LightFlood: Minimizing Redundant Messages and Maximizing the Scope of Peer-to-Peer Search

Song Jiang, *Member*, *IEEE*, Lei Guo, *Member*, *IEEE*,
Xiaodong Zhang, *Senior Member*, *IEEE*, and Haodong Wang

Abstract—*Flooding* is a fundamental file search operation in unstructured peer-to-peer (P2P) file sharing systems, in which a peer starts the file search procedure by broadcasting a query to its neighbors, who continue to propagate it to their neighbors. This procedure repeats until a time-to-live (TTL) counter is decremented to 0. Flooding can seriously limit system scalability, because the number of redundant query messages grows exponentially during the message propagation. Our study shows that more than 70 percent of the generated messages are redundant in a flooding with a TTL of 7 in a moderately connected Gnutella network. Existing efforts to address this issue have been focused on limiting the use of the flooding operation. We propose a new flooding scheme, called *LightFlood*, with the objective of minimizing the number of redundant messages and retaining a similar message-propagating scope as that of the standard flooding. In the scheme, each peer keeps track of the connectivities of every immediate and next indirect neighbor peers, which can be acquired locally. *LightFlood* identifies the neighbor with the highest connectivity and uses the link to that neighbor to form a suboverlay within the existing P2P overlay. In *LightFlood*, flooding is divided into two stages. The first stage is a standard flooding with a limited number of TTL hops, where a message can spread to a sufficiently large scope with a small number of redundant messages. In the second stage, message propagating is only conducted along the suboverlay, significantly reducing the number of redundant messages. Our analysis and simulation experiments show that the *LightFlood* scheme provides a low-overhead broadcast facility that can be effectively used in P2P search. For example, compared with standard flooding with seven TTL hops, we show that *LightFlood* with an additional two to three hops can reduce up to 69 percent of the flooding messages and retain the same flooding scope. We believe that *LightFlood* can be widely used as a core mechanism for efficient message broadcasting in P2P systems due to its near-optimal performance.

Index Terms—Peer-to-peer system, file searching, overlay network, query flooding.

1 Introduction

A peer-to-peer (P2P) file sharing system is built as an overlay on the existing Internet infrastructure to provide file sharing service to a highly transient population of users (peers). Early systems such as Napster use a cluster of servers to store indices of participating peers. This centralized design and practice arouse the concerns of performance bottleneck and single point of failure. Researchers and practitioners have studied decentralized approaches in order to provide a scalable file sharing service. Instead of maintaining a huge index store in a centralized system, a decentralized system distributes all searching and locating loads across all participating peers. Among these systems, Gnutella is a representative. Although the decentralized approach addresses the overloading and reliability issue and is promising in building a highly scalable P2P system, its success is heavily dependent on an efficient mechanism to broadcast messages across a large population of peers. Reaching out to a large scope of peers is a fundamental procedure in unstructured and ad hoc P2P networks. For example, when a peer joins a P2P system, it will notify other peers of its participation by broadcasting a ping message so that others can respond to it with pongs. After the peer joins the system, it wants its queries to reach a large population in its neighborhood to ensure the quality of search service. Because there are no controls and no accurate information on the network topologies and locations of desired files in ad hoc systems, message broadcasting becomes essential, and system scalability is directly affected by the efficiency of the mechanism.

The major existing mechanism for message broadcasting is flooding, in which a peer sends a message to its neighbors, who, in turn, forward the message to all their neighbors, except the message sender. Each message has a unique message ID. A message that has the same message ID as the one received previously by the same peer is considered a redundant message and will be discarded. Flooding is conducted in a hop-by-hop fashion counted by a time-to-live (TTL) count. A message starts off with its initial TTL, which is decremented by one when it travels across one hop. A message comes to its end either because it becomes a redundant message or because its TTL is decremented to 0.

- *S. Jiang is with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202. E-mail: sjiang@eng.wayne.edu.*
- *L. Guo is with Yahoo! Inc., 701 First Avenue, Sunnyvale, CA 94089. E-mail: lguo@yahoo-inc.com.*
- *X. Zhang is with the Department of Computer Science and Engineering, Ohio State University, Columbus, OH 43210. E-mail: zhang@cse.ohio-state.edu.*
- *H. Wang is with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23187. E-mail: wanghd@cs.wm.edu.*

Manuscript received 9 May 2006; revised 3 Aug. 2007; accepted 7 Aug. 2007; published online 11 Sept. 2007.
Recommended for acceptance by M. Singhal.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-01-2006.
Digital Object Identifier no. 10.1109/TPDS.2007.70772.

Comparatively, during the lifetime of a message, we call the sequence of initial hops of a message's path as *low hops*, whereas the rest of its hops, that is, the sequence of final hops of its path, are called *high hops*. We call this flooding procedure, which is widely used in P2P systems like Gnutella, *pure flooding* in the rest of this paper in order to distinguish from the flooding scheme that we propose.

