

PAPER

An Efficient Index Dissemination in Unstructured Peer-to-Peer Networks

Yusuke TAKAHASHI^{†a)}, *Nonmember*, Taisuke IZUMI^{††b)}, *Member*, Hirotsugu KAKUGAWA^{†c)}, *Nonmember*,
and Toshimitsu MASUZAWA^{†d)}, *Member*

SUMMARY Using Bloom filters is one of the most popular and efficient lookup methods in P2P networks. A Bloom filter is a representation of data item indices, which achieves small memory requirement by allowing one-sided errors (false positive). In the lookup scheme based on the Bloom filter, each peer disseminates a Bloom filter representing indices of the data items it owns in advance. Using the information of disseminated Bloom filters as a clue, each query can find a short path to its destination. In this paper, we propose an efficient extension of the Bloom filter, called a Deterministic Decay Bloom Filter (DDBF) and an index dissemination method based on it. While the index dissemination based on a standard Bloom filter suffers performance degradation by containing information of too many data items when its dissemination radius is large, the DDBF can circumvent such degradation by limiting information according to the distance between the filter holder and the items holders, i.e., a DDBF contains less information for faraway items and more information for nearby items. Interestingly, the construction of DDBFs requires no extra cost above that of standard filters. We also show by simulation that our method can achieve better lookup performance than existing ones.

key words: Bloom filter, unstructured P2P networks, lookup problem, index dissemination, query routing

1. Introduction

1.1 Background

Recently, peer-to-peer (P2P) systems have been occupying an important position of distributed computing. In contrast to the traditional server-client architecture, participants in P2P networks (called peers) communicate with each other in fully decentralized manner. P2P systems have distinct advantages in load balancing, scalability, fault tolerance and so on. Thus several applications based on the P2P paradigm are proposed. One typical example is a file-sharing system, where each peer shares a large amount of files by providing a part of its storage area. In P2P file-sharing systems, data items (e.g., image files, program sources and so on) are distributed over many peers, and thus a peer needs to lookup peers storing the target item it wants to

obtain (called a *lookup problem*). Generally, P2P lookup mechanisms are classified into three categories: centralized, decentralized/structured and decentralized/unstructured. In centralized schemes, all indices of existing data items are managed by central servers. Each peer can obtain the location of the target item by sending a query to the central servers [1]. Conversely, decentralized schemes need no central servers but manage indices in some distributed manner. Decentralized/structured schemes maintain the logical network topology on peers to enable efficient lookup [2]–[5]. While structured systems achieve smaller lookup cost (e.g., time and the number of messages), topology maintenance cost is imposed. This cost becomes considerably large, especially in highly dynamical systems where many peers frequently join and leave. In the decentralized/unstructured approach, the network topology allowed to be arbitrary. Although this approach takes only small maintenance cost, it can use no structural properties for efficient lookup. Thus, lookup is done by naive methods such as query flooding or random walking. Because of its simplicity, the decentralized/unstructured approach is adopted by many real P2P systems [6], [7]. However, the lookup performance of this approach is usually inefficient, and thus its improvement has been actively studied by many researchers [8]–[11].

A Bloom-filter-based lookup method is one of the most successful approaches for improving lookup performance [12], [13]. A Bloom filter is a compact data structure representing a set of data indices with one-sided errors (false positive). In Bloom-filter-based search methods, the Bloom filters are used as navigation information for query routing. Then, a peer receiving a query forwards it to the most likely neighbor by referring the Bloom filters of its neighbors. In those methods, each peer can obtain more useful hints for query routing by enlarging the filter's dissemination radius, and thus the lookup performance is expected to improve. However, too large dissemination makes each peer gather too many filters. It contradicts small memory requirement, which is an advantage of Bloom filters. To avoid managing too many filters, the gathered filters are ordinary combined into a smaller number of filters by bitwise-OR operations. However, since such combination brings too many indices into one filter, its probability becomes higher, and this causes performance degradation of lookup. Several variants of the Bloom filter have been proposed to resolve this problem [14], [15].

Manuscript received November 7, 2007.

Manuscript revised February 21, 2008.

[†]The authors are with the Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560-8531 Japan.

^{††}The author is with the Graduate School of Engineering, Nagoya Institute of Technology, Nagoya-shi, 466-8555 Japan.

a) E-mail: yusuke-t@ist.osaka-u.ac.jp

b) E-mail: t-izumi@nitech.ac.jp

c) E-mail: kakugawa@ist.osaka-u.ac.jp

d) E-mail: masuzawa@ist.osaka-u.ac.jp

DOI: 10.1093/ietisy/e91-d.7.1971

1.2 Contribution of Our Paper

The aim of this paper is to propose an efficient index dissemination based on an extension of the Bloom filter, called a Deterministic Decay Bloom Filter (DDBF). The DDBF is based on a similar strategy with existing methods such as Attenuated Bloom Filter (ABF) [14] and Exponentially Decay Bloom Filter (EDBF) [15]. However, the DDBF-based lookup scheme can achieve both small memory requirement and better performance with no additional communication cost.

In (standard) Bloom filters, several hash functions are used to encode a set X of data items to one filter F . While standard filters use a same set of hash function for all items in X , the DDBF can use different numbers of hash functions at different peers to encode a data item. More precisely, the key idea of DDBF is to decrease deterministically the number of hash functions used in the Bloom filter construction according to the distance between a filter owner and the data item holders. Using this technique, we can avoid the performance degradation caused by wide-area dissemination of filters. We also show by simulation that our method achieves better lookup performance than existing ones. The evaluation result shows that the DDBF technique much improves the performance of lookup.

1.3 Related Works

As mentioned in Sect. 1.1, there are many works about P2P lookup mechanisms. One of directions in those works is the study about Distributed Hash Table, such as Chord [4], CAN [2], Pastry [3] and Tapestry [5]. Another direction is the lookup mechanism in unstructured peer-to-peer systems. The method based on Bloom filters is a successful approach. This method has been proposed earlier in various contexts [12]–[19]. The work in the Gnutella [6] development community [18], [19] has focused on specifying and implementing a protocol for constructing and maintaining Bloom-filter-based query routing. As variations of Bloom-filter-based lookup, Attenuated Bloom filter [14] and Exponentially Decay Bloom Filter [15] are proposed, which aim to improve efficiency of index dissemination and lookup performance. Handling the popularity of items is also actively considered in previous works. Square-root replication [8], [10] and their variants [9], [11] are successful approaches for handling the popularity. They are classified into Index-free lookup method, which does not use index of each item to reduce the communication cost. In addition, these methods follow the square-root principle, which realizes optimal solution of the expected number of hops for an arbitrary query to succeed.

1.4 Organization

The paper is organized as follows: In Sect. 2, we present the system model and miscellaneous definitions. We introduce

the DDBF and the search method based on it in Sect. 3. Section 4 shows the comparative results of some Bloom-filter-based methods. A brief discussion on our methods is given in Sect. 5. Finally we conclude this paper in Sect. 6.

