Unified Framework and an Algorithm for Searching in Pure P2P—Hop-value-based Query-packet Forwarding

Masato Uchida[†] *and Shinya Nogami*

Abstract

In pure peer-to-peer (P2P) file sharing applications and protocols using a flooding-based query algorithm, a large number of control packets (query packets) are transmitted on the network to search for target files. This flooding-based query-packet-forwarding algorithm is clearly not scalable because it will lead to an overhead, such as an overwhelming amount of query traffic and a high CPU load, as more servents (servent = server + client) join the overlay network. To solve such problems, this paper proposes a new query algorithm based on a unified framework that describes a wide variety of query algorithms for pure P2P. This framework determines the number of destinations for query packets based on the hop value recorded in received query packets. Simulation results revealed that the proposed query algorithm can reduce the overhead in the flooding-based query algorithm without decreasing the success rate of retrieval regardless of the density of target files in the network.

1. Introduction

The past few years have seen the development of several new file sharing applications and protocols based on the pure peer-to-peer (P2P) model, which does not have any servers to support clients. In this model, every client acts as a server. These so-called "servents" (servent = server + client) form a decentralized and unstructured application-level overlay network on the physical layer, by connecting to existing servents.

Several pure P2P applications pass messages (packets) that implement file sharing among servents on the overlay network. For example, query packets for target files are broadcast on the overlay network. This flooding-based query-packet-forwarding algorithm (flooding-based query algorithm) is clearly not scalable because it will obviously lead to an overhead, such as an overwhelming amount of query traffic and a high CPU load, as more servents join the overlay network. Against this background, the possibility of multiple random walks (k-random walks) on a pure P2P network to find target files has been discussed [1]. In the k-random walks, a requesting node sends k query packets, and each query packet takes its own random walk on the network. Although another search algorithm has been studied [2], it needs an additional function to cache query results that cannot be obtained by the usual pure P2P protocols.

This paper first provides a unified framework that can describe a wide variety of query algorithms. Based on this framework, we then propose a new query algorithm that can reduce the overhead without decreasing the success rate of retrieval regardless of the density of the target files in the network. Our framework includes the flooding-based query algorithm, which can determine the number of destinations for query packets based on the hop value recorded in received query packets. Note that the hop value is information that the query packet usually has. We evaluated three query algorithms within the framework (flooding, k-random walk, proposed) by simulation, where the algorithms ran on a model drawn from actual topology data accessible to the public [3].

† NTT Service Integration Laboratories
Musashino-shi, 180-8585 Japan
E-mail: uchida.masato@lab.ntt.co.jp

2. Flooding-based query algorithm

Several pure P2P applications, such as Gnutella and its clones, pass query packets that implement file sharing between servents on the overlay network. Specifically, each servent forwards the received query packets to all of its neighbors. Each packet's header contains a time-to-live (TTL) field. TTL is used in the same fashion as in the IP protocol: at each hop its value is decremented until it reaches zero, at which point the packet is discarded. That is, TTL shows how many times the packet will be forwarded by servents before it is removed from the network. Each packet's header also contains a hop field. The hop value is incremented at each hop. That is, the hop value is the number of times the packet has been forwarded.

3. Unified framework for query algorithms

This section first provides a unified framework that allows us to describe a wide variety of query algorithms. Then, we give examples of how algorithms are set up within the framework.

The key idea of the proposed framework is to utilize the hop value *h* recorded in the received query packets when they are forwarded (**Figs. 1** and **2**). That is, a servent forwards the received query packet to some of its neighbors according to *h*, while a servent that uses the flooding-based query algorithm forwards the received query packet to all of its neighbors regardless of *h*.

The proposed framework is outlined in Figs. 1 and 2, where the crossed circle is connected with the circled circle, closed circles, and open circles. When the crossed circle receives a query packet (squared square) whose TTL is *t* and hop value is *h* from the circled circle, it performs the following procedure.

Step 1: If $t = 0$, the crossed circle drops the query packet; otherwise, it determines the number of destinations $N(h)$.

Step 2: If $N(h) \le n$, the crossed circle selects $N(h)$ servents (closed circles) randomly from *n* servents without multiplicity (Fig. 1); otherwise, it selects $N(h)$ servents (closed circles) randomly from the *n* servents with multiplicity (Fig. 2), excluding the servent that transmitted the incoming query packet (circled circle).

Step 3: The crossed circle decrements *t* and increments *h* of the received query packet. Then, it forwards the query packets (crossed squares) to the selected servents.

