Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 31 (2008) 304–317
computer communications
www.elsevier.com/locate/comcom

Efficient file search in non-DHT P2P networks

Shiping Chen [a], Zhan Zhang [b,*], Shigang Chen [b], Baile Shi [c]

[a] Network Center, University of Shanghai for Science and Technology, China
[b] Department of Computer & Information Science & Engineering, University of Florida, USA
[c] Department of Computer Science, FuDan University, China

Available online 17 August 2007

Abstract

Unstructured P2P networks dominate in practice due to their small maintenance overhead. However, the high volume of search traffic threatens its continued growth. The focus of this paper is to study how to improve the search efficiency in a non-DHT P2P network without a distributed indexing structure. We identify possible performance problems in KaZaa and Gnutella, and propose a flexible two-phase ticket-based search algorithm (TBS). In particular, the first phase is designed to reduce the search overhead and lookup delay in searching popular (highly replicated) files, and the second phase is designed to reduce the excessive duplicate messages in searching unpopular (rare) files. In addition, a random sampling solution is proposed to estimate timeout between consecutive search rounds, and a moving anchor solution is proposed to reduce duplicate visits of the same node. Moreover, we propose a ticket-based broadcast algorithm (TBA) by slightly modifying TBS, which has the significance in supporting various network functionalities. We evaluate the performance of the new techniques by both analysis and simulations, which demonstrate that the proposed solutions outperform the existing alternatives.

Keywords: P2P networks; Overlay algorithms; Distributed file sharing; Overlay broadcast

1. Introduction

P2P (peer-to-peer) systems for file sharing fall in two categories: unstructured P2P networks [1–5] and structured P2P networks with DHT-based (Distributed Hash Table) search algorithms [6–19]. They contrast each other in trade-off between search overhead and maintenance overhead.

A structured P2P network has a strictly defined overlay topology, which infers a unique search path for any lookup request. Its search overhead is small, but the maintenance overhead can be huge for a highly dynamic system. When a new node joins, the following operations have to be performed. First, a lookup request must be issued to find the node's location in the overlay network. Second, a portion of the file-index database is transferred from an adjacent node. Third, a number of lookup requests are issued to find its overlay neighbors. Fourth, for each file to be shared from this node, a lookup request is issued to find where the index information should be placed. For a single join, the overhead can be large if the node carries many shared files. Moreover, nodes may join and leave very frequently. The observations on FastTrack users showed that over 20% of the overlay connections last 1 min or less and 60% of the IP addresses keep active for no more than 10 min each time after they join the system [20]. This has two implications. First, a high rate of node joins means a high rate of lookup requests to add the index information for the files that the new nodes carry. Second, a high rate of node departures means that the existing index information must be timed out to prevent staleness, which further means that, to keep a shared file in the system, the index information must be periodically refreshed, causing addition overhead. Besides, it is a complicated matter to support multi-attribute or partial lookups in a structured P2P network.

* Corresponding author. Tel.: +1 3528465444.
E-mail addresses: spchen@usst.edu.cn (S. Chen), zzhan@cise.ufl.edu (Z. Zhang), sgchen@cise.ufl.edu (S. Chen), bshi@fudan.edu.cn (B. Shi).

0140-3664/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2007.08.011

Without a rigid index-placement scheme, unstructured P2P networks have small maintenance overhead, which is the reason that they dominate in practice. However, the file lookup is resolved by flooding, which has serious scalability problem [21]. Measurements in [22] have shown that the flooding algorithm generates 330 TB/month in a Gnutella network with only 50,000 nodes. Even with the use of super nodes in Kazaa (ultrapeers in Gnutella) and TTL-constrained flooding (also called expanding ring), traffic volume is still high. Observing that P2P traffic becomes dominant and even exceeding web traffic [20,23], many organizations block P2P traffic to prevent performance degradation of other network applications. Improving search performance of unstructured networks is extremely important to supporting future P2P applications without overloading the Internet.

A hybrid approach [24] was proposed to use an unstructured network for popular files and a structured network for unpopular files. However, without a central server maintaining the historical access statistics of all files, the problem of determining which files are popular or unpopular is not satisfactorily addressed. Furthermore, regardless how frequent they are accessed, if the number of unpopular files is large, the overhead of maintaining the indexes in the structured network remains a serious problem. Another hybrid approach [25] utilized partial index to help peers to find shared interests, and provide search hints for those data difficult to locate even after interest-based clustering.

This paper has three contributions. First, we identify performance problems in the existing P2P file search systems, and propose an efficient ticket-based search algorithm (TBS). The existing approaches suffers different problems in searching for popular (highly replicated) and unpopular(rare) files. The proposed algorithm addresses these problems, and is able to adapt itself to suit both popular and unpopular files.

Second, we propose two optimization methods to improve the performance of P2P file search systems. A random sampling solution is designed to estimate the appropriate timeout between consecutive search rounds and a moving anchor solution is designed to reduce duplicate visits of the same node.

Last but not least, we propose a broadcast algorithm, called ticket-broadcast algorithm (TBA), which is derived from the ticket-based search algorithm. TBA has its own significance in supporting various network functionalities, e.g., collecting statistics, distributing notifications, etc. TBA works far better than the existing broadcast alternatives.

The rest of the paper is organized as follows. Section 2 identifies problems in existing approaches. Section 3 proposes a ticket-based algorithm, and two heuristics to improve the performance of P2P search systems. Section 4 proposes a ticket-based broadcast algorithm, and studies its performance. Section 5 provides simulation results. Section 6 draws the conclusion.

2. Inefficiency in existing P2P file search systems

In this section, we first define the network model, study the optimal search overhead, and then discuss the problems in existing P2P file search systems in details.

2.1. Network model

We use Kazaa's structure, which is also adopted by the current Gnutella [26]. There are two types of nodes, *super nodes (or ultrapeers)* and *leaf nodes*. An overlay network is formed among the super nodes, each of which carries a set of leaf nodes. When a leaf node joins at a super node, it reports its file indexes to the super node. When a node looks up a file, it issues a request to its super node, which initiates a search process in the overlay network to locate the file. From now on, we shall focus only on the super nodes and the overlay network formed among them. In the sequel, we refer to a super node simply as a node.

Let N be the set of nodes, n be the number of nodes in N, and c_x be the number of neighbors of a node $x \in N$, and c be the expected value of c_x, i.e., the average number of neighbors per node or the average node degree. The k-neighborhood of x is the set of nodes that are k or less overlay hops away from x. While our discussions in this paper are made mainly for looking up a file, we want to point out that the proposed techniques can be easily applied to pattern search, i.e., finding a specified number of files that match a given pattern. The notations, including those defined later, can be found in Table 1 for quick reference.

2.2. Optimal search overhead

Suppose a file has m replicated copies *randomly* distributed in the overlay network. Let $p(i)$ be the probability of not finding any copy of the file after i nodes are searched. The file is not found if the set of m[1] nodes with the file does not overlap with the set of i nodes that are searched. $\binom{m}{n}$ is the total number of different ways that the file can be distributed in the network, and $\binom{m}{n-i}$ is the number of different ways that the file can be distributed among the nodes that are not searched. Hence,

$$p(i) = \frac{\binom{m}{n-i}}{\binom{m}{n}} = \frac{(n-i)!(n-m)!}{n!(n-i-m)!}$$

Suppose we sequentially search a random sequence of distinctive nodes one after another. Let Ω be the number of nodes to be searched before finding the first copy of the file. It is a random variable with a distribution between 1 and

[1] m is only for the purpose of analysis, and is not known in practice.