2. Preliminaries

2.1 Peer-to-Peer Network and Lookup Problem

In this paper, we consider a P2P network with arbitrary topology. A network is represented by a graph $G = (V, E)$, where V is the set of peers and E is the set of communication links. Two peers can directly communicate with each other by exchanging messages if they are connected by a link. A link between two peers v and w is denoted by (v, w) . In what follows, we use terms of graph theory for networks with no explicit explanations. Throughout this paper, we assume static networks for ease of explanations. Note that our method can be easily modified to work correctly on the systems where a set of peers and links in P2P networks change dynamically.

2.2 Bloom Filter

A Bloom filter is a simple space-efficient data structure representing a set of some elements. Let X be a (possibly infinite) universal set of elements. A Bloom filter $F(X)$ representing a set of elements $X \subseteq X$ provides operation $find(x)$, which tests the membership of x for X . Let $F.find(x)$ be the operation $find(x)$ for the Bloom filter F . This operation allows one-sided errors (false positive), that is, if $x \in X$, then $F.find(x)$ necessarily returns true, however, even if $x \notin X$, it does not necessarily return false.

In the following, we describe how to construct the Bloom filter $F(X)$ which represents a set of n elements $X = \{x_1, x_2, \dots, x_n\}$. A Bloom filter F consists of an array of m bits[†]. Let $F[1], F[2], \dots, F[m]$ be each bit of the array. A Bloom filter uses some hash functions $h_i (1 \leq i \leq k)$ such that $h_i : X \rightarrow [\infty, \Phi]$. The Bloom filter for X is an assignment $F[j] = 1$ iff $j = h_i(x)$ for some $i (1 \leq i \leq k)$ and $x \in X$. The operation $F.find(y)$ returns the boolean value $F[h_1(y)] \wedge F[h_2(y)] \wedge \dots \wedge F[h_k(y)]$. If y actually belongs to X , the result of $F.find(y)$ operation is necessarily true. On the other hand, even if $y \notin X$, the operation $F.find(y)$ may return true. In general, under the assumption of uniformly-random hash functions, the probability that $F.find(y)$ returns true in spite of $y \notin X$ (error probability) is

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k = (1 - p)^k, \quad (1)$$

where $p = e^{-kn/m}$ is the probability that each bit in the array is set to 1. This expression implies that the filters containing too many bits of 1 have higher error probability. To guarantee low error probability, we have to (1) decrease the

[†]Initially, all entries in the array are 0.

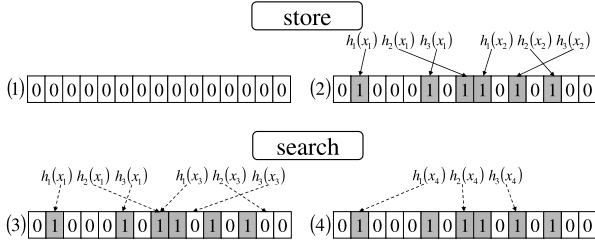


Fig. 1 An example of a Bloom filter.

number of hash functions k or (2) increase the size of the Bloom filter. However, in the case (1), we can see from the expression that too small k also causes high error probability. On the other hand, in the case of (2), we cannot achieve space-efficiency. Therefore, it is important to configure an appropriate combination of the size of filter m and the number of hash functions k .

Figure 1 shows an example of a Bloom filter. The indices set X consists of x_1 and x_2 , and hash functions are h_1, h_2 and h_3 . (1) The filter begins as an array of all 0s, and (2) each index in X is hashed by all the hash functions and the corresponding bits are set to 1 (In this figure, $h_1(x_1) = 2, h_2(x_1) = 8, \dots$ and so on). (3) To test the membership of an index, all the entries corresponding to its hashed values are checked. In the example, for x_1 , true is returned. On the other hand, for x_3 , false is returned, and the filter does not contain it. (4) However, in spite of $x_4 \notin X$, all of its corresponding bits are set to 1. Then the filter returns true, which is a wrong result. That is, Bloom filters may cause false positive. In general, filters containing too many indices for their size have high error probability.

2.3 Bloom-Filter-Based Lookup

In this subsection, we explain the lookup scheme using standard Bloom filters. In advance, each peer constructs a Bloom filter representing a set of the data items X it owned. Then, the constructed Bloom filter is disseminated to nearby peers. Each peer maintains one filter for one link. The filter associated with a link stores the union of all filters that were received via the link, where the union of two filters is the filter obtained by bitwise-OR operation of them. As a result, each peer v holds one filter for each neighbor w representing the set of data items owned by peers within dissemination radius from v via link (v, w) . In lookup of an item x , the searcher forwards a query for an item x to someone holding x . The query holder tests the membership of x for all filters it has. If a filter F returns true, the query is forwarded to the neighbor corresponding to F .

3. Design of DDBF: Deterministic Decay Bloom Filter

In Bloom-filter-based lookup methods, if the dissemination radius becomes large, the number of bits 1 in a combined filter rapidly grows (in typical topologies of P2P networks, its growth is exponential). Then, Bloom filters suffer the prob-

lem of high error probability as mentioned in Sect. 2. In this section, we propose Deterministic Decay Bloom Filter (DDBF), which is a variant of standard Bloom filters modified to avoid this problem. We also propose a search method based on the DDBF.

3.1 Overview of DDBF

While the number of hash functions used for the construction of standard Bloom filters is fixed for any peer and any item, the number of hash functions for DDBF depends on the distance from the filter owner and the item holder, i.e., it uses a larger number of hash functions for nearby data items and a smaller number of hash functions for faraway data items.

In the following, we introduce the formal definition of DDBF. Same as the standard one, DDBF F also consists of a bit array of size m and k hash functions. Each peer has one DDBF for each neighbor: the DDBF that a peer v owns for its neighbor w is denoted by F_{vw} . The dissemination radius of DDBFs is denoted by D . We consider a family $\mathcal{H} = \{\mathcal{H}_\infty, \mathcal{H}_\infty, \dots, \mathcal{H}_D\}$ of D sets of hash functions $H_1 = \{h_1, \dots, h_{k_1}\} (k_1 = k), H_2 = \{h_1, \dots, h_{k_2}\}, \dots, H_D = \{h_1, \dots, h_{k_D}\}$ satisfying $k_1 \geq k_2 \geq \dots \geq k_D$ and $H_1 \supseteq H_2 \supseteq \dots \supseteq H_D$. Letting v be a peer that owns a set of data items X , we define v 's hashed information $F(v, H)$ using a set of hash functions H as the (standard) Bloom filter for X constructed by using a set of hash functions H . Let $N(v, w, i)$ be the set of peers that are i hops away from w in the network $(V, E - (v, w))$. Then, DDBF F_{vw} for a family of hash functions H is defined as follows:

$$F_{vw} = \bigvee_{i=1}^D \left(\bigvee_{u \in N(v, w, i-1)} F(u, H_i) \right), \quad (2)$$

where $F_1 \vee F_2$ is the filter obtained by bitwise-OR operation of F_1 and F_2 . Intuitively, DDBF F_{vw} contains the hashed information of peers within D hops away from v via link (v, w) , where the hashed information of peers i ($1 \leq i \leq D$) hops away is constructed by using the set of hash functions H_i .

