteration, all residual neighbors 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. Hereby are the pseudocodes of subset selection.

```
1: $Initialization$
2: $ListHead \leftarrow$ {index to available neighbor list}
3: $D_m \leftarrow degreeofhead$
4: $Nberlist \leftarrow neighborlist$
5: for all $D_{ne} > 0$ do
6: chooseNeighbor(Nberlist [ListHead])
7: $D_{ne} = D_{ne} - D_m$
8: end for
```

Simulation results, which will be presented later in Section 6, demonstrate that the performances of our heuristic method is comparable to the original Knapsack solution. The main contribution of the heuristic method is that the low computational complexity makes SDQ a competitive candidate for other unstructured networks with lower node capability (such as wireless sensor networks).

6 Evaluation

In this section, we first describe our evaluation methodology. After this, we comprehensively compare the performance of these algorithms under different conditions.

6.1 Evaluation Methodology

We have implemented our SDQ algorithm in an event-driven simulator. The hardware platform is Intel P4 2.8G CPU with 2G memory. We have followed the protocol specifications [5], [12], [17], [20]. Except for Expanding Ring, a node with degree of at least 15 is picked to manage a query process; there is no restriction on the degree of peers which forward query packets. We use the approach described in [12] and [17] to estimate the theoretical horizon and the average popularity of the targeted item. The default maximum TTL value allowed for each neighbor is 4. The timeout interval is set to TTL times 2.4 seconds as recommended.

We evaluate the performance in three different topologies: 1) flat topology model designed by Waxman [28], where the nodes are randomly placed on an euclidean plane; 2) the Power Laws topology generated using the algorithm described by Palmer and Steffan [23]; 3) a snapshot of the Gnutella network topology on February 2, 2005 [26]. For each topology, the average node degree is 24. Eight different items are replicated in 160 K peers. Each item with replication ratio p is distributed randomly. A common probe phase suggested by [12] is used: the first query round packets are propagated down three neighbors with $nTTL = 1$. The replication ratio and the required number of results are specified in each experiment. Each evaluation result is the average of 1,000 independent simulation runs.

The evaluation metrics include:

1. *Response latency*: the latency is defined as the total time needed for complete one query process and is the most important metric.
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 required is considered as a successful query. The success ratio indicates the stability of an algorithm and should be larger than the accepted level (such as 99 percent).

TABLE 1
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

5. *Overshooting ratio*: the ratio of the number of returned results over the number of required.

6.2 Performance Comparison

In this part, we evaluate the performance comparisons among ER, DQ, enhanced dynamic query DQ(i)+/DQ(f)+, and SDQ, in three aforementioned network topologies. Each query targets for 50 returned results, and the replication value is fixed to 0.01. DQ rounds all its obtained floating TTL value to the ceil; DQ(i)+ uses 0.3 as the boundary value. Average outputs of 1,000 runs are given in Tables 1, 2, and 3, together with their standard deviations.

We refer to $\bar{R}/\bar{P}/\bar{L}$ as the mean values of *Number of Returned Results*, *Transmitted Packets*, and *Latency*, respectively, and σ_R, σ_P, σ_L as their corresponding standard deviations. The digits after the algorithm names denote the 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 limitation, we only provide results with various δ values in Table 1. To achieve fair comparison, for the rest of the simulations we uniformly set $\delta = 1$ for all algorithms wherever this parameter is required.

Tables 1, 2, and 3 demonstrate the results in three different network topologies. The performance differences of the algorithms are significant. In terms of *Transmitted Packets*, ER is almost three to four times over others. DQ(f)+ has the minimum *Number of Returned Results* and the minimum number of *Transmitted Packets*; its fine-grained control on last forwarding hop is effective. As regards to *Latency*, DQ has the most undesirable performance including both mean value and variance. This is the consequence of its conservative one-by-one query nature; DQ(i)+ and DQ(f)+ fall into the same level. We argue that SDQ is the best algorithm here: its performance is close to ER in terms of *Latency*, and is close to DQ algorithms in terms of *Number of Returned Results* and *Transmitted Packets*.

First, we investigate the impact of δ value over the algorithms. Table 1 shows that the larger the δ value, the better the traffic being controlled and the larger the *Latency*. This characteristic is shared by DQ(f)+, DQ(i)+, and SDQ. It is interesting that the *Latency* of DQ(i) increases more rapidly when δ increases. We trace the simulation logs and find that when there are only one or two results still need to be retrieved, too conservative estimation would stuck DQ(i) in the $nTTL = 1 (TTL = 2)$ state, and thus suffer the completion of the whole query.