Table 1
Notations

N	Set of (super) nodes
n	Number of (super) nodes
c_x	Number of neighbors of x
c	Average number of neighbors per node
m	Number of replicated copies of a file in the network
Ω	Number of nodes to be searched before locating a file
$E(\Omega\|m,n)$ or $E(\Omega)$	Expected value of Ω
l, q	Parameters for random walk
d	Parameter for limited-degree flooding
α, λ	Parameter for ticket-based flooding

$n-m+1$. The expected value $E(\Omega|\ m,n)$ is a function of both m and n.

$$E(\Omega|m,n) = \sum_{i=1}^{n-m+1} i \cdot p(i-1) \cdot \frac{m}{n-i+1} = \sum_{i=1}^{n-m+1} i \frac{(n-i+1)!(n-m)!}{n!(n-i-m+1)!} \frac{m}{n-i+1} \quad (1)$$

where $p(i-1)$ is the probability of not finding any copy at the first $(i-1)$ searched nodes and $\frac{m}{n-i+1}$ is the conditional probability of finding a copy at the ith node. Combined, $p(i-1)\frac{m}{n-i+1}$ is the probability of finding the first copy at the ith nodes. When the context makes it clear, we will abbreviate $E(\Omega|m,n)$ simply as $E(\Omega)$

Because $E(\Omega)\cdot m$ is an increasing function with respect to m according to the figure, we have[2]

$$E(\Omega\ |m=1,n) \leqslant E(\Omega)\cdot m \leqslant E(\Omega\ |\ m=n,n)\cdot n$$

From (1), $E(\Omega\ |\ m=1,n)=\frac{n+1}{2}$ and $E(\Omega|m=n,n)=1$. We have

$$\frac{n+1}{2} \leqslant E(\Omega)\cdot m \leqslant n$$
$$\frac{n+1}{2m} \leqslant E(\Omega) \leqslant \frac{n}{m}$$

Based on the above inequality, m has a great impact on the average number of nodes $E(\Omega)$ needed to be searched. The left plot of Fig. 1 shows the curve of $E(\Omega)$ with $n=10{,}000$. When $m>1000$, $E(\Omega)<10$, which means that finding a highly popular file is trivial. With high probability, it can be found in a very small neighborhood. The right plot shows how to quickly estimate the value of $E(\Omega)$. For $m>100$, $E(\Omega)\cdot m\approx n$ and thus $E(\Omega)\approx n/m$.

One approach, proposed by Kumar et al., is for each node to advertise the information about its files in a neighborhood [27]. When a node receives a lookup message and it knows the whereabout information of the requested file, it can help direct the message to the node that has the file. This approach can reduce the search overhead at the cost of periodic advertisement in order to prevent information staleness. The technique of artificially increasing m is orthogonal to the new techniques to be proposed in this paper, which improve the search efficiency for any m values. The two types of techniques can be used together for better performance.

[2] Note that $E(\Omega)$ is the abbreviation of $E(\Omega\ |\ m,n)$, which is a function of m.

2.3. Inefficiency in existing systems

There is significant room for improvement in the search efficiency of some commonly deployed P2P systems. In searching for popular (highly replicated) files, the existing systems suffer excessive unnecessary communication overhead or long lookup delay. In searching for unpopular (rare) files, they suffer from excessive duplicate messages.

2.3.1. Inefficiency in searching for popular files

The current Gnutella uses the TTL-constrained flooding approach to limit the search scope. The basic idea is to search increasingly larger neighborhood until the requested file is found. The source node broadcasts a lookup request with an initial TTL (time to live) value, specifying the maximum number of hops the request will travel. When a node receives the request for the first time, if it has the requested file, it replies to the source; otherwise, if the TTL is greater than zero, the node reduces TTL by one and forwards the request to all neighbors except the one from which the request is received. If the source does not receive any reply after a timeout period, it broadcasts another request with an increased TTL. This process repeats until one or more replies are received or TTL exceeds a threshold value. We call each broadcast of the request as *one search round* (or *round*).

In the case of searching for popular files, we can assume that the number of nodes in a k-neighborhood under search is roughly c^k. This assumption is reasonable, because the searched neighborhood is likely to represent a small portion of the network, the probability for the nodes to share common direct neighbors is very small, and consequently the search follows a tree structure with each internal node having c children on average. The number of leaf nodes, roughly c^k, is much more than the number of internal nodes when c is reasonably large.

Recall that Ω represents the number of nodes that are searched when the first copy of the file is located. In order to cover Ω nodes, the TTL value in the flooding has to grow as large as $\lceil \log_c \Omega \rceil$. The number of nodes to be searched in the $\lceil \log_c \Omega \rceil$-neighborhood is roughly $c^{\lceil \log_c \Omega \rceil}$. Because $\lceil \log_c \Omega \rceil$ can be close to $\log_c \Omega + 1$, the overhead (in terms of the number of nodes that are searched) can be as large as

$$c^{\log_c \Omega + 1} = c \cdot \Omega \quad (2)$$

which is c times of Ω. Namely, $(c-1)$ times of more nodes are searched even after the file is located. It represents a serious overhead problem even when c is modest, e.g., 10. The fundamental reason for this problem is that the overhead in TTL-constrained flooding increases at very coarse discrete steps – it increases by c times for each search round.

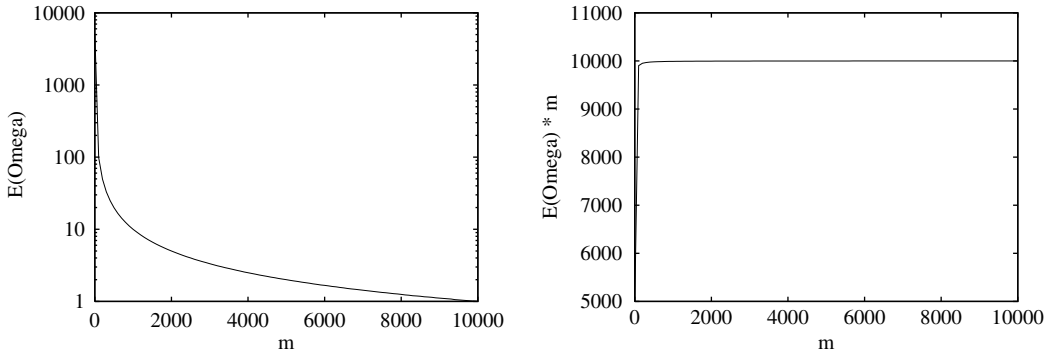

Fig. 1. $E(\Omega)$ is the expected number of nodes to be searched.

Limited-degree flooding artificially reduces the value of c. Each intermediate node forwards the lookup message to $d(<c)$ randomly selected neighbors. By (2) and with c replaced by d, the number of searched nodes is reduced to $d \cdot \Omega$. However, in this approach, fewer nodes can be covered than that in TTL-constrained flooding within the same number of hops. In order to cover Ω nodes, the value of TTL has to be $\lceil \log_d \Omega \rceil$, which is greater than $\lceil \log_c \Omega \rceil$. A larger TTL means that the lookup message has to travel more hops to complete a search round, which causes a greater lookup delay. Probabilistic flooding [28] artificially reduces the value of c by forwarding the lookup message to each neighbor with a certain probability. It has a similar problem as limited-degree flooding does. LightFlood [29] relies on pre-constructed tree-like sub-overlay called FloodNet to avoid revisiting the same nodes in the networks. It has the same serious overhead problem because the huge gap between two consecutive TTL values: the overhead is still as large as $d \cdot \Omega$ (d is the average degree of the nodes in the tree).

2.3.2. Inefficiency in searching for unpopular files

TTL-constrained flooding approach is designed to search for highly replicated (popular) files. To look up such a file, it is sufficient to search only a small number of nodes. When TTL of the constrained flooding is not large, recursively forwarding a lookup message to neighbors is a good way to distribute the message across the overlay network, especially when the neighbors are randomly selected. However, after a large portion of nodes have received the lookup message, further forwarding the message to neighbors is increasingly inefficient.