Figure 2 shows an example of the DDBF using four hash functions h_1, \dots, h_4 , where $H_1 = \{h_1, h_2, h_3, h_4\}$, $H_2 = \{h_1, h_2\}$ and $H_3 = \{h_1\}$. Peer v has a DDBF F containing indices of items owned by w, u and t . Then, the index of the item owned by w , which is one hop away from v , is contained in F by using all four hash functions (H_1), and for those owned by u and t , two (H_2) and one (H_3) hash functions are used respectively.

As with standard Bloom filters, DDBF also provides procedure $find(x)$ to test the membership of a data item x . However, while the standard one returns a boolean value, DDBF returns an integer value. Letting $F[H_i(x)] = F[h_1(x)] \wedge F[h_2(x)] \wedge \dots \wedge F[h_{k_i}(x)]$, the resulting value of DDBF for a family of the set of hash functions $H = \{H_1, H_2, \dots, H_D\}$ is the minimum i satisfying $F[H_i(x)] = 1$. If $F[H_i(x)] = 0$ for all $1 \leq i \leq D$, $find(x)$ returns ∞ . The

intuition of the returned value i is that the target item is expected to be located in a peer i hops away. If a peer i hops away from the peer holding a DDBF F has the target item x , it is guaranteed that $F.find(x)$ returns the value less than or equal to i . However, since DDBFs also allow one-sided errors, the fact of $F.find(x) = i$ does not necessarily imply that the item x is located at a peer i hops away.

3.2 Construction of DDBF

3.2.1 Algorithm

As we mentioned in Sect. 3.1, each peer v constructs a DDBF F_{vw} for each neighbor w . The DDBF F_{vw} is obtained from the hashed information $F(u, H_i)$ of every peer u that is i ($1 \leq i \leq D$) hops away from v via link (v, w) . In other words, every peer u needs to send $F(u, H_i)$ to peers i hops away from u . Our construction of DDBFs is to send each hashed information $F(u, H_1), F(u, H_2), \dots, F(u, H_D)$ separately in time-division manner to the corresponding peers.

To simplify protocol description, we assume that the system has round-based synchrony: In every round, each peer (1) sends a message to each neighboring peer, (2) receives messages from neighboring peers, and (3) computes

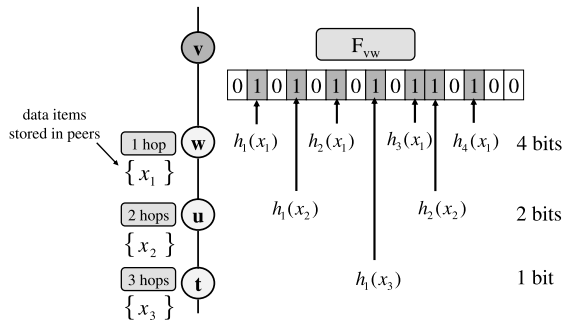


Fig. 2 An example of the DDBF.

locally. It is ensured that the messages sent at some round are necessarily received within the same round. In the following, we explain the construction of DDBF F_{vw} in detail:

- At round one, each peer v sends $F(v, H_D)$ to all the neighbors. Then, in the same round, it receives $F(u, H_D)$ from each neighbor u .
- At round r ($r \geq 2$), each peer v first constructs a filter for each neighbor u . It combines $F(v, H_{D-r+1})$ and all filters received from all neighbors except for u in round $r - 1$ by bitwise-OR operation. The resulting filter is sent to u at round r .

Repeating this task until round D , v can obtain the DDBF F_{vw} for its neighbor w : F_{vw} is the filter received from w at round D .

Figure 3 shows an example of constructing a DDBF F_{vw} . Letting dissemination radius $D = 3$, at round one, w sends $F(w, H_3)$ to v and receives $F(x_1, H_3)$ from x_1 . At round two, w combines $F(w, H_2)$ and all filters received from all neighbors except for v in round one by bitwise-OR operation. Then, w sends the resulting filter to v and receives $F(x_1, H_2) \vee F(x_2, H_3)$ from x_1 . At round three, w sends the combined filter $F(x_2, H_3) \vee F(x_1, H_2) \vee F(w, H_1)$ to v .

Notice that the round-based synchrony is easily realized in asynchronous systems using synchronizer α [20] without any additional communication cost.

3.2.2 Reducing the Communication Cost

Such a synchronous construction scheme enables each peer to send only one filter to each of its neighbors in one round. This implies no additional communication cost compared with existing methods. In addition, we can apply the delta compression technique to further reduce communication cost [15]. When peers send the update information, they send the differential filter obtained by XOR operation to the current filter with the previous one. Then, we can further

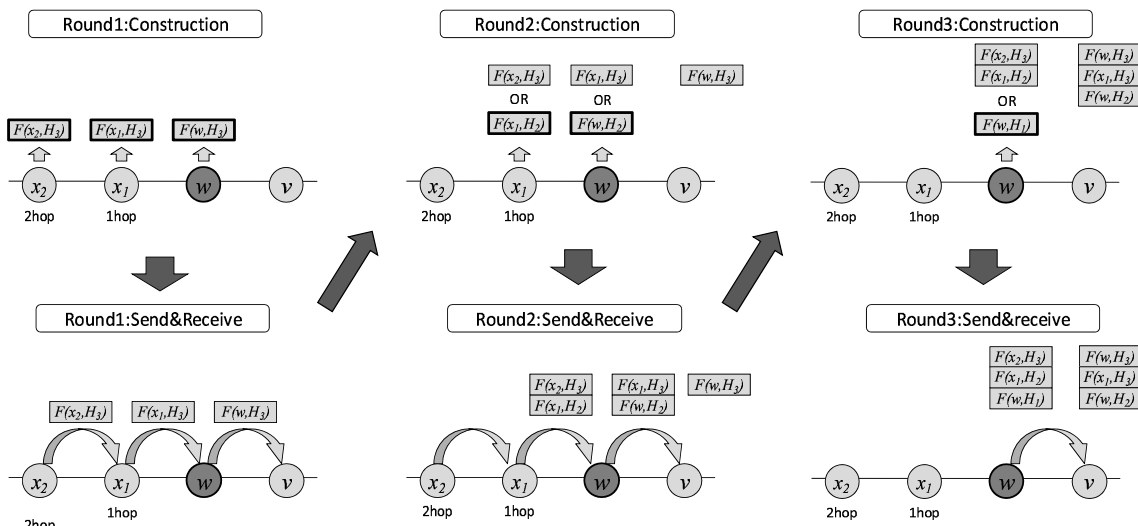


Fig. 3 An example of constructing the DDBF F_{vw} .

reduce the communication cost by compressing the differential filter by the arithmetic coding [21]. Letting the size of filters before compressing be m , and the fraction of bits set to 1 be λ , the size of the hashed information after compression is $mH(\lambda)$, where $H(\lambda)$ is an entropy function given by $H(\lambda) = -\lambda \log_2 \lambda - (1 - \lambda) \log_2 1 - \lambda$. $H(\lambda)$ is an upper convex function, and the maximal value of $H(\lambda)$ is 1 when $\lambda = 0.5$ and the minimum value of $H(\lambda)$ is 0 when $\lambda = 0$ or 1. Thus, the arithmetic coding produces no effect when $\lambda = 0.5$ and larger effects as λ approaches to 0 or 1. Generally, differential filter contains few bits set to 1 (i.e., λ is very small). Therefore, the communication cost can be substantially reduced by this technique.