We can construct various query algorithms based on the above by setting $N(h)$ appropriately. For example, the definition enables us to describe both the flooding-based query algorithm and *k*-random walks within the framework by setting $N(h)$ as follows.

$$N_1(h) = \begin{cases} n & \text{for } h \le d_1 \\ 0 & \text{for } h > d_1 \end{cases}, \quad (d_1\text{: constant}),$$

$$N_2(h) = \begin{cases} k & \text{for } h \le d_2 \\ 1 & \text{for } h > d_2 \end{cases}, \quad (d_2, k\text{: constant}).$$

Obviously, $N_1(h)$ is equivalent to the flooding-based query algorithm, and $N_2(h)$ is equivalent to the *k*-random walks when $d_2 = 0$. This means that the proposed

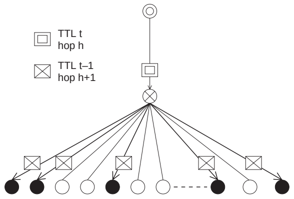

Fig. 1. Query packet forwarding algorithm for $N(h) \le n$. Circle and square objects indicate servents and query packets, respectively.

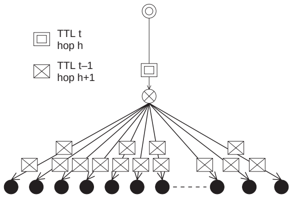

Fig. 2. Query packet forwarding algorithm for $N(h) > n$. Circle and square objects indicate servents and query packets, respectively.

framework is a natural extension of the notion of query algorithms.

Next, we propose a new query algorithm based on the above framework. It is defined as follows.

$$N_3(h) = \begin{cases} n & \text{for } h \le d_3 \\ \left\lceil n^{\frac{1}{1+h-d_3}} \right\rceil & \text{for } h > d_3 \end{cases}, \quad (d_3\text{: constant}).$$

In $N_3(h)$, the number of destinations for query packets that will be forwarded decreases as the hop value grows depending on the nodal degree. There are numerous similar examples to $N_3(h)$. For example,

$$N'(h) = \begin{cases} n & \text{for } h \le d' \\ 1 & \text{for } h > d' \end{cases}, \quad (d'\text{: constant}),$$

$$N''(h) = \begin{cases} n & \text{for } h \le d'' \\ n-(h-d'') & \text{for } h > d'' \end{cases}, \quad (d''\text{: constant}).$$

However, we found heuristically that $N_3(h)$ gives better performance than the examples we tried in simulations. Finding the best form of $N(h)$ theoretically is still an open problem.

4. Simulation

In this section, we consider the appropriate forms of $N(h)$ that can reduce the number of query packets without decreasing the success rate of retrieval. To achieve this, we compared the performance of $N_1(h)$, $N_2(h)$, and $N_3(h)$ through simulation. Though $N(h)$ can take other values, we can obtain a rough approximation by evaluating these forwarding algorithms. We also used real topology data accessible to the public [3]. Statistics about these data are listed in **Table 1**, where the edge is a link between two nodes and the degree of a node is the number of edges that the node has.

4.1 Preparation

We performed the simulation on the proposed framework as follows.

Step 1: Select topology t from crawl1, . . . , crawl6, where the set of servents forming a network is described by C_t.

Step 2: For each topology, place a target file, which will be searched for, on all servents included in C_t with probability (density) p, where the number of placements is described by NumPlace.

Step 3: For each placement, randomly select a servent, which will search for the target file, from C_t, where the number of selections is NumQuery.

Step 4: Transmit the query packets generated by each selected servent, based on the proposed framework in Section 3.

Note that, for each set of topology t and probability p, statistics can be collected NumPlace $\times$ NumQuery times through the above simulation. In this paper, we collected the following statistics concerning both success rate of retrieval and overhead of query algorithm, where $i = 1, . . . ,$ NumPlace $\times$ NumQuery.

$s_{t,p}(i)$: A value describing whether or not target files will be found before the search terminates (i.e., the value is 1 when the target files are found at least once and 0 otherwise).

$g_{t,p}(i)$: Number of (all) generated query packets that each node in the network must process before the search terminates (i.e., the average number of (all) existing query packets in the network before the search terminates).

$v_{t,p}(i)$: Number of visited nodes before the search terminates.

$d_{t,p}(i)$: Number of duplicated query packets that each node in the network must process before the search terminates, where the number of duplicated query packets is defined as $g_{t,p}(i) - v_{t,p}(i)$.

Table 1. Topology data statistics.