When the number of replicated copies of a file, m, is very small, the expected number of nodes to be searched, $E(\Omega)$, will be very large. For example, by (1), if $m = 1$, we have

$$E(\Omega \mid m = 1, n) = \sum_{i=1}^{n} i \frac{n-i+1}{n} \frac{1}{n-i+1} = \frac{1}{n} \sum_{i=1}^{n} i = \frac{n+1}{2}$$

Note that $\frac{n+1}{2}$ is the average number. In the worse case, all nodes have to be searched. In order to cover all nodes, the TTL of constrained flooding has to be as large as the network diameter, which causes the message to be forwarded on every overlay link, and the probability of hitting an already-searched node (called *duplicate visits*) becomes larger than the probability of hitting an unsearched node. Thus, this approach is too expensive for the search of (unpopular) files with one or a few copies.

Another approach is to carefully arrange the neighbor links to create a balanced broadcast tree among all nodes, which achieves the optimal hop complexity of $O(\log_c n)$ and the optimal message complexity of n. This approach is however impractical for a high-dynamic P2P network where tens or even hundreds of nodes join and depart at any moment. Moreover, the internal nodes, which account for a small portion of a tree with an average degree of c, will take the full burden of forwarding the broadcast traffic.

Random walk delivers a message through a set of random paths. Limited-degree flooding forwards a message to a fixed number (but not all) of neighbors at each hop. Probabilistic flooding forwards a message to neighbors with a certain probability at each hop. They are not suitable for broadcast. As our simulations will demonstrate, besides the performance problems, they may fail in reaching a portion of nodes.

Although *unpopular* files may be accessed less frequently, their number can be much larger than the number of *popular* files. Moreover, their importance to some users should not be under-estimated just because fewer users need them. Without an efficient algorithm searching for unpopular files, a major portion of these files are essentially *lost* in the network if TTL-constrained flooding is constrained to a limited scope.

The dynamic query algorithm was proposed in [30], which switches the queries from probabilistic searching to random walks if the searched files are unpopular. As discussed, the probabilistic searching may result in long response time. In addition, random walks may revisit nodes in searching unpopular files and increase the communication overhead. Another dynamic querying

algorithm was proposed in [31], which tries to bridge the gap between two consecutive TTL values by dynamically calculating TTL in each searching round. However, it has the same problem as in [29] if the searched files are unpopular.

3. Ticket-based search in non-DHT P2P networks

In this section, we first propose a ticket-based search algorithm, which addresses the inefficiency problems discussed in Section 2, and is able to efficiently search for both popular and unpopular files. We also will discuss two optimizations to improve the performance of P2P file search systems.

3.1. Two-phase ticket-based search algorithm

Our ticket-based search algorithm (TBS) consists of two phases. Phase one is called *ticket-based flooding*. Its purpose is to improve the search efficiency of popular files by significantly reducing the search overhead without significantly increasing the lookup delay. Phase two is called *segmented ring traversal*. Its purpose is to improve the search efficiency for unpopular files by avoiding excessive duplicate visits.

The algorithm begins with Phase one, which is able to find most popular files. It switches to Phase two after Phone one has delivered the lookup message to a certain percentage of all nodes, such that further distributing the message in a flooding way is no longer efficient due to duplicate visits. Phase two is able to resolve all lookups that cannot be answered by Phase one.

3.1.1. Phase one: ticket-based flooding

The concept of *tickets* was originally proposed in [32] under a totally different context. To find a QoS-constrained route, tickets specify the maximum number of wireless paths to be probed. To locate a file, tickets in this paper specify the number of nodes to be checked in one search round. Each ticket represents the permission of searching one node. If one search round does not find the file, the number of tickets is increased by a factor α. Within each round, when a node receives a lookup message for the first time, it consumes one ticket and then forwards the message to all neighbors as long as the message carries enough tickets, which are split among the neighbors.

Ticket-based flooding is given in Fig. 2 with min_tickets and λn specifying the minimum number and maximum

```
Lookup Routine at Source Node s
(1)   t := min_tickets
(2)   seq_num := large random number
(3)   sid := s|seq_num
(4)   while (t < λn)
(5)       send Lookup(msg, t, sid, 0) to all neighbors of s
(6)       wait until timeout or receiving a reply from a node having the file
(7)       if a reply has been received then
(8)           return the reply
(9)       else
(10)          t := αt
(11)          seq_num := seq_num + 1
(12)  t := λn
(13)  send Lookup(msg, t, sid, 1) to all neighbors of s

Upon receipt of message Lookup(msg, t, sid, f) by node x
      /* Phase One */
(1)   if t > 0 and x receives msg with sid for the first time then
(2)       if x has the requested file then
(3)           reply to the source node
(4)       else
(5)           t := t − 1
(6)           r := t − ⌊t/c_x⌋ c_x
(7)           for each of the first r neighbors, denoted as y, do
(8)               forward Lookup(msg, ⌈t/c_x⌉, id) to y
(9)           if ⌊t/c_x⌋ > 0 then
(10)              for each of the (c_x − r) remaining neighbors, denoted as y, do
(11)                  forward Lookup(msg, ⌊t/c_x⌋, id) to y
                  /* Phase Two */
(12)          if f = 1 and x hasn't forwarded the query with sid to successor(x) then
(13)              forward (msg, 0, sid, 1) to successor(x)
      /* Phase Two */
(14)  else if t = 0 and f = 1 and x hasn't forwarded the query with sid to successor(x) then
(15)      forward (msg, 0, sid, 1) to successor(x)
(16)  else
(17)      discard the message
```

Fig. 2. Ticket-based search algorithm (TBS).

number of tickets used in any round of Phase one, respectively, where n is the number of nodes in the system and $\lambda(<1)$ is a system parameter. Note that when the number of tickets in a round becomes larger than the maximum threshold λn, the algorithm switches to Phase two.

A lookup message contains at least four fields, *msg* carrying the information about the query, t carrying the number of tickets, *sid* uniquely identifying this round of search, and a flag f specifying which phase the algorithm is executing (0 for Phase one and 1 for Phase two).

The algorithm consists of two subroutines. One is executed at the lookup source node. It makes a sequence of search rounds with an increasing number of tickets until the file is found or the ticket number is larger than the threshold λn (in this case, switching to Phase two). The other subroutine is executed at all other nodes, upon receiving lookup messages. It consumes one ticket and forwards the received message to all neighbors, with the remaining tickets split evenly among them (lines 5–11 in the lookup routine at the source node).

In this phase, the number of searched nodes is increased in discrete steps by a tunable factor $\alpha \in (1, c]$ (line 10 in the lookup routine at the source node). In order to cover Ω nodes, the maximum number of tickets issued for one round will not exceed $\alpha\Omega$, which is smaller than $c\Omega$. Because a lookup message is forwarded to all neighbors as long as the message carries enough tickets, the search process roughly follows a tree structure with an average degree of c. Therefore, most tickets are expected to be consumed within $\log_c(\alpha\Omega) < \log_c\Omega + 1$ hops.

Ticket-based flooding achieves the benefits of both TTL-constrained flooding and limit-degree flooding. Like the former, it minimizes the lookup delay by forwarding the lookup message to all neighbors, which reduces the depth of the flooding tree in each round. Like the latter, it reduces the overhead increment in consecutive search rounds. Moreover, it makes the increment tunable by a system parameter α.

We will use simulations to demonstrate that $\alpha = 2$ achieves a favorable tradeoff between the search overhead and the search delay. α should not be chosen too small because otherwise it may cause too many search rounds.

3.1.2. Phase two: segmented ring traversal

Ticket-based flooding becomes inefficient when the number of tickets in a round is high enough to cause many duplicate visits. When this happens, the algorithm switches to Phase two, called *segmented ring traversal*, to distribute the lookup message more efficiently.