To further enhance compression effect, we can apply the following technique: In construction of DDBF, at round r ($r \leq 2$), each peer v first constructs a filter by combining $F(v, H_{D-r+1})$ and filters received in round $r - 1$ by bitwise-OR operation. To enhance compression effect we can combine received filters and the differential hashed information $F(v, H_{D-r+1} - H_{D-r+2})$. Then, each peer further compresses the resulting filter by arithmetic coding. Figure 4 shows an example of this technique. Let $D = 3$, $H_1 = \{h_1, h_2, h_3, h_4\}$, $H_2 = \{h_1, h_2\}$ and $H_3 = \{h_1\}$. At round one, each peer sends $F(w, H_3)$ to their neighbors. And at round two, it constructs a filter by combining $F(w, H_2) - F(w, H_3)$ and the received filters, and sends the resulting one. At round three, it sends the combination of $F(w, H_1) - F(w, H_2)$ and the received filters. Using this technique, the transmitted filter contains fewer bits set to 1, and thus we can enhance the effectiveness of arithmetic coding.

3.3 Query Forwarding Based on DDBF

In this subsection, we describe how to forward queries using DDBFs. The query forwarding method is fundamentally same as one by Kumar et al. [15].

DDBFs are used as the routing tables in forwarding queries, i.e., when a query for a data item x is received by a peer v , it is forwarded to the neighbor w such that the value $F_{vw}.find(x)$ is smallest among all neighbors that the query has not visited yet. If there exists two or more neighbors having the smallest value, one of them is selected as the next peer uniformly at random. If there exist no unvisited neighbors, the next peer is also determined by uniform random choice from all neighbors. To identify unvisited neighbors, IDs of visited peers are attached to the query. In addition,

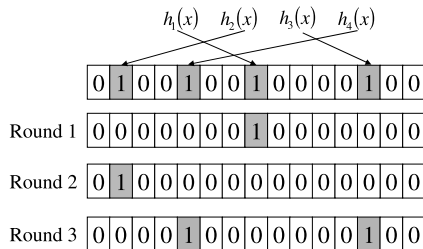


Fig. 4 An example of DDBF construction to reduce communication cost.

each query has TTL (time-to-live), which represents the life-time of the query and is decremented by one when it is forwarded to a neighbor. If the TTL of a query becomes zero, then this query is regarded as failure and discarded.

4. Evaluations

In this section, we evaluate the DDBF-based method. DDBF allows one-sided error, just as the method using standard Bloom filter and the high error probability degenerates the lookup performance. Basically, this probability depends on the density of filters, which is the fraction of bits set to 1 (the large density causes high error probability). The density of filters are affected by many factors, for example, structure of networks (the network topology, the number of peers and these degrees and so on), the size of filters, the number of hash functions, the dissemination radius and so on. While we need to set each parameter properly to keep down the density of filters, it is not easy to analytically optimize the value of each parameter. Consequently, we evaluate the lookup performance of our method by simulation on several situations.

In running the simulation, we compare our method with other lookup methods based on the Bloom filter and its variants. We compare the following four methods; (1) The DDBF-based method (called DDBF method), (2) The method using Exponential Decay Bloom Filter [15] (called EDBF method), (3) The method using Attenuated Bloom Filter [14] (called ABF method) and (4) The method using standard Bloom filter (called SBF method). In the EDBF method, each peer maintains one EDBF for each link, which is constructed as follows: First, each peer v sends the standard Bloom filter to its neighbors. When a neighbor w receives a filter, it resets each bit of the received filter to 0 with probability $1 - 1/d$, where d is a design parameter of EDBF (called decay factor). In the ABF method, D filters are assigned to one link. The i -th filter for a link represents a set of data items stored in peers i hops away from the filter holder (notice that different from DDBF, the number of hash functions used for all filters is same.).

Before the simulation, we compare the costs of the filter constructions for these four methods (no compression case). Table 1 presents the result, where the size of filter per one link and the dissemination radius is denoted by m and D respectively. In the ABF method, each filter consists of a m bits array of D normal Bloom filters. Thus, the memory and communication requirements are D times as those of others. The DDBF method and the EDBF method can construct each filter with no additional costs compared to

Table 1 The cost of filter construction.

| Method | Communication cost | Local memory requirement |
|--------|--------------------|--------------------------|
| DDBF | $m \cdot D$ | m |
| EDBF | $m \cdot D$ | m |
| ABF | $m \cdot D^2$ | $m \cdot D$ |
| SBF | $m \cdot D$ | m |

the SBF method.

4.1 Simulation Setting

In our simulation, we use two network topologies; flat topology and hierarchical topology. A flat topology is constructed by connecting any pair of peers with uniform probability. A hierarchical topology consists of two-tier peers, called ultra-peers (in the top-level layer) and leaf peers (in the bottom-level layer). Usually, Bloom-filter-based lookup methods assume nothing about network topology because their primary advantage is that they can be adopted on any network topology. Thus, we use these two topologies as two typical cases, flat topology as almost uniform one and hierarchical topology as biased one. Note that this classification of the simulation topology is the same as the simulation in the paper of the EDBF method [15].

The flat topology used in the simulation consists of 15000 peers and the average degree is 6 (i.e., each pair is connected with probability $6/15000$). Then, the diameter of the generated network becomes 8. The hierarchical topology used in the simulation consists of 1500 ultra-peers and 13500 leaf peers. The topology organized by ultra-peers is flat (the average degree set to 15). each leaf peer is connected to 2-3 ultra-peers randomly, and there are no links between leaf peers. The diameter of the generated network becomes 6. Note that the parameters of the hierarchical topology (the configuration of network, the fraction of ultra-peers and so on) is based on modern P2P file-sharing systems like Gnutella V0.6 [22]. We also assume the popularity of data items, where the target item of each query is probabilistically decided depending on the popularity distribution over all items. In our simulation, we consider two distributions, uniform distribution and Zipf-like distribution. In the Zips-distribution the search frequency of the i -th most popular item is proportional to $1/i^\alpha$. Note that the higher the value of α is, the larger the bias of popularity is. According to [23], the value of α in Gnutella is between 0.6 and 1.2. Thus, we set $\alpha = 1.0$. We experiment the following four cases; (1) flat topology and uniform distribution, (2) flat topology and Zipf-like distribution, (3) hierarchical topology and uniform distribution and (4) hierarchical topology and Zipf-like distribution. Here, we show the results of (1) and (4), that are two contrasting cases of them (the results of (2) and (3) have a tendency similar to those of (1) and (4) respectively). In all simulation scenarios, we generate 10000 queries. Each query is injected by a randomly selected peer and the TTL is set to 500. The size of each filter is set to 2^{15} bits in all the methods of our simulation. Exceptionally, in the ABF method, each link has an array of D filters with size $2^{15}/D$ bits to keep the communication and memory requirement equal to the other methods. This filter size is empirically configured by trying preliminary simulation a number of times: The appropriate setting of the filter size is a very important factor. If the filter size is too large (or too small), the simulation shows almost no difference because every method achieves good (or poor) performance.