Flooding has the following merits: modest latency (or response time), large coverage, and high reliability. A measurement-based study conducted in 2000 and 2001 shows that 95 percent of the peers in a Gnutella system could be reached within seven hops $(\mathrm{TTL} = 7)$ by flooding [11]. This is because more and more peers join to route the message in parallel while the flooding is going on, and the number of peers reached could be increased exponentially. The departure or failure of individual peers can hardly have a disruptive impact on the system ability to transmit messages in flooding, because all possible routes in the specified neighborhood are utilized simultaneously. For these merits, flooding is used widely in unstructured P2P systems. However, with the increasing popularity of P2P systems and the rapidly expanding system scales, we are facing a serious problem of flooding. Excessive traffic overhead is caused by a large number of redundant message forwarding, particularly in a system with a high-connectivity topology. When multiple messages with the same message ID are sent to a peer by its multiple neighbors, all, except for the first message, are considered as redundant messages. These redundant messages are pure overhead: they increase the network transfer and peer processing burden without enlarging the propagation scope. As a result, some researchers believe that fully decentralized Gnutella-style systems with flooding do not scale [14], [8], [12]. It was estimated that the total traffic on a Gnutella system of 50,000 nodes, where flooding search is used, accounted for about 1.7 percent of the total traffic over the US Internet backbone in December 2000 [11]. Considering that this volume is only the amount of messages for file search and does not include file transfer traffic, which is out of the Gnutella overlay, flooding search becomes a bottleneck for the scalability of unstructured P2P systems.

Although there are some proposed schemes restricting the use of flooding or withdrawing the system from a full decentralization to a hybrid architecture such as a supernode structure in the hope of alleviating the negative effect of flooding, these schemes still have to keep flooding as a major component to reach other peers in the ad hoc system. We will discuss this in Section 2.

Realizing the important role of flooding in ad hoc P2P systems and its problem, we propose an efficient flooding scheme, called *LightFlood*. *LightFlood* not only retains the merits of pure flooding but also can eliminate most of the redundant messages caused by pure flooding. Thus, *LightFlood* greatly enhances the scalability of Gnutella-style P2P systems. The design of *LightFlood* is motivated by two observations. First, the majority of redundant messages are generated within high hops. Second, the network coverage growth rates in low hops are much higher than those within high hops. We select a subset of links in a P2P overlay to form a suboverlay, which we call *FloodNet*, to connect all the participating peers. *FloodNet* is a tree-like network that uses the least number of existing P2P links to organize peers into a small number of low-diameter clusters.

The *LightFlood* scheme is divided into two stages. In the first stage, we let messages on their low hops be flooded by pure flooding in the P2P overlay (by giving a small TTL number). Those peers reached on the last hop of pure flooding $(\mathrm{TTL} = 0)$ become *seeds*, from which *FloodNet* is initiated for the second stage. The initial pure flooding ensures that a considerable number of *seeds* are dispersed across the overlay with a small number of redundant messages. The next stage of flooding in *FloodNet* ensures that most of the redundant messages caused by pure flooding within the rest of its hops are eliminated. The integration of these two stages retains the advantages of pure flooding: low latency, high coverage, and high reliability.

2 Related Work

To address the flooding problem in Gnutella-style P2P systems, researchers and practitioners have proposed many solutions, which can be categorized into three types: 1) adaptive flooding, 2) locality-based flooding reduction, and 3) partially centralized location service.

Unlike pure flooding, which always starts with a fixed TTL from a peer to reach its neighborhood within its radius, adaptive flooding takes more dynamic factors into consideration to reduce the flooding range while maintaining the necessary search quality. For example, in the expanding ring [7] (or iterative deepening [16]), several successive flooding searches are initiated with increasing TTLs. After each flooding, the requesting peer has to wait for some period of time to collect response messages. The continuation of a flooding depends on whether enough responses have been received so far. Although the method might reduce the radius that a flooding has to reach in order to save traffic overhead and might reduce the latency of popular files, its performance could be uncertain for less popular files due to the repeated use of flooding. Directed BFS [16] selects a neighbor satisfying a specific criterion to send its query, which, in turn, starts regular flooding, in order to save cost, whereas many quality results still can be obtained. There is a list of potential criteria for neighbor selection, among which is choosing a neighbor that has the shortest message queue, the shortest latency, or the highest degree. The use of these criteria turns directed BFS into an adaptive flooding. Although a high-cost flooding plays a major role in their search operations, schemes using local adaptive decisions are limited in reducing the query cost. Our *LightFlood* promises to significantly reduce the overhead of flooding, regardless of the fluctuating system factors such as file distribution and local connecting conditions. Once integrated into these adaptive schemes, *LightFlood* can be very effective by turning high-cost flooding into a lightweight operation.