To facilitate segmented ring traversal, we augment the overlay network with an "unrestricted" non-DHT ring that connects all nodes. Each node knows the next node on the ring, called the *successor*. The unrestricted ring is fundamentally different from the DHT-based ring found in many structured P2P networks. In the DHT-based ring, each node has a specific location, which makes the maintenance of the ring difficult when a new node joins the network. In the unrestricted ring, a node can be anywhere in the ring. The ring maintenance is trivial. As long as a new node x knows another node z that is already in the ring, it informs z to set x as the successor, and then it sets its own successor to be z's previous successor. To prevent the ring from being broken after an abrupt node departure, each node should also know a number of nodes after the successor down the ring. It also knows the previous node on the ring, called the *predecessor*. In the rare case that all those nodes depart abruptly together, a broadcast is performed to identify the nodes that do not have predecessors and successors so that they can be connected to form a ring. In the description of our algorithm, when we refer to the neighbors of a node, we do not include the successor or the predecessor, which is treated separately.

We call the nodes reached in the last round of phase one as *phase-one nodes*. The operation of segmented ring traversal is illustrated by the left plot of Fig. 3. Suppose the source node s has three neighbors and nodes i, j, and l each have two neighbors. In the last round of Phase one, the message is delivered to nine nodes. The phase-one nodes partition the ring into segments, which are called *phase-two segments*. As illustrated in the right plot of Fig. 3, each phase-one node is responsible for one phase-two segment. It forwards the message to its successor, which further forwards the message to the successor's successor,. . .,until the message reaches a node that has already received the message. The second phase avoids the intense collision that happen at the final stage of flooding – most message forwardings at the last steps of flooding are made to nodes that have received the message.

The pseudo code of Phase two can be found in Fig. 2. When the ticket number t is larger than the threshold λn, the source node sets the ticket number to be λn and the flag f to be 1 (lines 12–13 in the lookup routine at the source node). When a node x receives the message, if $f = 1$, and the node has not forwarded message with *sid* to its successor before, x performs Phase two by forwarding the message along the successor links until reaching a node that has received the message previously (lines 12–13 and lines 14–15 in lookup routine at intermediate nodes).

3.1.3. Determining the threshold λn

To determine the threshold λn for transition from Phase one to Phase two, we must know λ and n. λ is a

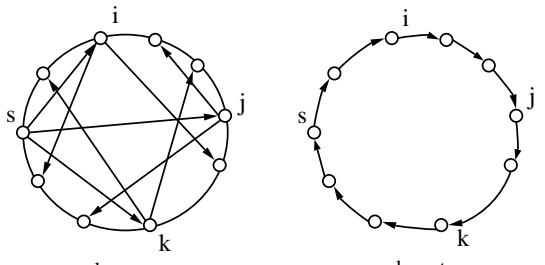

Fig. 3. Two-phase search.

system parameter that can be chosen, but n has to be measured. We use a random sampling approach to estimate n. In the last round of Phase one, the source node makes a number of special tickets. For each node that consumes a special ticket, after its phase-two segment is traversed, the length of the segment is sent back to the source. Such information can be collected during the segment traversal in Phase two. After the source receives the segment lengths, it revises the estimation of n to be the sum of the lengths divided by the number of special tickets and then multiplied by the total number of tickets. The new values of n will be used to determine the threshold in the next query.

Each query message will carry the most up-to-date estimation of n at the source if the calculation is performed recently. A node will collect these values from all received messages and update its own estimation. For example, it may take the average of all received estimations in a recent period.

We performed simulations which showed that the above approach will quickly converge the estimated values of n towards the real values if Phase two is performed regularly in the network. On the other hand, when Phase two is rarely performed, the problem of precisely determining the threshold also becomes less important.

3.2. Optimizations

3.2.1. Timeout problem and random sampling solution

Timeout between search rounds (line 6 in the lookup routine at the source node) may be a major contributor to the search delay. Traditionally, the next round starts after the previous round completes. The timeout period is set large in order to accommodate the worst-case delay of the previous round. However, the Internet delay follows a long-tail distribution [33]. The delays of most Internet paths are clustered in a relatively small range, but the delays of a small portion of Internet paths are very large. It is non-optimal when the timeout is set to be a long time (e.g., 10 s) while the lookup message reaches most nodes of a round in a short time (e.g., hundreds of milliseconds). Can we intelligently start the next round at an earlier time before timeout?

The observation is that we do not have to start the next round strictly after the previous round completes. For example, we may start the next round after the lookup message reaches a percentage β (= 75% in this paper) of nodes in the previous round. Because of the long-tail delay distribution, it takes much less time to reach a majority (instead of all) of nodes if we do not worry about the small number of nodes that are distant in terms of delay. In order for a β percentage to cover Ω nodes, the number of tickets issued by TBS has to be at least $\frac{1}{\beta}\Omega$. Due to the discrete ticket increment, the actual number of tickets issued will be up to $\alpha\frac{1}{\beta}\Omega$, a modest increase comparing with the prior number of $\alpha\Omega$.

How to estimate when the percentage of nodes that have been searched reaches β? We propose a random sampling solution. The idea is to make a small number of tickets special. They are called *sampling tickets*. When a node receives a lookup message for the first time, it consumes a ticket randomly selected from the ticket pool carried by the message. If it happens to consume a sampling ticket, the node sends a notification to the lookup source. Suppose the lookup source starts one round at time t_0, and it receives notifications for a β percentage of all sampling tickets before time t_1. If t_1 is before timeout, the node starts the next round immediately.

3.2.2. Overlapped search problem and moving anchor solution

Any search round in Phase one will revisit all nodes that have been searched in the previous rounds. One optimization is to start the rounds at different nodes. These nodes are called *anchors*. The first anchor is the lookup source, which chooses one of its neighbors as the anchor for the second round. The source sends a lookup message to all neighbors except the next anchor. The information about the next anchor is carried in the lookup message. After receiving the message, a node will not forward the message to this anchor even if it is a neighbor. To start the next round, the anchor is notified, which chooses one of its neighbors as the anchor for the yet next round before sending out the lookup message. The above process repeats until the file is found. By using different anchors to start the search rounds, the likelihood of overlapped search is reduced.

4. An efficient broadcast algorithm

In this section, we propose a broadcast algorithm, called *ticket-based broadcast algorithm (TBA)*. An efficient broadcast function in overlay networks can serve many purposes. For instance, an administrator or a user may be interested in learning the statistics of the overlay network and how the network properties change over a period of time. Providing a broadcast function is useful, but its usage should be restricted based on a privilege-based or incentive-based mechanism, which is beyond the scope of this paper.

4.1. Ticket-based broadcast algorithm (TBA)

Recall that the nodes visited by the last round of Phase one partition the ring into segments, and then in Phase two, each of those nodes forwards the query message to its successor recursively until the segment closest to the node is totally covered. Obviously, the whole network can be reached by the last round of Phase one combined with Phase two. Thus, a broadcast algorithm can be derived from the ticket-based search algorithm as follows: Instead of going through multiple rounds before entering Phase two, Phase one is changed to only include the last round, where the ticket number is set be λn, i.e., $t := \lambda n$, and the flag f is set to be 1. We call the resulting algorithm as *ticket-based broadcast algorithm (TBA)*.

4.2. Analysis

We analyze the complexities of TBA. The hop complexity is defined as the expected number of hops that a broadcast message travels from the broadcast source to an arbitrary node. The communication complexity is the expected number of times that a broadcast message is forwarded in order to reach all nodes. A smaller hop complexity means a more balanced delivery tree and a smaller average delivery latency. A smaller communication complexity means less network bandwidth consumed.

The proof of the following theorems and corollaries can be found in the appendix. To make the problem tractable, the analysis of hop complexity assumes that each node has c neighbors and forwarding a message takes one unit of time per hop. The general case where nodes have different numbers of neighbors and varied link delays will be studied by simulations.