Also, SDQ exhibits good performance in all three network topologies. In Tables 2 and 3, we find that SDQ can result in much smaller *Transmitted Packets* ($\bar{P}$) and much smaller *Latency* than other DQ algorithms. Also, in both Flat and Power Law topologies, SDQ is significantly stable with small variations compared to others.

TABLE 3
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

6.3 Sensitivity to Replica Distribution

6.3.1 Skewed Replica Distribution

Instead of uniform replication scenarios, in this part we analyze the impact of skewed replica distribution over all algorithms. Intuitively, the ER algorithm should not be affected. With high degree numbers, all topologies have small radius; we expect that this *small world* nature would make all dynamic query algorithms insensitive to skewed replica distribution.

Flat topology. As shown in Table 4, our first evaluation is in Flat network topology which is the only topology that incorporates euclidean proximity. In this scenario, 80 percent of all replicas are put on the left half of the area, and the rest 20 percent replica on the right part; the density of each item in the left half network is four times over that in the right half.

TABLE 2
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 4
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

TABLE 5
Gnutella Topology with Hot Spots

algo	$\bar{R}$	σ_R	$\bar{P}$	σ_P	$\bar{L}$	σ_L
ER	235.64	123.70	24953.60	17271.31	15.20	2.65
DQ	56.62	22.5	5202.145	2519.88	69.62	31.82
DQ(f)	55.31	9.86	5098.49	1091.32	32.06	14.91
DQ(i)	63.58	30.36	5898.152	3219.32	50.69	21.73
SDQ	57.60	13.32	5251.34	1375.50	21.72	7.51

Compared with uniformly distributed scenarios, the *Number of Returned Results* is only slightly increased for all algorithms. This is consistent with our expectation, and overall, SDQ still shows a good performance, with a combination of small traffic and small latency, compared with others.

Hot spots. Table 5 shows the results with the Gnutella topology while we intensively create resources hot spots: after randomly selecting a node, we also allocate the items to all its neighbors; so on and so forth until all items are distributed. We found that ER, DQ, DQ(f), DQ(i), and SDQ exhibit stable performance, similar to uniform replica scenarios in Table 1.

6.3.2 Sensitivity to the Replication Ratio

To extensively evaluate algorithms' performance under different conditions, we study ER/DQ(f)+/SDQ with a large range of the replication ratios. First, we evaluate their performances in low replication value scenarios. The replication values are increased from 0.002 to 0.03 by a 0.002 step. Total 15 scenarios are scheduled, each with 1,000 runs. The average *Number of Returned Results* and *Latency* are shown in Fig. 1, respectively.

For every replication value, SDQ and DQ(f)+ are in the same level in terms of traffic cost and outperform ER. In terms of response latency, SDQ has a much better performance than DQ(f)+, and its latencies are close to the minimum one via ER. There is a turn point of ER: when the replication ratio increases near 0.004, the traffic suddenly drops and starts to increase again and after that, the latency maintains steady after that point.

In the high replication ratios group, we also evaluate a broader popularity from 0.03 to 0.30 by a 0.03 step. Total 10 scenarios are scheduled, each with 1,000 runs. The average *Number of Returned Results* and *Latency* are shown is Fig. 2, respectively. The performances are similar to that in low replication ratio scenarios. Consistent with our analysis, there is another turn point of ER near 0.09, again illustrating its inefficiency.

6.4 Impacts of Other Factors

Impact of network scale. Fig. 3 shows the algorithm performances in different network scales. Again, the replication value is 0.01 and 50 results are required. There are four different Power Laws topologies, generated using [23], with 40, 80, 120, and 160 K nodes, respectively.

All algorithms maintain stable performance. Again, SDQ archives the best overall control of both traffic cost and response latency than others. At least in tens of thousands scale, the network scale has no obvious impact on query algorithms.