Data name	crawl1.log	crawl2.log	crawl3.log	crawl4.log	crawl5.log	crawl6.log
Number of nodes	737	1568	653	435	2282	1406
Number of edges	803	1906	738	459	2765	1583
Average degree	2.1791	2.43112	2.26034	2.11034	2.42331	2.25178
Variance of degree	4.02762	31.1113	49.3197	26.5579	17.2467	8.59948
Maximum degree	18	186	143	64	108	61

Using these statistics, we can define the following criteria, where $|C_t|$ is the number of elements included in the set C_t. We evaluated the simulation results for each set of topology t and probability p using these criteria.

$S_{t,p}$: Probability of finding the target object before the search terminates.

$$S_{t,p} = \frac{\sum_{i=1}^{\mathrm{NumPlace}*\mathrm{NumQuery}} s_{t,p(i)}}{\mathrm{NumPlace}*\mathrm{NumQuery}}$$

$G_{t,p}$: Overhead of an algorithm measured by the average number of generated query packets that each node in the network must process.

$$G_{t,p} = \frac{1}{|C_t|}\frac{\sum_{i=1}^{\mathrm{NumPlace}*\mathrm{NumQuery}} g_{t,p(i)}}{\mathrm{NumPlace}*\mathrm{NumQuery}}$$

$D_{t,p}$: Overhead of an algorithm measured by the average number of duplicated query packets that each node in the network must process.

$$D_{t,p} = \frac{1}{|C_t|}\frac{\sum_{i=1}^{\mathrm{NumPlace}*\mathrm{NumQuery}} d_{t,p(i)}}{\mathrm{NumPlace}*\mathrm{NumQuery}}$$

4.2 Results and discussion

The simulation results are shown in **Figs. 3–6**, where the parameter values used for these simulations are listed in **Table 2**. The results are summarized in **Table 3**. Although these figures are for crawl5, the results were almost the same even when other topology data was used.

We measured the overheads of algorithms by $G_{t,p}$ and $D_{t,p}$. Note that the average number of generated query packets per node ($G_{t,p}$) and the average number of duplicated query packets per node ($D_{t,p}$) increased as the values of d_1, k, and d_3 rose.

As we can see from **Fig. 3**, which is for $p = 0.05$, $N_2(h)$ and $N_3(h)$ found target files with higher probability than $N_1(h)$, when $N_1(h)$, $N_2(h)$, and $N_3(h)$ used the same number of generated query packets. Additionally, we can see from **Fig. 4**, which is for $p = 0.01$, that $N_1(h)$ and $N_3(h)$ found the target files with higher probability than $N_2(h)$, when $N_1(h)$, $N_2(h)$, and $N_3(h)$ used the same number of generated query packets. This means that $N_3(h)$ found the target files with a smaller number of generated query packets regardless of the value of p, although $N_1(h)$ found target files with a smaller number of generated query packets only when p was low, and $N_2(h)$ found target files with a smaller number of generated query packets only when p was high.

Moreover, as we can see from **Fig. 5**, which is for $p = 0.05$, $N_3(h)$ found target files with higher probability than $N_1(h)$ and $N_2(h)$, when $N_1(h)$, $N_2(h)$, and N_3(h) used the same number of duplicated query packets. Additionally, as we can see from **Fig. 6**, which is for $p = 0.01$, $N_1(h)$ and $N_3(h)$ found the target files with higher probability than $N_2(h)$, when $N_1(h)$, $N_2(h)$, and $N_3(h)$ used the same number of duplicated query packets. This means that $N_3(h)$

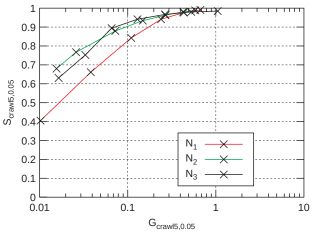

Fig. 3. Success rate of retrieval ($S_{\mathrm{crawl5,0.05}}$) versus the average number of generated query packets per node ($G_{\mathrm{crawl5,0.05}}$). The horizontal axis is a log scale.

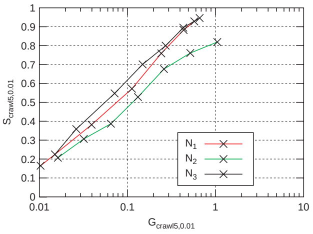

Fig. 4. Success rate of retrieval ($S_{\mathrm{crawl5,0.01}}$) versus the average number of generated query packets per node ($G_{\mathrm{crawl5,0.01}}$). The horizontal axis is a log scale.