Theorem 1 (Hop complexity). *The expected number of hops for a broadcast message to reach an arbitrary node is bounded by*

$$\log_c(\lambda n) + \frac{1}{2\left(1-\frac{1}{c}\right)\left(1-\frac{\lambda c}{c-1}\right)\lambda}$$

Corollary 1 (Hop complexity). *If* $\lambda = \Theta\left(\frac{1}{\log_c n}\right)$, *then the expected number of hops for a broadcast message to reach an arbitrary node is* $O(\log_c n)$.

Theorem 2 (Communication complexity). *The expected number of times a broadcast message is forwarded before reaching all nodes is bounded by* $\left(1+\frac{\lambda}{1-\lambda}\right)n$.

Corollary 2 (Communication complexity). *If* $\lambda = \Theta\left(\frac{1}{\log_c n}\right)$, *then the expected number of times a broadcast message is forwarded before reaching all nodes is bounded by* $n + O\left(\frac{n}{\log_c n}\right)$.

Based on the above analysis, when $\lambda = \Theta\left(\frac{1}{\log_c n}\right)$, TBA achieves near optimal performance with hop complexity $O(\log_c n)$ and communication complexity $n + O\left(\frac{n}{\log_c n}\right)$. In order to pick $\lambda = \Theta\left(\frac{1}{\log_c n}\right)$, a node needs a rough knowledge about the values of n and c. Both of them can be estimated by the random sampling approach introduced in Section 3.1.

5. Simulation

In this section, we first use simulations to evaluate the performance of the ticket-based search algorithm proposed in Section 3, and then evaluate the performance of the broadcast algorithm in Section 4.

5.1. Setup

A Gnutella-like overlay network is constructed among a group of ultrapeer nodes. The default number of ultrapeer nodes in the simulations is 50,000, but networks of different sizes will also be studied. We assume the nodes come from all over the Internet, rather than clustered in a few places. In [33], Mukherjee found that the end-to-end packet delay on the Internet can be modeled by a shifted Gamma distribution, which is a long-tail distribution. The shape parameter varies from approximately 1.0 during low loads to 6.0 during high loads on the backbone. In this paper we set the shape parameter to be 4.0 and the average packet delay for one overlay link to be 100 ms, including the nodal processing time.

The recent work [34] by Stutzbach and Rejaie showed that the top-level node degree of an ultrapeer in Gnutella does not follow the power-law distribution even though that was claimed by some previous studies. Instead, the node degrees concentrate in the range of 1–30, with a spike at 30, which is the default maximum degree of the most popular Gnutella implementation, LimeWire. We ignore the less-significant tail distribution after 30, which is the result of other less-popular Gnutella implementations. For any value in the range [1..30], there are a significant number of nodes having that degree. Since no mathematical model has been proposed to fit the observed degree curve, in our simulations we choose a uniform degree distribution in the range [1..29] for two thirds of nodes and approximate the spike at 30 by making one third of nodes to have that degree. The average node degree is therefore 20. Beside the above default configuration, some simulations will study networks with other average node degrees, where the maximum degree is other than 30.

5.2. Evaluation of search performance

We implement four search algorithms: *Ticket-based search*, *TTL-constrained flooding*, *TTL-constrained limited-degree (LD) flooding*, *Random walk*. Ticket-based search combines ticket-based solution, random sampling solution, and moving anchor solution in Section 3. By default, $\lambda = 0.15$, and $\alpha = 2$ in the ticket-based solution, but we will study how different α values will affect the search performance. TTL-constrained flooding, or simply *TTL flooding*, is the algorithm used in Gnutella. TTL-constrained limited-degree (LD) flooding, or simply *TTL LD-d flooding*, forwards a received lookup message to only a fixed number d of randomly selected neighbors. The parameters for random walk are the same as in [35]. The source node sends 32 probes, each of which follows a random path and queries the source node after every four hops until the file is found.

The timeout between consecutive rounds of TTL flooding can dramatically increase the search time. The experiments in [24] showed that, for Gnutella lookups that return 10 or fewer results (which means the searched files are rare items with only a small number of replications), 50 s on average elapsed before the first result came back. In Fig. 4, our simulation demonstrates the same problem. The search times of TTL flooding become very large if the timeout period is

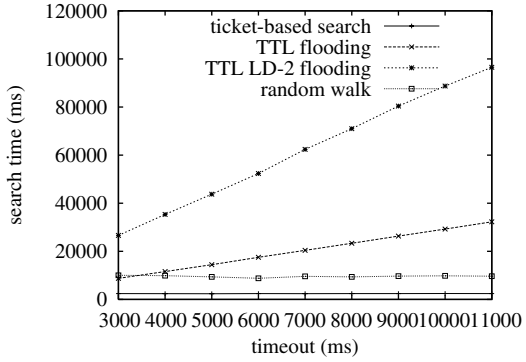

Fig. 4. Random sampling solution reduces search time of ticket-based search.

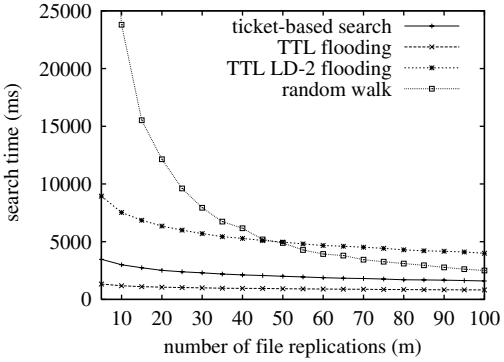

Fig. 6. Search time with respect to m.

large. Random walk does not use timeout. Ticket-based search avoids timeout by random sampling solution, which significantly reduces the search time. To make a sensible comparison in the rest of the simulations, we implement random sampling solution for TTL flooding and TTL LD flooding as well. Without specifying otherwise, the default number of replications of the searched file is 25.

The performance metrics that we use to evaluate the search algorithms include (1) *search overhead*, which is measured by the average number of times that a lookup message is forwarded before the file is located, and (2) *search time*, which is measured by the average time that elapses before the file is located.

We compare the performance of the four search algorithms. The results show that the ticket-based search algorithm achieves a favorable tradeoff between search overhead and search time.

5.2.1. Performance w.r.t m

Figs. 5 and 6 compare the algorithms in terms of search overhead and search time, with respect to the number m of replications of the searched file. m ranges from 5 to 100. TTL flooding is the worst in terms of search overhead but the best (after random sampling solution is implemented) in terms of search time. Random walk is the best in terms of search overhead but the worst in terms of search time when m is small. However, the search time of random walk reduces quickly when m is large. TTL flooding is too high in search overhead; random walk is too high in search time for small m. Ticket-based search and TTL LD-2 flooding make tradeoff between the two extremes, but the former is far better than the latter in terms of search time. We do not plot the performance of TTL LD-d flooding for other d values. Their performances lay between TTL LD-2 flooding and TTL flooding. When d increases, the search overhead becomes larger and the search time becomes smaller.

5.2.2. Performance w.r.t n

Figs. 7 and 8 compare the algorithms in terms of search overhead and search time with respect to the number n of nodes in the network. The search overhead of TTL

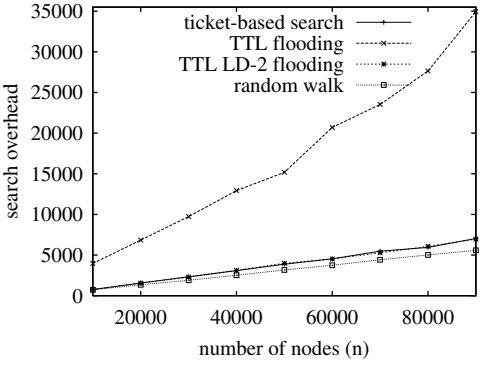

Fig. 7. Search overhead with respect to n.