To reduce the use of flooding and improve search efficiency, a certain locality has been assumed and exploited by abstracting hints from a past search. Interest-based locality [4], [5], [14] states that if peer A has a particular piece of content that peer B is interested in, then it is likely that A will have other pieces of content that B is also interested in. Based on this assumption, a peer selects others that have previously provided results to its queries as peers

TABLE 1
The DSS Clip2 Topology Traces Used in Our Simulation Experiments

Topology	Original Name	Average Degree	Average 2-hop Neighboring Peers	Number of Peers
T1	graph052701104502.xml	3.4	34.1	42822
T2	graph050301100618.xml	4.7	54.8	28895
T3	graph183011126.xml	5.4	129.8	21781

with common interests and creates shortcut links to these peers. In this scheme, common interests are detected through flooding, and flooding on the original overlay is used whenever searching through shortcut links fails. A similar scheme is also proposed in [10]. Directed BFS becomes a locality-based flooding reduction technique when the criteria for choosing a neighbor find a neighbor returning the greatest number of results previously, having the shortest average time to satisfaction, or having the smallest average number of hops taken by results in the last 10 queries [16]. Similarly, subsequent query forwarding from the neighbor is also conducted by flooding. Apparently, flooding still plays an important role in these schemes, although history information could be useful to reduce the use of flooding. Low-cost flooding is still essential to achieve the goal of low overhead in the worst case.

The third prevalent solution adopts superpeers to provide a partially centralized location service [15], like Kazaa [1] and the current Gnutella implementation. A superpeer is a node that acts as a centralized server to a subset of clients. It maintains the indices of its client peers and conducts searching and locating on behalf of its clients among superpeers. These superpeers connect to each other, forming a pure Gnutella-style network. Although a partially centralized location service could alleviate the flooding burden on fully decentralized systems, the number of clients that a superpeer can serve is limited because of the concerns on superpeer overload, system security, and reliability. For example, an online statistic [2] shows that a supernode has an average of five to seven clients in Gnutella, making its supernode network size above 20,000. With the rapidly expanding scale of the systems, the inefficiency of flooding in a superpeer network itself remains a grave concern to be addressed.

Different from the aforementioned and other existing content search algorithms in unstructured P2P networks, which exploit the common interests and locality of content among peers or utilize different heuristics for improving the search quality, *LightFlood* is a general-purpose flooding scheme for P2P networks, which utilizes the topology property of unstructured P2P networks to broadcast messages in the entire P2P network with low overhead. By constructing a *FloodNet* over the P2P network, *LightFlood* well exploits the connectivity characteristics in a P2P network topology to eliminate most of the redundant messages without compromising the flooding coverage and propagation delay. *LightFlood* is a lightweight broadcast scheme, which can be used as a basic operation and is a core technique in large-scale P2P networks. First, *LightFlood* can be the building block of other P2P search algorithms, since all P2P search algorithms use flooding more or less. Second, the application of *LightFlood* is not limited to the P2P search. The message broadcast in *LightFlood* can be a query, a notification, or other information for broadcasting. In this paper, we use content search as an example of P2P flooding. However, *LightFlood* can be used for any broadcasting purposes.

3 Flooding versus Hops

Recall that flooding is conducted in a hop-by-hop fashion. With the increment of hops, more new peers are reached, and more forwarded messages are generated. Part of the messages are redundant. In this section, we study the regularity of changes in coverage and the number of redundant messages with the increase in hops. Our observations show that pure flooding is efficient only in earlier stages (with low hops). In latter stages, a large number of redundant messages are generated, whereas the coverage growth rate is much less than that of the initial stages. This motivates us to develop an efficient flooding scheme that only generates minimal redundant messages and provides the maximum search scope at the same time.

We use Gnutella topologies collected during the first six months of 2001, which are provided by Clip2 Distributed Search Solutions [3], to study the flooding behavior. In this paper, we mainly use three topology traces, which are listed in Table 1, as the representatives of a variety of topology sizes and average connectivity degrees. In the table, "Original Name" refers to the trace filename used in [3], whereas "Average Two-hop Neighboring Peers" denotes the average of the number of peers within two hops for each peer (see Section 4 for details). The differences among the topologies are due to the different subsets of peers and collection times to "snapshot" the overlay.

We use these connectivity graphs to simulate the behavior of flooding in real-life P2P topologies, whose connectivity degrees follow a two-segment power-law distribution (see [11] for details). In our simulation experiments, a query is sent from each peer in the network with a given TTL value. Usually, a TTL of 7 can achieve at least a 95 percent coverage. Then, for each flooding hop h of a query, we collect the number of new peers reached P_h and the number of forwarded messages generated M_h. We know that it takes at least N messages to reach N new peers. Thus, the number of redundant messages on hop h is $M_h - P_h$. We average the P_h and $M_h - P_h$ over all the queries sent from each peer in