Table 2 The setting in flat topology.

| Method | #hash functions | Name |
|--------|-------------------------|------|
| DDBF | $ H_k = 16(1/2)^{k-1}$ | D1 |
| | $ H_k = 32(1/2)^{k-1}$ | D2 |
| EDBF | $k = 16, d = 2$ | E1 |
| | $k = 32, d = 2$ | E2 |
| ABF | $k = 3$ | A1 |
| | $k = 4$ | A2 |
| SBF | $k = 3$ | S1 |
| | $k = 4$ | S2 |

Table 3 The setting in hierarchical topology.

| Method | #hash functions | Name |
|--------|-------------------------|------|
| DDBF | $ H_k = 16(1/2)^{k-1}$ | D3 |
| | $ H_k = 32(1/2)^{k-1}$ | D4 |
| EDBF | $k = 16, d = 2$ | E3 |
| | $k = 32, d = 2$ | E4 |
| ABF | $k = 3$ | A3 |
| | $k = 4$ | A4 |
| SBF | $k = 3$ | S3 |
| | $k = 4$ | S4 |

The filter of 2^{15} bits presents the difference clearly on our simulation setting.

In both topologies, we generate three different numbers of data items, 30000, 45000 and 60000, and deploy them to peers in proportion to their degrees. We assume that each data item is different and no item has its replications. Besides, we consider the dissemination radius of 3, 4 and 5 in the flat topology and 2 and 3 in the hierarchical topology. We omit the cases of the dissemination radius more than 5 in the flat topology and more than 3 in hierarchical topology because for such cases every method shows drastic performance degradation due to the high density of filters. Throughout the simulation, we use the salted MD5 algorithm as hash functions. Each method uses two decay patterns for the number of hash functions. These patterns combined with dissemination radii are shown in Tables 2 and 3. The first character of each name represents the method (e.g. the DDBF method is “D”).

We evaluate the four methods by the following measurements: (1) the density of filters (in the ABF method, we show the density for both of 1st and D -th filters), (2) the average number of hops taken until queries reach the target and (3) the success ratio of lookups (i.e., the fraction of the queries that can find their targets until their TTLs become zero). Throughout the simulations, the value of TTL is set to 500.

4.2 Simulation Results

Table 4 presents the results on the flat topology. Among the three measurements, the average number of hops most directly shows lookup performance. Thus, we also show Figure 5 representing the results of the average number of hops to clarify the performance difference. Totally, every method increase their performances with the dissemination radius but the performance of the EDBF gets relatively worse. It is

Table 4 The result in flat topology.

| #generated items = 30000 | | | | | | | | | |
|--------------------------|-----------------------------|--------------|-------------|-------|--------------|--------|--------|-------|--------|
| Dissemination Radius | Evaluation items | Pattern | | | | | | | |
| | | D1 | D2 | E1 | E2 | A1 | A2 | S1 | S2 |
| 3 | Density of 1st filters | 0.012 | 0.023 | 0.012 | 0.023 | 0.001 | 0.001 | 0.007 | 0.010 |
| | Density of D -the filters | - | - | - | - | 0.018 | 0.023 | - | - |
| | Average #hops | 95.09 | 96.38 | 96.29 | 86.85 | 114.76 | 114.33 | 113.2 | 113.37 |
| | Success rate of lookup | 0.990 | 0.989 | 0.987 | 0.990 | 0.980 | 0.981 | 0.982 | 0.981 |
| 4 | Density of 1st filters | 0.033 | 0.064 | 0.033 | 0.064 | 0.001 | 0.001 | 0.038 | 0.051 |
| | Density of D -the filters | - | - | - | - | 0.120 | 0.156 | - | - |
| | Average #hops | 20.61 | 20.72 | 27.32 | 23.37 | 23.88 | 23.89 | 23.91 | 24.21 |
| | Success rate of lookup | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 5 | Density of 1st filters | 0.085 | 0.161 | 0.087 | 0.164 | 0.001 | 0.001 | 0.184 | 0.237 |
| | Density of D -the filters | - | - | - | - | 0.557 | 0.656 | - | - |
| | Average #hops | 8.88 | 7.58 | 18.63 | 20.42 | 9.27 | 9.35 | 10.83 | 10.67 |
| | Success rate of lookup | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

| #generated items = 45000 | | | | | | | | | |
|--------------------------|-----------------------------|---------|--------------|-------|--------------|--------|--------|--------|--------|
| Dissemination Radius | Evaluation items | Pattern | | | | | | | |
| | | D1 | D2 | E1 | E2 | A1 | A2 | S1 | S2 |
| 3 | Density of filters | 0.018 | 0.035 | 0.018 | 0.035 | 0.001 | 0.001 | 0.011 | 0.014 |
| | Density of D -the filters | - | - | - | - | 0.026 | 0.035 | - | - |
| | Average #hops | 95.98 | 98.65 | 94.77 | 86.68 | 112.77 | 114.88 | 113.59 | 113.74 |
| | Success rate of lookup | 0.990 | 0.986 | 0.989 | 0.991 | 0.984 | 0.982 | 0.983 | 0.982 |
| 4 | Density of filters | 0.048 | 0.094 | 0.048 | 0.094 | 0.001 | 0.002 | 0.057 | 0.075 |
| | Density of D -the filters | - | - | - | - | 0.173 | 0.224 | - | - |
| | Average #hops | 20.87 | 20.61 | 30.73 | 26.22 | 23.87 | 24.09 | 24.31 | 24.27 |
| | Success rate of lookup | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 5 | Density of filters | 0.124 | 0.231 | 0.126 | 0.233 | 0.001 | 0.002 | 0.262 | 0.332 |
| | Density of D -the filters | - | - | - | - | 0.697 | 0.790 | - | - |
| | Average #hops | 9.73 | 7.93 | 24.47 | 29.77 | 11.57 | 12.08 | 12.55 | 11.78 |
| | Success rate of lookup | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

| #generated items = 60000 | | | | | | | | | |
|--------------------------|-----------------------------|---------|--------------|-------|--------------|--------|--------|--------|--------|
| Dissemination Radius | Evaluation items | Pattern | | | | | | | |
| | | D1 | D2 | E1 | E2 | A1 | A2 | S1 | S2 |
| 3 | Density of filters | 0.023 | 0.046 | 0.023 | 0.046 | 0.001 | 0.002 | 0.014 | 0.019 |
| | Density of D -the filters | - | - | - | - | 0.035 | 0.046 | - | - |
| | Average #hops | 95.46 | 95.64 | 95.78 | 84.01 | 113.93 | 114.53 | 113.86 | 114.28 |
| | Success rate of lookup | 0.989 | 0.987 | 0.987 | 0.993 | 0.980 | 0.981 | 0.984 | 0.981 |
| 4 | Density of filters | 0.064 | 0.123 | 0.064 | 0.123 | 0.002 | 0.002 | 0.075 | 0.099 |
| | Density of D -the filters | - | - | - | - | 0.224 | 0.285 | - | - |
| | Average #hops | 20.96 | 20.02 | 33.05 | 29.55 | 23.8 | 23.65 | 24.25 | 24.48 |
| | Success rate of lookup | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 5 | Density of filters | 0.161 | 0.294 | 0.164 | 0.295 | 0.002 | 0.003 | 0.332 | 0.414 |
| | Density of D -the filters | - | - | - | - | 0.035 | 0.046 | - | - |
| | Average #hops | 10.57 | 8.44 | 30.48 | 42.89 | 14.42 | 15.16 | 15.69 | 14.82 |
| | Success rate of lookup | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