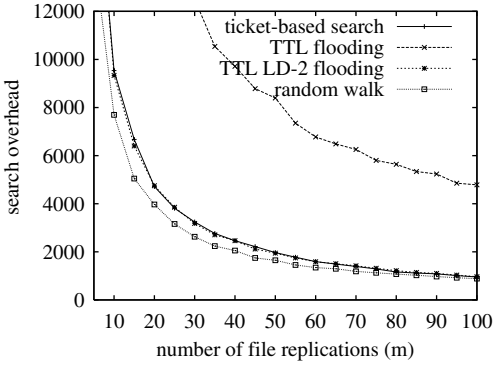

Fig. 5. Search overhead with respect to m.

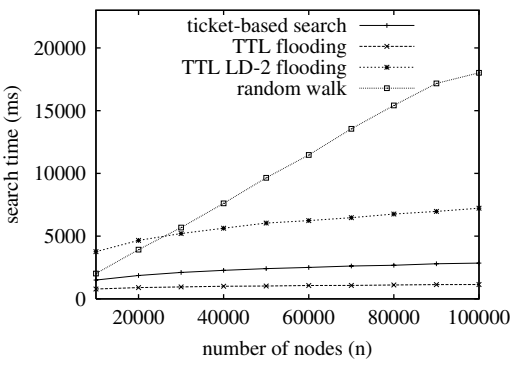

Fig. 8. Search time with respect to n.

flooding grows much faster than other algorithms when n increases. On the other hand, the search time of random walk grows much faster than other algorithms. Again the ticket-based search algorithm makes a favorable tradeoff, with both modest search overhead and modest search time when comparing with the best values.

5.2.3. Performance w.r.t node degree

Figs. 9 and 10 compare the algorithms with respect to the average node degree in the network. The algorithms are largely insensitive to the change of node degree, except that the overhead of TTL flooding generally grows at a significant rate. That is because larger node degree means larger discrete steps in overhead increment, which tends to cause larger mean overhead as explained in Section 3. However, the overhead of TTL flooding stays about the same for the node degrees between 15 and 21. That is because the reduced search depth (maximum TTL needed) fully compensates the increased search width (node degree) in this range.

5.2.4. Impact of α

We show the impact of α on the performance of ticket-based search in Figs. 11 and 12. Recall that α is the rate at which the number of tickets increases in consecutive search rounds. A larger α value means coarser discrete steps in overhead increment, which results in larger average search overhead, but smaller search time because the algorithm reaches a certain number (Ω) of nodes in a less number

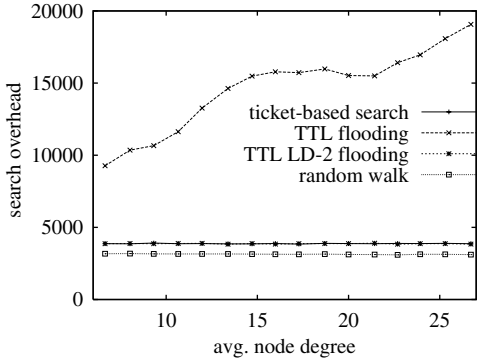

Fig. 9. Search overhead with respect to avg. node degree.

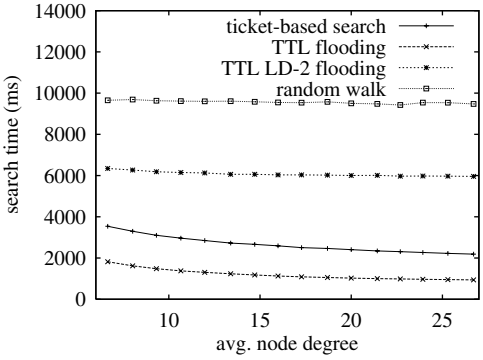

Fig. 10. Search time with respect to avg. node degree.

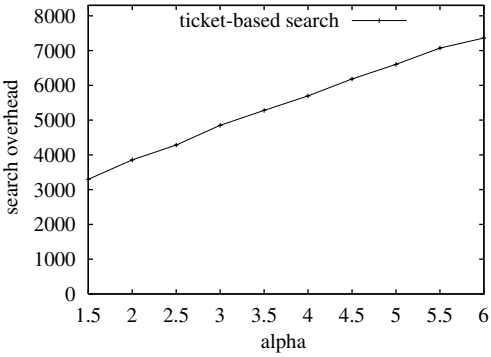

Fig. 11. Search overhead with respect to α.

of search rounds. We pick $\alpha = 2$ in all other simulations because it seems to make a reasonable balance between search overhead and search time.

5.3. Evaluation of broadcast performance

We also implement four broadcast algorithms: *ticket-based broadcast*, *flooding*, *probabilistic flooding*, and *limited-degree (LD) flooding*. Ticket-based broadcast is what we proposed in Section 4. Flooding is one of the most popular broadcast algorithms on the IP networks; reverse path forwarding is one example. In our case, each node forwards a copy of a message to all neighbors when it receive the message for the first time. In order to control the overhead, probabilistic flooding forwards a copy of the message to each neighbor with a certain probability, which means the message is not forwarded to all neighbors unless the probability is one. On the other hand, LD flooding forwards a copy of the message to only a fixed number of neighbors.

The performance metrics that we use to evaluate the broadcast algorithms include (1) *broadcast overhead*, which is measured by the number of times that a broadcast message is forwarded, and (2) *number of nodes reached*. Not all algorithms under comparison can reach every node in the network.

In Figs. 13 and 14, we compare the performance of the four broadcast algorithms in terms of broadcast overhead and number of nodes reached, respectively. In ticket-based broadcast, $\lambda = 0.25$. In probabilistic flooding, the probability

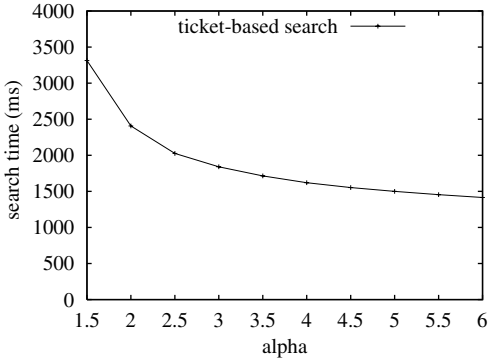

Fig. 12. Search time with respect to α.

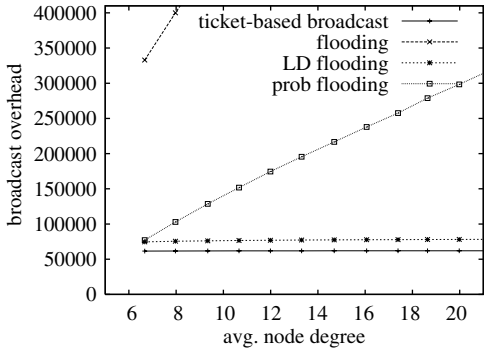

Fig. 13. Comparison of broadcast overhead.

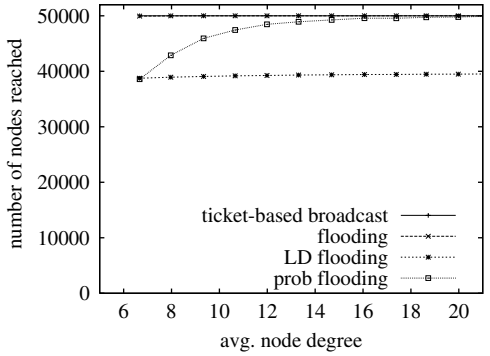

Fig. 14. Number of nodes reached by broadcast.

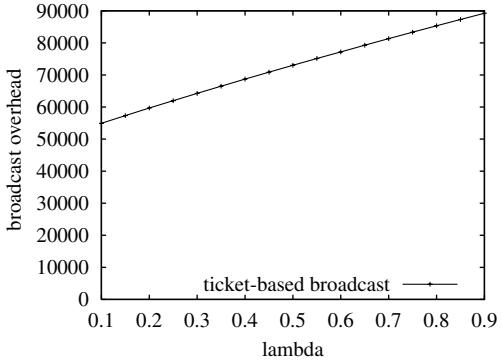

Fig. 15. Broadcast overhead with respect to λ.