Impact of required number of results. In this part, we fix the replication value to be 0.01, and evaluate the algorithms with different required numbers of results needed. Here, we introduce a new metric *Overshooting Ratio*. Let O, C, and N stand for *Overshooting Ratio*, *Number*

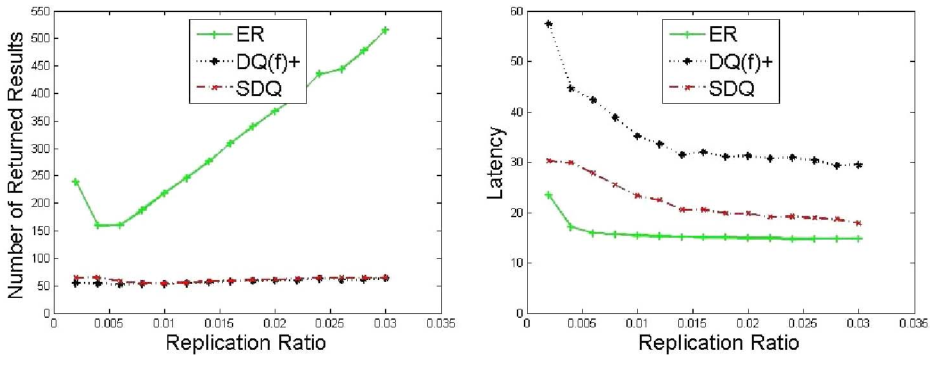


Fig. 1. Performance comparison for replica ratio impact: (1) number of results and (2) latency.

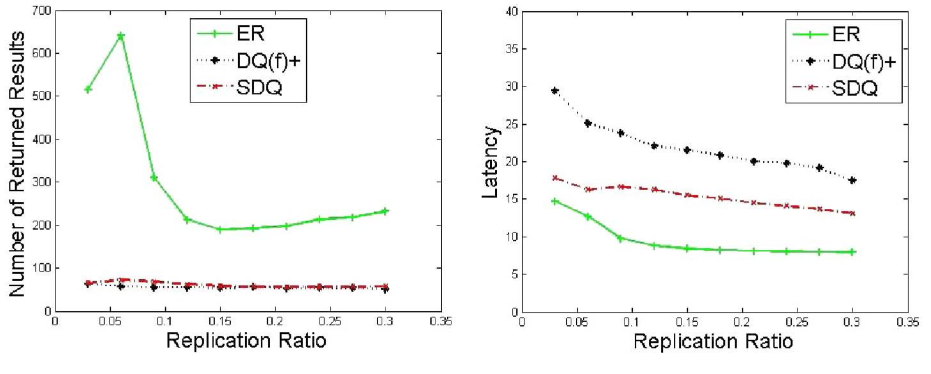


Fig. 2. Performance comparison for high replica ratio impact: (1) number of results and (2) latency.

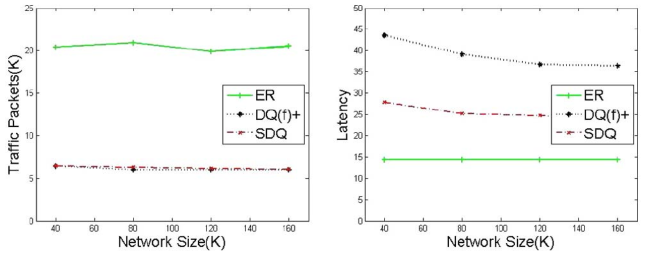


Fig. 3. Performance comparison with a variety of network sizes: (1) number of transmitted packets and (2) latency.

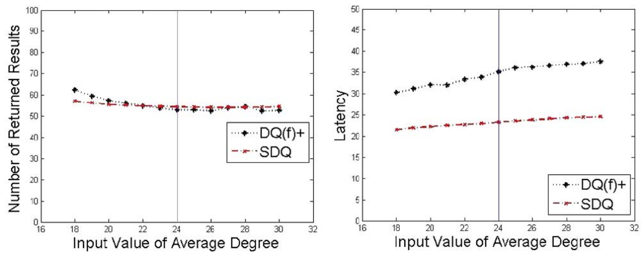


Fig. 5. Performance comparison with deviated average degree estimation: (1) number of transmitted packets and (2) latency.

of Returned Results, and *Required Number of Results,* respectively; we have $O = C/N$. We vary N from 10 to 150 with the step of 20. One hundred fifty is the largest number of requests allowed by a Gnutella ultrapeer [9], [12]. The results are shown in Fig. 4.

When the required number is small, ER shows the high *Overshooting Ratio*. When N is increased, ER becomes stable. Again, SDQ shows a good performance in terms of both latency and traffic cost. In general, the required number of results N has no significant impact for query algorithms.