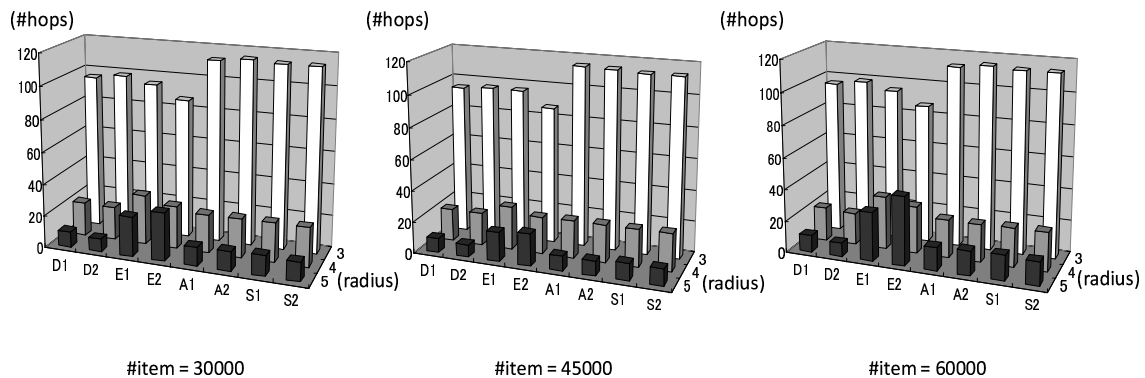
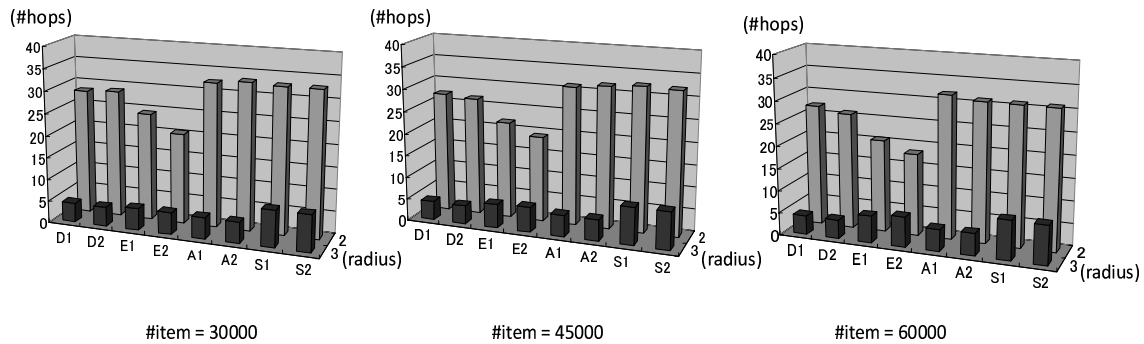
**Fig. 5** The result of the average number of hops in flat topology.

Table 5 The result in hierarchical topology.

| #generated items = 30000 | | | | | | | | | |
|--------------------------|-----------------------------|---------|-------------|-------|--------------|-------|-------|-------|-------|
| Dissemination Radius | Evaluation items | Pattern | | | | | | | |
| | | D1 | D2 | E1 | E2 | A1 | A2 | S1 | S2 |
| 2 | Density of 1st filters | 0.012 | 0.024 | 0.012 | 0.024 | 0.001 | 0.001 | 0.004 | 0.006 |
| | Density of D -the filters | - | - | - | - | 0.008 | 0.011 | - | - |
| | Average #hops | 28.39 | 28.82 | 24.27 | 20.48 | 32.55 | 33.27 | 32.91 | 32.91 |
| | Success rate of lookup | 0.999 | 0.998 | 0.999 | 1.000 | 0.999 | 0.998 | 0.999 | 0.999 |
| 3 | Density of 1st filters | 0.088 | 0.165 | 0.090 | 0.167 | 0.001 | 0.001 | 0.063 | 0.083 |
| | Density of D -the filters | - | - | - | - | 0.163 | 0.209 | - | - |
| | Average #hops | 4.39 | 4.31 | 4.99 | 4.98 | 4.81 | 4.72 | 8.36 | 8.34 |
| | Success rate of lookup | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

| #generated items = 45000 | | | | | | | | | |
|--------------------------|-----------------------------|---------|-------------|-------|--------------|-------|-------|-------|-------|
| Dissemination Radius | Evaluation items | Pattern | | | | | | | |
| | | D1 | D2 | E1 | E2 | A1 | A2 | S1 | S2 |
| 2 | Density of filters | 0.018 | 0.036 | 0.018 | 0.036 | 0.001 | 0.001 | 0.007 | 0.009 |
| | Density of D -the filters | - | - | - | - | 0.013 | 0.017 | - | - |
| | Average #hops | 27.20 | 26.64 | 21.84 | 19.25 | 31.19 | 31.96 | 32.53 | 32.25 |
| | Success rate of lookup | 0.999 | 0.999 | 1.000 | 1.000 | 0.998 | 0.998 | 0.999 | 0.998 |
| 3 | Density of filters | 0.128 | 0.233 | 0.130 | 0.234 | 0.001 | 0.001 | 0.092 | 0.120 |
| | Density of D -the filters | - | - | - | - | 0.229 | 0.288 | - | - |
| | Average #hops | 4.29 | 4.18 | 5.41 | 5.68 | 4.83 | 4.80 | 8.50 | 8.53 |
| | Success rate of lookup | 01.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

| #generated items = 60000 | | | | | | | | | |
|--------------------------|-----------------------------|---------|-------------|-------|--------------|-------|-------|-------|-------|
| Dissemination Radius | Evaluation items | Pattern | | | | | | | |
| | | D1 | D2 | E1 | E2 | A1 | A2 | S1 | S2 |
| 2 | Density of filters | 0.024 | 0.047 | 0.024 | 0.047 | 0.001 | 0.001 | 0.009 | 0.012 |
| | Density of D -the filters | - | - | - | - | 0.017 | 0.022 | - | - |
| | Average #hops | 27.11 | 25.93 | 20.68 | 18.39 | 31.95 | 31.14 | 31.04 | 31.02 |
| | Success rate of lookup | 0.999 | 0.999 | 1.000 | 1.000 | 0.998 | 0.999 | 0.999 | 0.999 |
| 3 | Density of filters | 0.165 | 0.290 | 0.167 | 0.293 | 0.001 | 0.001 | 0.120 | 0.156 |
| | Density of D -the filters | - | - | - | - | 0.288 | 0.355 | - | - |
| | Average #hops | 4.28 | 4.18 | 6.03 | 6.70 | 4.91 | 4.99 | 8.82 | 8.70 |
| | Success rate of lookup | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

**Fig. 6** The result of the average number of hops in hierarchical topology.

because the error probability in the EDBF method rapidly grows if the density of filters becomes higher (the reason is mentioned in Sect. 5). By the same reason, the result of E2 (many hash functions case) is worse than that of E1 (a few hash functions case) and the result for 60000 generated items is worse than that for 30000 items. Next, we compare DDBF, ABF and SBF method. Among them, the DDBF method marks the smallest average number of hops.