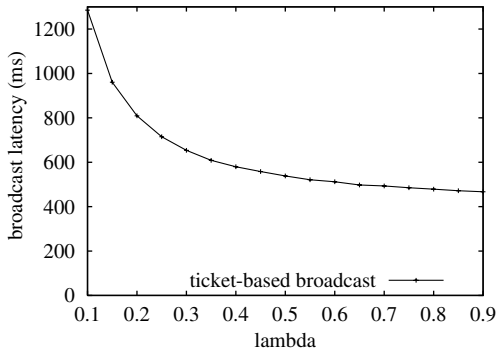

Fig. 16. Broadcast latency with respect to λ.

for a node to forward a message to a neighbor is 30%. In LD flooding, each node forwards a message to two neighbors.

Fig. 13 shows that ticket-based broadcast has the least overhead, and flooding has the highest. The overheads of flooding and probabilistic flooding grow quickly when the node degree increases. Ticket-based broadcast has smaller overhead because its segmented ring traversal avoids the heavy collision (duplicate visits) that would happen in the final stage of flooding.

One important thing to be pointed out is that probabilistic flooding and LD flooding cannot guarantee to reach every node because they traverse the graph without using all available (directed) links. As shown in Fig. 14, LD flooding can only reach less than 80% of all nodes, and probabilistic flooding reaches more nodes when the average node degree is larger. These two algorithms cannot be used if the application requires that a broadcast message must reach all nodes.

We show the impact of λ on the performance of ticket-based broadcast in Figs. 15 and 16. Recall that λn is the initial number of tickets used in Phase one. The broadcast overhead increases linearly with λ, agreeing with Theorem 2. The broadcast latency decreases non-linearly with λ. Therefore, to make a favor tradeoff, λ should not be too large. A value in the range of [0.2..0.4] is appropriate.

6. Conclusion

We identify a number of performance problems in the search algorithms of the current unstructured P2P networks, and propose a new ticket-based search algorithm to solve these problems. The new algorithm can efficiently search both popular and unpopular files without knowing which files are popular and which are not. The algorithm can also be modified for efficient broadcasting. We use simulations to evaluate the performance of the new techniques, in comparison with the existing ones.

Acknowledgement

This work is in part supported by the National Natural Science Foundation of China under grant 60573142 and by Shanghai Leading Academic Disclipline Project (Project Number T0502).

Appendix A. Proof

Let λn be the number of tickets used in Phase one, T be the number of nodes visited in this phase, called *phase-one nodes*, Q_i be the number of nodes that are i or less hops away from the lookup source, and q_i be the number of nodes that are exactly i hops away.

Lemma 1. *The number of hops a broadcast message travels from the source to any phase-one node is bounded by* $\log_c(\lambda n)$.

Proof. Consider an arbitrary phase-one node x. Let l be the number of hops the message travels from the source to x. At each hop, the number of tickets carried by the message is consumed by one and then split to a fraction of $\frac{1}{c}$.

Specifically, at the first hop, the number of tickets carried in the message sent by the source is no more than $\lceil \lambda n/c \rceil$. At the second hop, the number of tickets in the message is no more than $\lceil (\lceil \lambda n/c \rceil - 1)/c \rceil \ldots$ When x receives the message, the number of tickets must be at least one. Therefore,

$$\begin{aligned} (\lceil \ldots \lceil (\lceil \lambda n/c \rceil - 1)/c \rceil \ldots \rceil - 1)/c &\geqslant 1 \\ \tfrac{\lambda n}{c^l} &\geqslant 1 \\ l &\leqslant \log_c n \qquad \square \end{aligned}$$

Lemma 2

$$Q_i \leqslant \sum_{j=0}^{i} c^j$$

Proof. The number of nodes that are i hops away is bounded by the total number of neighbors from nodes that are $i-1$ hops away.

$$q_i \leqslant q_{i-1} c$$

Recursively applying the above inequality, we have

$$q_i \leqslant q_0 c^i = c^i$$

Hence,

$$Q_i = \sum_{j=0}^{i} q_i \leqslant \sum_{j=0}^{i} c^j \qquad \square$$

Lemma 3

$$E(T) > \left(1 - \lambda\left(\frac{c}{c-1}\right)^2\right)\lambda n$$

Proof. At the ith hop of Phase one, copies of the message are forwarded from nodes $(i-1)$-hops away to their neighbors. When a copy reaches the receiver, in worst case, there are no more than Q_i nodes in the network that have received the message. The probability for the copy to reach a node that has received the message is[3]

$$\begin{aligned} p(i) &\leqslant \frac{Q_i - 1}{n-1} \\ &< \frac{\sum_{j=0}^{i} c^i}{n} \end{aligned}$$

The total number of tickets carried in all copies of the message is bounded by λn. Let d_i be the number of tickets that are dropped at the ith hop.

$$E(d_i) \leqslant p(i)\lambda n < \lambda \sum_{j=0}^{i} c^i$$

By Lemma 1, the maximum number of hops a message travels is bounded by $\log_c(\lambda n)$. Let D be the total number of dropped tickets during the broadcast of a message.

$$E(D) = E\left(\sum_{i=0}^{\log_c(\lambda n)} d_i\right) < \lambda \sum_{i=0}^{\log_c(\lambda n)} \sum_{j=0}^{i} c^i < \left(\frac{c}{c-1}\right)^2 \lambda^2 n$$

Each ticket that is not dropped must be consumed by a phase-one node. Therefore,

$$\begin{aligned} T &= \lambda n - D \\ E(T) &= \lambda n - E(D) \\ &> \lambda n - \left(\frac{c}{c-1}\right)^2 \lambda^2 n \\ &= \left(1 - \lambda\left(\frac{c}{c-1}\right)^2\right)\lambda n \qquad \square \end{aligned}$$

Theorem 1 (*Hop complexity*). *The expected number of hops for a multicast message to reach any node is bounded by*

$$\log_c(\lambda n) + \frac{1}{2\left(1 - \lambda\left(\frac{c}{c-1}\right)^2\right)\lambda}$$

Proof. By Lemma 1, Phase one terminates in no more than $\log_c(\lambda n)$ hops. By Lemma 3, after Phase one, the ring is partitioned into more than $\left(1 - \lambda\left(\frac{c}{c-1}\right)^2\right)\lambda n$ segments. Phase two delivers the message in parallel along these segments. The expected length of a segment is bounded by

$$\frac{n}{\left(1 - \lambda\left(\frac{c}{c-1}\right)^2\right)\lambda n} = \frac{1}{\left(1 - \lambda\left(\frac{c}{c-1}\right)^2\right)\lambda}$$

In Phase two, the average number of hops for the message to reach a node is half of the segment length. Combining Phase one and Phase two, the expected number of hops for a multicast message to reach any node is bounded by

$$\log_c(\lambda n) + \frac{1}{2\left(1 - \lambda\left(\frac{c}{c-1}\right)^2\right)\lambda} \qquad \square$$

Corollary 1 (Hop complexity). *If* $\lambda = \Theta(\frac{1}{\log_c n})$ *and* $c \geqslant 2$, *then the expected number of hops for a multicast message to reach any node is* $O(\log_c n)$.

Proof. By Theorem 1, the upper bound on the expected number of hops is

$$\begin{aligned} &\log_c(\lambda n) + \frac{1}{2\left(1-\lambda\left(\frac{c}{c-1}\right)^2\right)\lambda} \\ &= (\log_c n - \Theta(\log_c \log_c n)) + \frac{1}{2\left(1-\Theta\left(\frac{1}{\log_c n}\right)\left(\frac{c}{c-1}\right)^2\right)\Theta\left(\frac{1}{\log_c n}\right)} \\ &= \Theta(\log_c n) + \frac{1}{\Theta\left(\frac{1}{\log_c n}\right)} \\ &= \Theta(\log_c n) + \Theta(\log_c n) = \Theta(\log_c n) \qquad \square \end{aligned}$$

Theorem 2 (Communication complexity). *The expected number of copies of a multicast message that are transmitted is bounded by* $(1 + \lambda)n$.