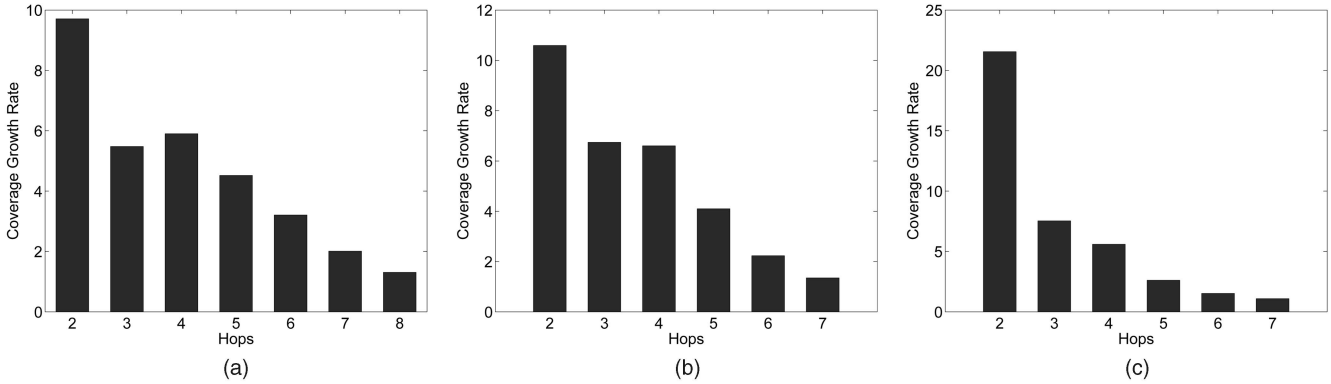


Fig. 1. Coverage growth rate of message flooding. The bar associated with "i" hops $(i = 2, 3, \ldots, 8)$ represents the growth rate of coverage between the initial i hops and $i-1$ hops. (a) Topology T1. (b) Topology T2. (c) Topology T3.

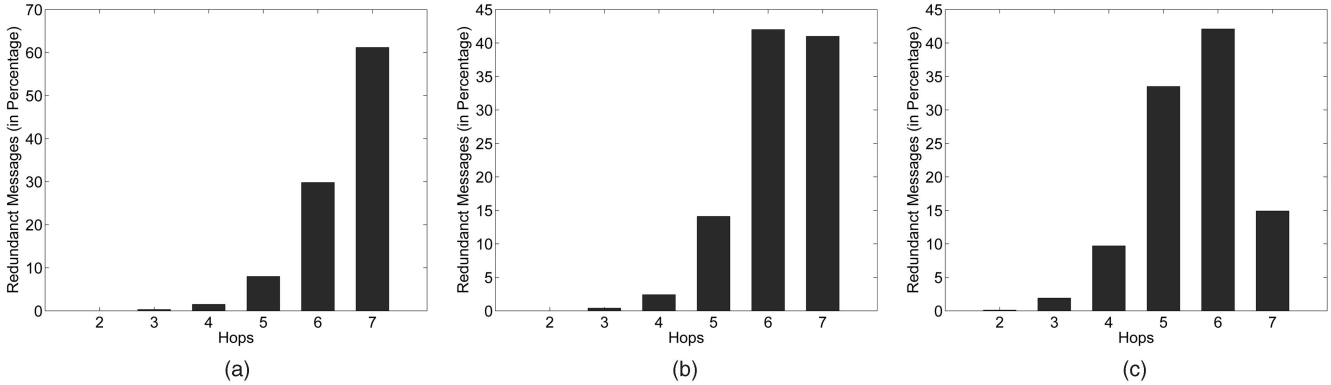


Fig. 2. Redundant message distribution. The bar associated with "i" hops $(i = 2, 3, \ldots, 7)$ represents the percentage of redundant messages generated on the ith hop in all redundant messages generated in the seven-hop floodings. (a) Topology T1. (b) Topology T2. (c) Topology T3.

the topologies for each flooding hop. To observe the growth rate of message flooding coverage, we list $\sum_{i=1}^{h} P_i / \sum_{i=1}^{h-1} P_i$ for $h = 2, 3, \ldots, 8$ in Fig. 1 for each topology. In the figure, we an see that the peer coverage growth rate reduces quickly with the increment of hops. This phenomenon can be explained with the following reasons:

1. Although the overlay topology of a Gnutella network follows a heavy-tailed distribution, it is not a *real* power-law network but is a two-segment power-law distribution [11], since the high-connectivity nodes in a Gnutella network are not *powerful* enough. According to the work on searching in power-law networks [9], the average number of nodes covered by h-hop broadcast (or flooding) can be given as follows:

$$z_m = \left(\frac{z_2}{z_1}\right)^{m-1} z_1, \tag{1}$$

where z_m is the average number of new peers discovered on the mth hop flooding, and z_1 and z_2 stand for the average numbers of neighbors within one hop and two hops from the peers, respectively. This formula tells us that the coverage growth should be z_2/z_1 for each additional hop. However, we only find this trend on the second hop in three topologies. On the second hop, the growth rates are 9.7, 10.6, and 21.6 for topologies T1, T2, and T3, respectively. The growth rates for the third and fourth hops have only half the number expected. The reason is that due to the two-segment power-law distribution, the Gnutella networks do not have very powerful nodes to be exploited to maintain the growth rate during the flooding on the third and fourth hops.
2. Due to the limited size of the Gnutella networks (only more than 40,000 nodes in our traces), redundant query messages are prevalent in the entire network at the latter stage in a pure flooding. As a result, the coverage growth rate drops accordingly, with the redundant messages being discarded.