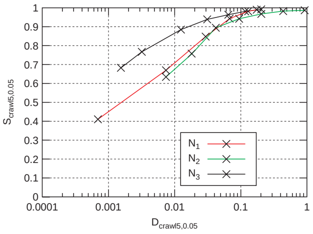

Fig. 5. Success rate of retrieval ($S_{crawl5,0.05}$) versus the average number of duplicated query packets per node ($D_{crawl5,0.05}$). The horizontal axis is a log scale.

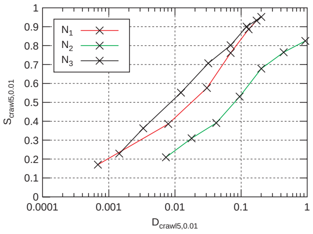

Fig. 6. Success rate of retrieval ($S_{crawl5,0.01}$) versus the average number of duplicated query packets per node ($D_{crawl5,0.01}$). The horizontal axis is a log scale.

Table 2. Parameter values.

NumPlace	20
NumQuery	200
TTL (0)*	7
p	0.01, 0.05
d_1	0, 1, 2, 3, 4, 5, 6
d_2	0
d_3	0, 1, 2, 3, 4, 5, 6, 7
k	10, 20, 40, 80, 160, 320, 640

* The initial value of TTL defined by requesting servent

Table 3. Results of simulations.

		N_1	N_2	N_3
$G_{t,p}$	$p = 0.01$	good	bad	good
	$p = 0.05$	bad	good	good
$D_{t,p}$	$p = 0.01$	good	bad	good
	$p = 0.05$	bad	bad	good

found target files with a smaller number of duplicated query packets regardless of the value of p, although $N_1(h)$ found the target files with a smaller number of generated query packets only when p was low, and $N_2(h)$ found target files with a larger number of duplicated query packets regardless of the value of p. This is because if the query algorithm based on $N_2(h)$ is used, the requesting servent's neighbors must transmit numerous redundant query packets when k is high. We therefore conclude that $N_3(h)$ can find target files with fewer query packets without duplication, regardless of the value of p, if the values of the parameters listed in Table 2 are appropriately adjusted, although $N_1(h)$ can find target files with fewer query packets without duplication only when p is low, and $N_2(h)$ can find target files with fewer query packets by duplication only when p is high.

5. Conclusion and further study

This paper proposed a new query algorithm based on a unified framework which describes a wide variety of query algorithms for pure P2P such as the flooding-based query algorithm and the k-random walks. The proposed query algorithm can reduce the overhead without decreasing the success rate of retrieval regardless of the density of the target files in the network, where our framework can determine the number of destinations for query packets based on the hop value recorded in received query packets. More specifically, simulation study found that the new query algorithm is a better query algorithm than the flooding-based query algorithm and the k-random walks. Based on the above, we can conclude that the number of servents to which the query packet will be forwarded should be small as the hop value increases like $N_3(h)$. The results in this paper are useful for designing a scalable query algorithm for pure P2P.

References

[1] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and Replication in Unstructured Peer-to-Peer Networks,” Proceedings of the 16th Annual ACM International Conference on Supercomputing, New York City, U.S.A., June 2002.
[2] K. Sripanidkulchai, “The Popularity of Gnutella Queries and Its Implications on Scalability,” http://www-2.cs.cmu.edu/kunwadee/research/p2p/gnutella.html.
[3] http://crawler.limewire.org/data.html

Masato Uchida
Communication Traffic & Service Quality Project, NTT Service Integration Laboratories.
He received the B.E. and M.E. degrees in information engineering from Hokkaido University, Sapporo, Hokkaido in 1999 and 2001, respectively. In 2001, he joined NTT Service Integration Laboratories, Tokyo, Japan. He is a member of the Institute of Electronics, Information and Communication Engineers of Japan (IEICE). He received the research award (IEICE Communication Quality Technical Group) in 2003 and the young investigators’ award (IEICE) in 2004.

Shinya Nogami
Senior Research Engineer, Communication Traffic & Service Quality Project, NTT Service Integration Laboratories.
He received the B.E., M.E., and D.Eng. degrees in electrical and communications engineering from Tohoku University, Sendai in 1979, 1981, and 1984, respectively. In 1984, he joined Nippon Telegraph and Telephone Public Corporation (now NTT) where he has been researching traffic design and the performance evaluation of information communications and switching systems. He is a member of the Communications Society of the IEEE and the Information Processing Society of Japan, and he has been an associate editor of IEICE Trans. on Communications since 2003.