[3] "Minus one" is because the sender will not send the copy to itself.

Proof. In Phase one, each copy of the message forwarded from one node to another causes at least one ticket to be either consumed or dropped. Therefore, the number of copies forwarded in Phase one is bounded by the number of tickets, which is λn. Phase two sends no more than n messages. Therefore the expected number of copies that are transmitted for a multicast message is $(1+\lambda)n$. □

Corollary 2 (Communication complexity). *If* $\lambda = \Theta\left(\frac{1}{\log_c n}\right)$, *then the expected number of copies of a multicast message that are transmitted is bounded by* $n + O\left(\frac{n}{\log_c n}\right)$.

Proof. Directly from Theorem 2. □

References

[1] Gnutella, <http://gnutella.wego.com/>.
[2] KaZaA, <http://www.kazaa.com/>.
[3] FastTrack, <http://www.fasttrack.nu/>.
[4] N. Sarshar, P.O. Boykin, V.P. Roychowdhury, Percolation search in power law networks: Making unstructured peer-to-peer networks scalable, in: Proc. P2P'04, 2004.
[5] Z. Zhang, Y. Tang, S. Chen, Speed up queries in unstructured peer-to-peer networks, in: Proc. ICC'07, 2007.
[6] C. Plaxton, R. Rajaraman, A. Richa, Accessing nearby copies of replicated objects in a distributed environment, in: Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), June 1997.
[7] P. Druschel, A. Rowstron, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, in: Proc. 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), November 2001.
[8] B. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Department, 2001.
[9] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proc. ACM SIGCOMM'01, August 2001.
[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proc. ACM SIGCOMM'01, August 2001.
[11] L. BarriFre, P. Fraigniaud, E. Kranakis, D. Krizanc, Efficient routing in networks with long range contacts, in: Proc. 15th International Conference on Distributed Computing, October 2001.
[12] P. Maymounkov, D. Mazieres, Kademlia: a peer-to-peer information systems based on the XOR metric, in: Proc. IPTPS 2002, March 2002.
[13] D. Malkhi, M. Naor, D. Ratajczak, Viceroy: a scalable and dynamic emulation of the butterfly, in: Proc. ACM PODC'02, July 2002.
[14] K.P. Gummadi, R. Gummadi, S.D. Gribble, S. Ratnasamy, S. Shenker, I. Stoica, The impact of DHT routing geometry on resilience and proximity, in: Proc. ACM SIGCOMM 2003, August 2003.
[15] P. Fraigniaud, C. Gavoille, The Content-Addressable Network D2B, Technical Report 1349, LRI, Univ. of Paris-Sud, France, January 2003.
[16] F. Kaashoek, D.R. Karger, Koorde: a Simple Degree-Optimal Hash Table, in: Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), February 2003.
[17] G.S. Manku, Routing networks for Distributed Hash Tables, in: Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC), June 2003.
[18] G.S. Manku, M. Bawa, P. Raghavan, Symphony: distributed hashing in a small world, in: Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS), March 2003.
[19] G.S. Manku, M. Naor, U. Wieder, Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks, in: Proc. 36th ACM Symposium on Theory of Computing (STOC), June 2004.
[20] S. Sen, J. Wang, Analyzing peer-to-peer traffic across large networks, ACM SIGCOMM Internet Measurement Workshop, August 2002.
[21] J. Ritter, Why Gnutella can't Scale. No, Really, <http://www.tch.org/gnutella.html/>.
[22] M. Ripeanu, A. Iamnitchi, I. Foster, Mapping the Gnutella network, IEEE Internet Comput. J. Special Issue on Peer-to-Peer Networking, 6 (1), 2002.
[23] S. Saroiu, K.P. Gummadi, R.J. Dunn, S.D. Gribble, H.M. Levy, An analysis of internet content delivery systems, in: Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI), December 2002.
[24] B.T. Loo, R. Huebsch, I. Stoica, J.M. Hellerstein, The case for a hybrid P2P Search infrastructure, in: Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS), February 2004.
[25] R. Zhang, Y.C. Hu, Assisted peer-to-peer search with partial indexing, in: Proc. INFOCOM'05, 2005.
[26] Gnutella_Ultrapeers, <http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm/>.
[27] A. Kumar, J. Xu, E. W. Zegura, Efficient and scalable query routing for unstructured peer-to-peer networks, in: Proc. INFOCOM'05, March 2005.
[28] D. Tsoumakos, N. Roussopoulos, Adaptive probabilistic search for peer-to-peer network, in: Proc. 3rd International Conference on Peer-to-Peer Computing (P2P'03), September 2003.
[29] S. Jiang, L. Guo, X. Zhang, LightFlood: an efficient flooding scheme for the file search in unstructured peer-to-peer system, in: Proc. ICPP'03, 2003.
[30] H. Wang, T. Lin, On efficiency in searching networks, in: Proc. INFOCOM'05, 2005.
[31] H. Jiang, S. Jin, Exploiting dynamic querying like flooding techniques in unstructured peer-to-peer networks, in: Proc. ICNP'05, 2005.
[32] S. Chen, K. Nahrstedt, Distributed quality-of-service routing in ad-hoc networks, IEEE J. Selected Areas in Communications, Special Issue on Ad-Hoc Networks, 17 (8) (1999).
[33] A. Mukherjee, On the dynamics and significance of low frequency components of internet load, Internetworking: Res. Exp. 5 (4) (1994) 163–205.
[34] D. Stutzbach, R. Rejaie, Characteristing the two-tier Gnutella topology, in: Proc. SIGMETRICS'05, June 2005.
[35] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proc. ACM SIGMETRICS'02, June 2002.

Shiping Chen received his B.S.degree in Electircal Engineering from JiangXi University of China in 1984. He received his M.S. and PH.D. degrees in Computer Science from Institute of Computing Technology of Chinese Academy Sciences and Fudan University in 1990 and 2006, respectively. He joined University of Shanghai for Science and Technology in 1990 and is currently a full professor in the computer science department. He is also the director of the network center of the university. His research interests include peer-to-peer networks, network communications, and database systems.

Zhan Zhang received his M.S. degree in computer science from Fudan University of China in 2003, and received his Ph.D degree in Computer and Information Science and Engineering from University of Florida in 2007. His research fields include overlay networks, wireless and sensor networks, and network security.

Shigang Chen received his B.S. degree in computer science from University of Science and Technology of China in 1993. He received M.S. and Ph.D. degrees in computer science from University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for three years before joining University of Florida as an assistant professor in 2002. His research interests include network security, peer-to-peer networks, and sensor networks. He received IEEE Communications Society Best Tutorial Paper Award in 1999. He was a guest editor for ACM/Baltzer Journal of Wireless Networks (WINET) and IEEE Transactions on Vehicle Technologies. He served as a TPC co-chair for the Computer and Network Security Symposium of IEEE IWCCC 2006, a vice TPC chair for IEEE MASS 2005, a vice general chair for QShine 2005, a TPC co-chair for QShine 2004, and a TPC member for many conferences including IEEE ICNP, IEEE INFOCOM, IEEE SANS, IEEE ISCC, IEEE Globecom, etc.

Baile Shi joined the Department of Computer and Information Technology of Fudan University in 1975. He was promoted to an associated and then full professor in 1980 and 1985, respectively. He was the department chair from 1985 to 1996. His research field is database theories and applications. He has published over 70 papers in the top Chinese journals and written more than 10 textbooks. He has won numerous awards, including one national science and technology advancement award, one Guanghua award, nine Shanghai science and technology advancement awards, and four textbook awards.