The distribution of redundant messages across the hops is shown in Fig. 2. We list the percentage of redundant messages on a specific hop over all redundant messages within seven hops. Apparently, the redundant messages generated within the initial four hops of flooding are much less than those within the latter stages. For example, the numbers of redundant messages within the second, third, and fourth hops combined are only 1.9 percent, 2.9 percent, and 10.7 percent of all the seven-hop redundant messages in topologies T1, T2, and T3, respectively. This is because at a larger hop, there are much more redundant messages generated than those when discovering new nodes.

Considering the large coverage growth rate and small overhead in terms of the percentage of redundant messages within the initial stages of flooding and poor coverage

efficiency within the latter stages of flooding, we propose a *FloodNet* suboverlay network. *FloodNet* can effectively reduce the redundant messages during the latter stages of flooding and achieve the same moderate coverage growth rate as pure flooding does.

4 Description of the *LightFlood* Scheme

If we build a spanning tree that connects all peers on a P2P overlay network, we can eliminate all redundant messages. However, the spanning tree approach has the following disadvantages in practice:

1. It is very costly and impractical to build and maintain a spanning tree on an unstructured P2P overlay network. The procedure of building a spanning tree over a large network takes a long time. Because unstructured P2P networks are self organized, peers may frequently come and go. Thus, the spanning tree has to be rebuilt each time there is a change in the network.
2. Broadcasting along the spanning tree of a well-connected network like Gnutella is not desirable due to the concern of long latency. Our simulation shows that it takes only seven hops to reach 95 percent of nodes with pure flooding, but it takes more than 30 hops to have the same coverage with a flooding on a randomly constructed spanning tree over the network.
3. Flooding on a spanning tree also has poor reliability, because unstructured P2P networks are so dynamic that any node on the flooding route could be disconnected at any time. A disconnected peer would make all downstream nodes unreachable in a considerable amount of time until the tree is rebuilt.

4.1 *FloodNet*: A Tree-Like Suboverlay

Inspired by the pros and cons of the pure-flooding scheme and the approach of flooding over a spanning tree, we propose *FloodNet*, a tree-like suboverlay for unstructured P2P networks. The idea behind *LightFlood* is to flood over *FloodNet* simultaneously from a group of peers that are randomly and uniformly located in the P2P networks. We call these peers *seeds*, which are identified by pure flooding from a source peer X with a TTL value of L. When $L = 1$, *seeds* are the immediate neighbors of peer X. Similarly, when $L = 4$, *seeds* are all four-hop neighbors of peer X.

There are several principles in constructing *FloodNet*: 1) because the system is designed to be fully autonomous and highly dynamic, only local information can be inexpensively available and is used for the construction, 2) to increase the coverage of a flooding with a given TTL, the topology diameter of *FloodNet* should be low, and 3) because of the high transiency of the system, *FloodNet* must be efficiently maintained. In addition to the three principles above, we allow *FloodNet* to be composed of multiple separated graph components. If the number of seeds is much larger than the number of separated components in *FloodNet*, these seeds will highly likely reach each of the components, and the coverage of flooding in *FloodNet* would be close to that of pure flooding on the network.

Based on these principles, we construct *FloodNet* this way:

1. Each peer informs its immediate neighbors of its connectivity degree.
2. Once the degrees of all its neighbors are known, a peer calculates the sum of its neighbors' degree, which is the number of the secondary neighbors of the peer.
3. Each peer informs its immediate neighbors of the number of its secondary neighbors.
4. After all information has been exchanged, a peer selects the neighbor that has the maximum secondary degree as its *father* and notifies its father.

Thus, there could be a peer without a father, because all its neighbors choose it as *father*. Note that the *FloodNet* construction could generate multiple unconnected tree-like components. In each component, the tree-root peer has the largest number of secondary neighbors within the scope of two hops from the root peer. In power-law networks, powerful nodes have the tendency to connect with each other, so the number of these unconnected components is expected to be small. However, our preliminary results in [6] show that the number of unconnected components is still too large if we use the number of immediate neighbors as the heuristic to construct *FloodNet*. This is because powerful nodes may not be necessarily connected with each other directly. Many of them are connected through a third party. By using the number of secondary neighbors as the heuristic for deciding on the parent-child relationship of the nodes, the number of unconnected components can be reduced significantly. For the topologies used in our experiments, that is, T1, T2, and T3, the numbers of unconnected components are 6, 3, and 2, respectively. Although examining peers of more hops away can further decrease the number of unconnected components, it still cannot ensure that *FloodNet* is fully connected, and it introduces additional overhead when the network topology changes. With the two-stage flooding scheme of *LightFlood*, examining peers with two hops is good enough to reach all unconnected components. Our evaluation shows that we have a significant performance gain over the original *LightFlood* scheme proposed in [6].