Impact of deviated average degree estimation. There is a concern from the society: when D, the average degree of the network, is not correctly estimated, would those dynamic query algorithms perform well? In this part, we maintain the network degree to 24, and change the input D to algorithms from 18 to 30. For simplicity, only performances of DQ(f)+ and SDQ are shown in the Fig. 5.

The algorithms are still stable: when the input D is less than the real network average degree, the dynamic algorithms would "consider" this situation as a higher item popularity in the popularity estimation step; on the contrary, when the input D is higher than the real network average degree, this situation can be considered to lower popularity. To sum up, the input D has trivial impact for the dynamic query algorithms.

7 Conclusion

Traffic cost and response latency are two critical metrics for resource query algorithms in unstructured peer-to-peer networks. In this paper, we propose a novel protocol in this paper: SDQ, which calculates an optimal combination of an integer TTL value and a set of neighbors for the next query round. Our experiments demonstrate that SDQ provides a fine-grained control: its latency is close to the well-known minimum one via ER; in the mean time its traffic cost is also small. Through extensive simulations, performances of the controlled-flooding algorithms including SDQ and other existing algorithms are extensively analyzed under a variety of scenarios.

We will further explore SDQ for a wider range of applications, in particular, other unstructured networks such as wireless ad hoc networks and sensor networks.

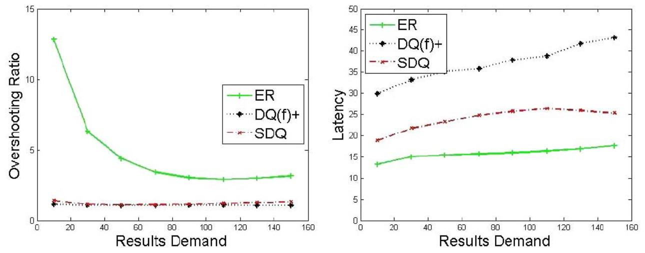


Fig. 4. Performance comparison for required number impact: (1) results overshooting and (2) latency.

Acknowledgments

This work was supported in part through National Natural Science Foundation of China (No. 60803115, No. 60873127, and No. 61073147), National Natural Science Foundation of China—Microsoft Research Asia (No. 60933012), the Fundamental Research Funds for the Central Universities (No. 2011QN014), the Youth Chenguang Project of Wuhan City (No. 201050231080), and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, as well as New Century Excellent Talents in University (No. NCET-10-408), State Education Ministry. An early version of this work appeared as [27]. Hongbo Jiang is the corresponding author.

References

[1] E. Adar and B.A. Huberman, "Free Riding on Gnutella," technical report, Xerox PARC, 2000.

[2] R. Beraldi, "Biased Random Walks in Uniform Wireless Networks," *IEEE Trans. Mobile Computing,* vol. 8, no. 4, pp. 500- 513, Apr. 2009.

[3] H. Cai and J. Wang, "Exploiting Geographical and Temporal Locality to Boost Search Efficiency in Peer-to-Peer Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 17, no. 10, pp. 1189-1203, Oct. 2006.

[4] A.J. Chakravarti, G. Baumgartner, and M. Lauria, "The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network," *IEEE Trans. Systems, Man, and Cybernetics, Part A: Systems and Humans,* vol. 35, no. 3, pp. 373-384, May 2005.

[5] N. Chang and M. Liu, "Revisiting the TTL-Based Controlled Flooding Search: Optimality and Randomization," *Proc. ACM MobiCom,* 2004.

[6] N. Chang and M. Liu, "Optimal Controlled Flooding Search in a Large Wireless Network," *Proc. IEEE Third Int'l Symp. Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt),* 2005.

[7] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-Like P2P Systems Scalable," *Proc. ACM SIGCOMM,* 2003.

[8] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks," *Proc. ACM SIGCOMM,* 2002.

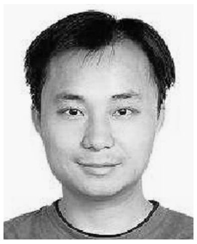
[9] O.S. Community, http://gnutella.wego.com/, 2010.

[10] A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems," *Proc. IEEE 22nd Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.

[11] C. Doulkeridis, A. Vlachou, K. Norvag, Y. Kotidis, and M. Vazirgiannis, "Multidimensional Routing Indices for Efficient Distributed Query Processing," *Proc. Int'l Conf. Information and Knowledge Management,* 2009.