Table 5 presents the results on the hierarchical topol-

ogy. We also show Figure 6 presenting the results of the average number of hops. When dissemination radius is two, the EDBF method marks better performance than other three methods. However, when dissemination radius are three, the DDBF method marks the best performance in four methods. This result shows the DDBF method is better than other methods also in the case of biased topology. It is a witness to show the advantage of the DDBF method on various topologies.

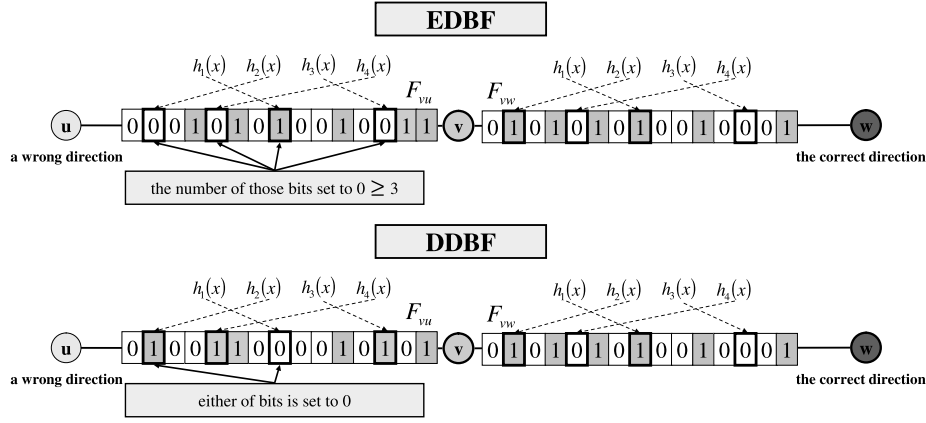


Fig. 7 The conditions for correct forwarding in EDBF and DDBF.

On both topologies, the DDBF method marks better performance than other methods in almost all cases. Exceptionally, when the dissemination radius is too small, the performance of the DDBF method is worse than the EDBF method because of too small filter density. However, this is not the optimized case and we can enlarge the dissemination radius in order to improve the performance. Actually, the DDBF method enlarging the dissemination radius achieves quite better performance than the EDBF with a small radius (e.g., comparison between the pattern E2 with radius three and D2 with radius four). For the reasons mentioned above, we can conclude that the DDBF method has a distinct advantage among the four methods.

5. Discussion

As we mentioned in Sect.4.2, the DDBF method can achieve better lookup performance than existing ones. In this subsection, we investigate why DDBF works better than EDBF and ABF.

5.1 The Reason Why DDBF Works Better Than EDBF

The DDBF method and the EDBF method have the essential difference in their filter construction and the scheme of membership test. Since the EDBF method probabilistically constructs the filters, it ensures only the expected number of bits sets to 1 for each item. In addition, queries are forwarded by referring to only the number of bits set to 1 corresponding to the hashed values of their targets. On the other hand, the DDBF method deterministically constructs the filters. Thus, it does not ensure only the number of bits but also their locations. To help to understand the reason behind the advantages of the DDBF method over the EDBF method, we show an example, which is shown in Fig. 7. We consider that a peer v is forwarding the query for item x to one of its neighbor. Only w is the correct direction, and the other peer u is a wrong direction. Let the number of hash functions $k = 4$, $H_1 = \{h_1, h_2, h_3, h_4\}$, $H_2 = \{h_1, h_2\}$, $H_3 = \{h_1\}$, $F_{vu}[h_1(x)] = 1$, $F_{vw}[h_2(x)] = 1$, $F_{vw}[h_3(x)] = 0$, and $F_{vw}[h_4(x)] = 0$. Then, in the EDBF method, if the query is

forwarded to the correct direction w , the number of bits corresponding to $F_{vu}[h_1(x)]$, \dots , $F_{vu}[h_4(x)]$ set to 0 is at least three (to simplify the argument, we do not consider the case when the values of $F_{vw}.find(x)$ and $F_{vu}.find(x)$ are equal). On the other hand, in the DDBF method, if the bit corresponding to either $F_{vu}[h_1(x)]$ or $F_{vu}[h_2(x)]$ is set to 0, the query is necessarily forwarded to the correct direction w . Clearly, the latter case in DDBF occurs with higher probability than the former case in EDBF if the probability that each bit is set to 1 in DDBF is equal to that in EDBF. This also implies that the performance in the EDBF method rapidly degenerates when the dissemination radius become overlarge and that implies that the DDBF method has an advantage in respect to the robustness for parameter settings and that the performance of the DDBF method is relatively insensitive to the deviation from the optimal setting.

5.2 The Reason Why DDBF Works Better Than ABF

In the DDBF method, one filter is assigned to one link in the DDBF method and the number of hash functions depends on the distance from the filter owner to the item holder. On the other hand, in the ABF method, D filters are assigned to one link and an i -th filter for a link represents a set of data items stored in peers i hops away from the filter holder. Thus, if the memory requirements for one link in two methods are equal, the size of each filter in the ABF method is as one- D th as that in the DDBF method.

In the ABF method, each peer receives a few filters from nearby peers and many filters from faraway peers and the number of hash functions used all filters is same. Thus, the number of bits 1 in a combined filter for nearby peers is very small. This implies the waste of memory space. In contrast, that for faraway peers is very large. This causes high error probability when a query stays in a faraway peer from an item holder. Consequently, the ABF method does not so efficiently use memory space than the DDBF method. We show an example as follows (Fig. 8). Let the size of filter $m = 60$, the dissemination radius $D = 3$, the number of hash functions is $|H_1| = 3$, $|H_2| = 2$ and $|H_3| = 1$ in the DDBF

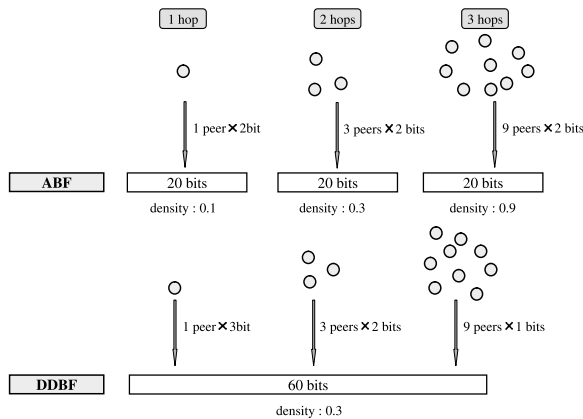


Fig. 8 An example of density difference in DDBF and ABF.

method, and $k = 2$ in ABD method. We assume that the number of one-hop away peers is one, and those of two-hop and three-hop away peers are three and nine respectively. Note that the size of i -th filters in the ABF method is 20. To simplify the argument, we assume that the conflict of the hashed values never occurs. In the DDBF method, a peer receives total 18 bits and the density of this filter is 0.3. On the other hand, in the ABF method, a peer receives two bits from one-hop away peer and the density of the first filter is 0.1. And it receives 6 bits from two hops away peer and 18 bits from three hops away peer. Then the density of the second filter is 0.3 and that of the third filter is 0.9. Since the error probability depends on the density of filters, if a query stays in a peer three hops away from an item holder, this probability in ABD method is much higher than the DDBF method. If a query stays in a peer one-hop away from an item holder, the density in ABF is smaller than EDBF. However, the error probability in ABF is not much smaller than DDBF (To be concrete, the error probability in ABF is 1.0% and that in DDBF is 2.7% in this example). As a result, the DDBF method achieves better lookup performance than the ABF method.