FloodNet can be constructed with little cost by using only local information. The number of secondary neighbors can be easily obtained locally, and its diameter is small, because links to high-degree peers are utilized. Furthermore, its maintenance cost is small: when a peer's immediate neighbors arrive or depart or their degrees change, the peer reevaluates the neighbor degrees, possibly selects a new father peer, and notifies the affected neighbors. Compared with the cost of updating supernode indices and interest-based locality, *FloodNet* can be maintained with little overhead.

It is possible that two or more neighbor nodes have the same number of secondary neighbors, and no other neighbor has more secondary neighbors than these nodes. As a result, a cycle may be formed while building *FloodNet*. A simple way

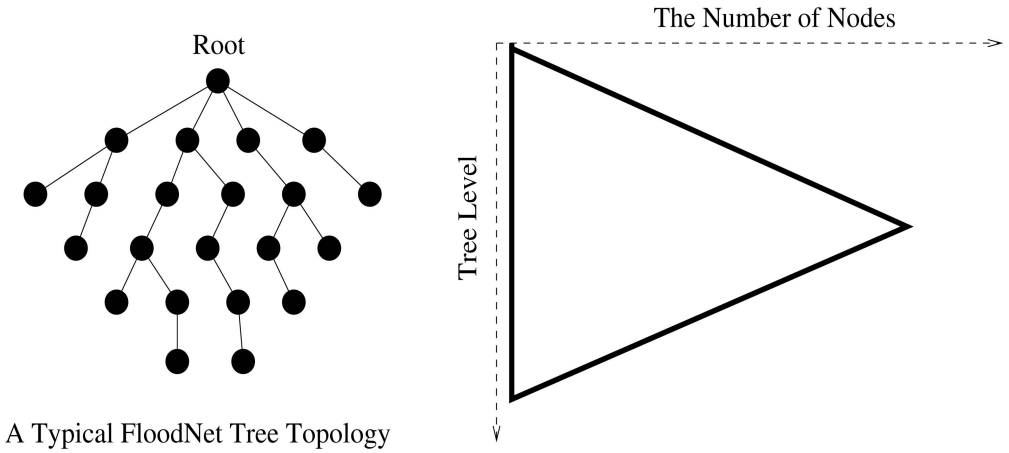


Fig. 3. The *FloodNet* tree structure has a "diamond" shape. Most nodes are concentrated in the middle level, and the number of nodes exponentially decreases when the level goes up or down.

of preventing this from happening is to use a tie-break mechanism when any two nodes have the same secondary degree. If two or more neighbors find that their secondary degrees are the same, the node who has the smallest node ID becomes the parent node of other nodes.

4.2 The Two-Stage *LightFlood* Scheme

There are two overlay networks in the system: the original P2P overlay and the new *FloodNet* as a suboverlay. Accordingly, the *LightFlood* scheme is divided into two stages. A *LightFlood* arrangement is expressed in terms of an ordered pair (M, N). In the first stage, the query message is broadcast in the original overlay network with a small TTL value of M. Recall that the first several steps of a pure flooding in the overlay network have the properties of high efficiency and fast coverage growth. During the first M hops, peers receive and process the query messages exactly as they do in the pure-flooding scheme. The difference comes when TTL turns to zero: instead of discarding the query message, the receiving peers turn themselves into *seeds*, which are the starting points of broadcasting in the second stage. In other words, when the receiving peers find that the TTL value of the query message becomes zero, they start the second stage immediately by broadcasting the message in *FloodNet* and setting the corresponding TTL to N. The first stage can be considered as the procedure of spreading *seeds*. In the second stage, these *seeds* sprout and efficiently pass the query messages to the every corner of the network.

4.3 *FloodNet* Maintenance

Since the unstructured P2P networks are self organized, an individual peer may come and go frequently. As a result, some nodes may be separated from *FloodNet*. Therefore, each node needs to update periodically its neighbor information. A small update interval can produce accurate information about the *FloodNet* topology, but frequently updating messages consumes extra bandwidth resource. A large update interval, however, has lower communication costs but may not reflect the latest network topology. The selection of update interval should follow the principle that the extra communication costs introduced by topology changes should not exceed the bandwidth savings from the *LightFlood* scheme. In reality, it is very difficult to find such a balanced point, because the network condition is dynamic, such as user query frequency and volatility of nodes.

However, due to the two-stage *LightFlood* scheme, a large update interval does not cause big performance degradation. Although a number of nodes may be disconnected from *FloodNet* if their parent node leaves, the user query message can still reach these portion of nodes, because many seeds can be created during the initial pure flooding of a message. Therefore, the *LightFlood* scheme is robust against volatile nodes. We will further confirm it by experiments in Section 6.