[12] A. Fisk, "Gnutella Dynamic Query Protocol v0.1," http://www9.limewire.com/develop-r/dynamic_query.html, 2003.
[13] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," *Proc. IEEE INFOCOM,* 2004.
[14] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid Search Schemes for Unstructured Peer-to-Peer Networks," *Proc. IEEE INFOCOM,* 2005.
[15] N. Inc, http://www.napster.com/, 2011.
[16] S. Ioannidis, "Absence of Evidence as Evidence of Absence: A Simple Mechanism for Scalable P2P Search," *Proc. IEEE INFOCOM,* 2009.
[17] H. Jiang and S. Jin, "Exploiting Dynamic Querying Like Flooding Techniques in Unstructured Peer-to-Peer Networks," *Proc. IEEE 13th Int'l Conf. Network Protocols (ICNP),* 2005.
[18] S. Jin and H. Jiang, "Novel Approaches to Efficient Flooding Search in Peer-to-Peer Networks," *Computer Networks,* vol. 51, no. 10, pp. 2818-2832, 2007.
[19] Limewire, http://www.limewire.com/, 2011.
[20] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. Int'l Conf. Supercomputing,* 2002.
[21] Q. Lv, S. Ratnasamy, and S. Shenker, "Can Heterogeneity Make Gnutella Scalable," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2002.
[22] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization.* John Wiley, 1988.
[23] C. Palmer and G. Steffan, "Generating Network Topologies that Obey Power Laws," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM),* 2000.
[24] K. Puttaswamy, A. Sala, and B.Y. Zhao, "Searching for Rare Objects Using Index Replication," *Proc. IEEE INFOCOM,* 2008.
[25] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM,* 2001.
[26] D. Stutzbach and R. Rejaie, "Characterizing the Two-Tier Gnutella Topology," *Proc. ACM Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS),* 2005.
[27] C. Tian, H. Jiang, X. Liu, W. Liu, and Y. Wang, "Towards Minimum Traffic Cost and Minimum Response Latency: A Novel Dynamic Query Protocol in Unstructured P2P Networks," *Proc. IEEE 37th Int'l Conf. Parallel Processing (ICPP),* 2008.
[28] B. Waxman, "Routing of Multipoint Connections," *IEEE J. Selected Areas in Comm.,* vol. 6, no. 9, pp. 1617-1622, Dec. 1988.
[29] C.-J. Wu, K.-H. Yang, and J.-M. Ho, "Antsearch: An Ant Search Algorithm in Unstructured Peer-to-Peer Networks," *Proc. IEEE Symp. Computers and Comm.,* 2006.
[30] Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," Technical Report UCB/CSD-01-1141, Computer Science Dept., University of California, 2001.
[31] M. Zhong, K. Shen, and J.I. Seiferas, "The Convergence-Guaranteed Random Walk and Its Applications in Peer-to-Peer Networks," *IEEE Trans. Computers,* vol. 57, no. 5, pp. 619-633, May 2008.
[32] Y. Zhu and Y. Hu, "Enhancing Search Performance on Gnutella-Like P2P Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 17, no. 12, pp. 1482-1495, Dec. 2006.

Chen Tian received the BS, MS, and PhD degrees from the Department of Electronics and Information Engineering at the Huazhong University of Science and Technology, China, in 2000, 2003, and 2008, respectively. He joined the faculty as a lecture in the Department of Electronics and Information Engineering at the Huazhong University of Science and Technology, China. His research interests include distributed networks and wireless networks.



Hongbo Jiang received the BS and MS degrees from Huazhong University of Science and Technology, China. He received the PhD degree from Case Western Reserve University in 2008. After that he joined the faculty of Huazhong University of Science and Technology as an associate professor. His research interests include computer networking, especially algorithms and architectures for high-performance networks, and wireless networks. He is a member of the IEEE.

Xue Liu received the BS and MS degrees in mathematics and in automatic control both from Tsinghua University, China. He received the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 2006. He is an associate professor in the Department of Computer Science and Engineering, University of Nebraska of Lincoln. His research interests include real-time and embedded systems, networked server performance management, and software reliability. He is a member of the IEEE.

Wenyu Liu received the MS and PhD degrees from the Department of Electronics and Information Engineering at the Huazhong University of Science and Technology, China. He is a professor of electronics and information engineering at the Huazhong University of Science and Technology, China. His research interests include image processing, distributed networks, and wireless networks. He is a member of the IEEE.