The point in which DDBF is inferior to ABF is the robustness for the setting of overlarge dissemination radius. When the dissemination radius is very large, the density of each filter in the DDBF method becomes too large. This causes high error probability even if a query is one-hop away from the peer having its target. On the other hand, each peer in the ABF method separately maintains D filters. Then, regardless of the dissemination radius, the first filter keeps low density, and thus if a query is one-hop away from the target holder, it is forwarded to the correct direction with high probability. This is why the performance of the DDBF method is worse than that of the ABF method in the case of overlarge dissemination radius. However, this advantage of the ABF method does not make so much sense because in such cases the ABF method is also far from appropriate settings. Unfortunately, in every Bloom-filter-based method, neither the formula to calculate the best parameter setting nor the adjusting scheme that makes the system converge to

the optimal setting is found yet. It is one of future works to find the optimal and to develop such self-adjusting mechanism.

6. Conclusion

In this paper, we proposed an efficient index dissemination based on an extension of Bloom filters, called Deterministic Decay Bloom Filter (DDBF) and a search method based on it. The key idea of DDBF is to decay deterministically the number of used hash functions according to the distance between filter owners and data item holders. That is, each peer maintains more information in filters disseminated with small range, less information in those with large range. With this technique, we can reduce the error probability in referring to filters, and thus improve the lookup efficiency. We showed by simulations that the DDBF-based method can achieve better lookup performance than the existing ones.

The future work includes improvement of query routing and the analysis of proper parameter setting. In particular, designing self-adjusting mechanisms for parameter setting is one of our future challenges.

Acknowledgment

This work is supported in part by Global COE (Centers of Excellence) Program of MEXT, Grant-in-Aid for Scientific Research ((B) 19300017, (B) 17300020, (B) 20300012) of JSPS, Grant-in-Aid for Scientific Research on Priority Areas (16092215) of MEXT, Grant-in-Aid for Young Scientists ((B) 19700058) of JSPS, and the Ookawa Foundation Research Grant.

References

- [1] Napster Inc., "the napster homepage," <http://www.napster.com/>, 2001.
- [2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," Proc. ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Aug. 2001.
- [3] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," Proc. Middleware, Lecture Notes in Computer Science, vol. 2218, pp. 329–350, Nov. 2001.
- [4] I. Stoica, R. Moryis, D.R. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," Proc. ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Aug. 2001.
- [5] K. Hildrum, J.D. Kubiatowicz, S. Rao, and B.Y. Zhao, "Distributed object location in a dynamic network," Proc. 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA), Aug. 2002.
- [6] O.S. Community, "Gnutella," <http://gnutella.com/>, 2001.
- [7] P. to-Peer technology company, "FastTrack," <http://www.fasttrack.nu/>, 2001.
- [8] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," Proc. ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp. 177–190, Aug. 2002.

- [9] B.F. Cooper, "Quickly routing searches without having to move content," Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS), Lecture Notes in Computer Science, vol.3640, pp.163–172, Feb. 2005.
- [10] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," Proc. 16th International Conference on Supercomputing (ICS), pp.84–95, 2002.
- [11] M. Zhong and K. Shen, "Popularity-biased random walks for peer-to-peer search under the square-root principle," Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS), 2006.
- [12] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," Internet Mathematics, vol.1, no.4, pp.485–509, 2004.
- [13] M. Mitzenmacher, "Compressed bloom filter," IEEE/ACM Trans. Netw., vol.10, no.5, pp.604–612, Oct. 2002.
- [14] J. Kubiawicz and S.C. Rhea, "Probabilistic location and routing," Proc. 21st Conference of the IEEE Communications Society (INFOCOM), pp.1248–1257, June 2002.
- [15] A. Kumar, J. Xu, and E.W. Zegura, "Efficient and scalable query routing for unstructured peer-to-peer networks," Proc. 24th Conference of the IEEE Communications Society (INFOCOM), pp.1162–1173, March 2005.
- [16] J.W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," IEEE/ACM Trans. Netw. (TON), vol.12, no.5, pp.767–780, Oct. 2004.
- [17] A. Cheng and Y. Joung, "Probabilistic file indexing and searching in unstructured peer-to-peer networks," Comput. Netw., vol.50, no.1, pp.106–127, Jan. 2006.
- [18] M.T. Prinkey, "An efficient scheme for query processing on peer-to-peer networks," <http://aeolusres.homestead.com/files/index.html>
- [19] C. Rohrs, "Query routing for the gnutella network," http://www.limewire.com/developer/query_routing/keyword%20routing.htm, May 2001.
- [20] B. Awerbuch, "Complexity of network synchronization," J. ACM (JACM), vol.32, no.4, pp.804–823, Oct. 1985.
- [21] I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic coding for data compression," Commun. ACM (CACM), vol.30, no.6, pp.520–540, June 1987.
- [22] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern p2p file-sharing systems," ACM SIGCOMM Internet Measurement Conference, pp.49–62, Oct. 2005.
- [23] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability," <http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html>, 2001.



Taisuke Izumi received the M.E. and D.I. degrees in computer science from Osaka University in 2003 and 2006. He is now an Assistant Professor of Graduate School of Engineering, Nagoya Institute of Technology. His research interests include distributed algorithms. He is a member of ACM and IEEE.



Hirotosugu Kakugawa received the B.E. degree in engineering in 1990 from Yamaguchi University, and the M.E. and D.E. degrees in information engineering in 1992, 1995 respectively from Hiroshima University. He is currently an associate professor of Osaka University. He is a member of the IEEE Computer Society and the Information Processing Society of Japan.



Toshimitsu Masuzawa received the B.E., M.E. and D.E. degrees in computer science from Osaka University in 1982, 1984 and 1987. He had worked at Osaka University during 1987–1994, and was an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST) during 1994–2000. He is now a professor of Graduate School of Information Science and Technology, Osaka University. He was also a visiting associate professor of Department of Computer Science, Cornell University between 1993–1994. His research interests include distributed algorithms, parallel algorithms and graph theory. He is a member of ACM, IEEE and IPSJ.



Yusuke Takahashi received the B.E degree in computer science from Osaka University in 2006. He is now a student of Graduate School of Information Science and Technology, Osaka University. His research interests include distributed algorithms.