5 *FloodNet* Performance Analysis

In this section, we present the performance analysis of flooding in *FloodNet*. We make the following assumptions for the analysis:

1. Our simulation results show that there are only a small number of separated trees, and the dominant tree, the one that has the most nodes, holds at least 70 percent of the nodes. Therefore, we use a single tree in our analysis.
2. Except for the tree root, most of the nonleaf nodes are evenly distributed. Although there could be some powerful nodes, the number of such nodes is small. Most of the nodes have similar degrees.
3. The *FloodNet* tree has a *diamond* shape, in which most nodes are concentrated in the middle levels, and the number of node exponentially decreases when the level goes up or down (as shown in Fig. 3).
4. After several initial hops of pure flooding, *seeds* are evenly distributed in the *FloodNet* tree.

The assumption of a diamond-shaped tree is based on the examination of more than 20 Gnutella topology snapshots. Table 2 describes the *FloodNet* structures created for topologies T1, T2, and T3. We count the number of nodes on each tree level (the tree roots are on level 0) and list the number in the table. Obviously, most of the tree nodes are concentrated in the middle levels of the trees. Starting from the middle level (such as level 6), the number of nodes

TABLE 2
The Number of Peers at Each Level of the *FloodNet* Structure in Topologies T1, T2, and T3

Topology	T1	T2	T3
level 0	1	1	1
level 1	477	457	981
level 2	928	1043	2178
level 3	4295	3014	4866
level 4	4914	3613	2502
level 5	6696	4947	1683
level 6	5398	5359	1600
level 7	3307	3727	1503
level 8	1774	2091	1003
level 9	968	1075	432
level 10	411	421	138
level 11	165	117	56
level 12	56	14	11
level 13	30	0	0
Total	29420	25879	16954

gradually decreases when the level either decreases or increases.

5.1 Seed Sprouting

As we have described in the previous section, we use the term *seeds* to describe the nodes that are reached on the last hop of a pure flooding of TTL M in the first stage. Then, the seeds start "sprouting" in the *FloodNet* suboverlay in the second stage. Although the coverage growth for a single seed sprouting is not significant, pure flooding of the first stage can produce a sufficient number of seeds to sprout simultaneously.

We denote n as the number of nodes in the tree, which is equal to the number of peers in the P2P network. Thus, the total number of degrees of the tree is $2(n-1)$. When n is large, $2(n-1) \approx 2n$. Given any node in the tree, we assume that the probability for this node to have at least one child is q, the probability for a node to be a leaf is p, and $p+q=1$. Then, we use Deg to denote the average degrees for the nonleaf nodes and obtain

$$2n = p \cdot n + q \cdot n \cdot Deg \Longrightarrow Deg = \frac{2-p}{q} = \frac{2-p}{1-p}. \quad (2)$$

Fig. 4 illustrates a typical flooding procedure in a *FloodNet* tree. Let us examine the coverage growth in the *FloodNet* flooding. After the first hop, the number of new nodes reached is

$$q \cdot (Deg-1) = q \cdot Deg - q = 2 - p - q = 1. \quad (3)$$

After the second-hop flooding, the number of new nodes is

$$Deg - 1 + q \cdot (Deg-1) \cdot q \cdot (Deg-1) = Deg - 1 + 1 = Deg. \quad (4)$$

The number of nodes discovered on the third-hop flooding will be

$$\begin{aligned} &Deg - 1 + (Deg-2) \cdot q \cdot (Deg-1) + (q \cdot (Deg-1))^3 \\ &\quad = Deg + (Deg-2). \end{aligned} \quad (5)$$

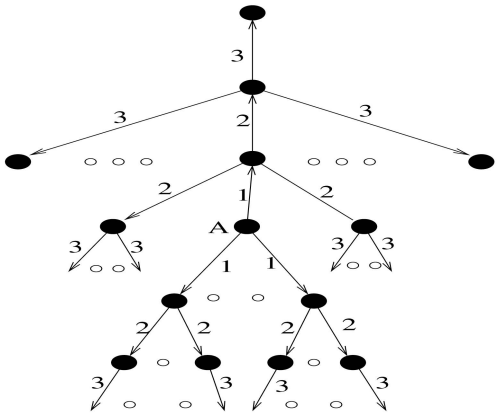


Fig. 4. The scenario of the first three-hop flooding from node A in the *FloodNet* tree. The numbers 1, 2, and 3 denote the three-hop flooding message sequence.

Applying the same rule, on the mth hop, we will obtain the following number of new nodes:

$$\begin{aligned} &Deg - 1 + (Deg - 2) \cdot q \cdot (Deg - 1) + \ldots \\ &+ (Deg - 2)(q \cdot (Deg - 1))^{m-2} + (q \cdot (Deg - 1))^{m} \\ &\quad = Deg + (m - 2) \cdot (Deg - 2). \end{aligned} \tag{6}$$

Combining (3), (4), (5), and (6), after m-hop flooding in the *FloodNet* tree, the number of new nodes discovered can be generalized as

$$2 + (m - 1) \cdot Deg + \frac{(m - 1)(m - 2)}{2} \cdot (Deg - 2). \tag{7}$$

Recall that Deg can be written as $\frac{2-p}{1-p}$. Since the value of p (the probability of a node being a leaf) does not fluctuate much and our simulation shows that the value of p is normally between 0.7 and 0.8, the average degree of a node Deg is greater than 2 and is almost a constant. Therefore, the growth rate of the number of new nodes is asymptotically equal to $\frac{Deg-2}{2} \cdot m^2$. The solid curve in Fig. 5 illustrates the growing trend of discovered nodes in topology T1.

Let us consider the case when $p = 0.7$, as shown in Fig. 5. The numbers of new nodes found from the second hop to the sixth hop are 7.3, 14.2, 23.3, 35.6, and 51.7, respectively. The combined growth rate after six hops of flooding is about 52. Let us consider the flooding arrangement of (4, 6) for topology T1. After four hops of pure flooding in T1, our simulation shows that the average number of seeds is around 800. If each of those seeds gains the growth of 52 after six hops of flooding in *FloodNet*, totally, 41,600 peers will be discovered at the end of (4, 6) flooding, which is about 97 percent of the coverage. On the other hand, simulation shows that the (4, 6) flooding coverage is 90 percent, which is very close to our analysis result. The 7 percent difference can be explained as follows: In the latter part of the second stage, the flooded messages could collide with each other and be dropped, because the flooding messages are saturated in *FloodNet*.

5.2 Avalanche Effect

Although the seed sprouting model well explains the coverage growth trend for the flooding arrangement of (4, 6) on topology T1 in Fig. 5, it barely matches the growth

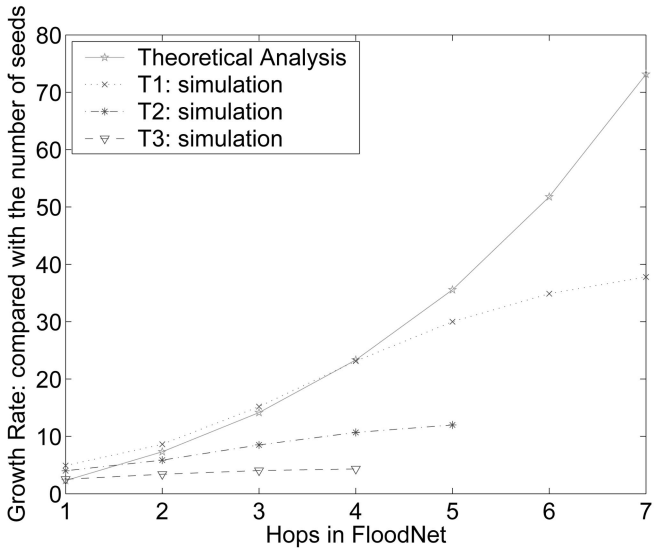


Fig. 5. Comparison of growth rates between the analysis results for seed sprouting and the simulation results of the $(4, *)$ arrangement on T1, T2, and T3. The rates are computed by comparing against a number of seeds. The solid curve reveals the growth of the seed sprouting model when $p = 0.7$. The simulation results on T1 match the seed sprouting analysis very well in the first four hops. The growth rates on the fifth and sixth hops are significantly lower in the simulation, because *FloodNet* is almost saturated in the last several hops. The simulations on topologies T2 and T3 show that the actual growth rate is much lower than the analytical result. There are two reasons for this: 1) The T2 and T3 networks are much smaller than T1, so the chance of message collision is higher for the flooding arrangement $(4, *)$, and the networks get saturated very quickly, and 2) The T2 and T3 networks have higher connectivity, so after the initial four-hop flooding, there are 2,223 seeds and 4,939 seeds generated, respectively. The large number of seeds also leads to frequent message collision and early saturation.

patterns for the arrangements of $(1, *)$, $(2, *)$, and $(3, *)$ on all the three topologies. In Fig. 6, we illustrate the coverage growth with the arrangements of $(1, *)$, $(2, *)$, and $(3, *)$ on T1, T2, and T3. For example, the simulation on topology T1 shows that $(1, *)$ flooding can obtain the growth rate of over 8,000, which is 10 times more than that of the seed sprouting analysis. Obviously, the seed sprouting model cannot explain the coverage growth pattern observed in those arrangements. Carefully examining Fig. 6, we find that the seed sprouting model can only be used to predict the coverage growth for the first several hops. Therefore, the coverage growth pattern has changed after several initial hops of flooding in *FloodNet*. Note that this phenomenon only happens when a flooding arrangement has a very limited scope of its pure flooding (such as $(1, *)$, $(2, *)$, and $(3, *)$).

Recall that *FloodNet* is made of several diamond-shaped tree-like networks. After the first several hops of flooding in *FloodNet*, the query messages reach the tree root. Starting from the next hop, the root node starts flooding the message down to the entire tree. This flooding model is very different from the seed sprouting model that we discussed in the previous section. We call it the *avalanche effect*. The avalanche effect happens when the number of seeds is relatively small, especially for flooding arrangements $(1, *)$ and $(2, *)$. In that case, the seed sprouting procedure will quickly end as soon as the *FloodNet* tree root is reached by

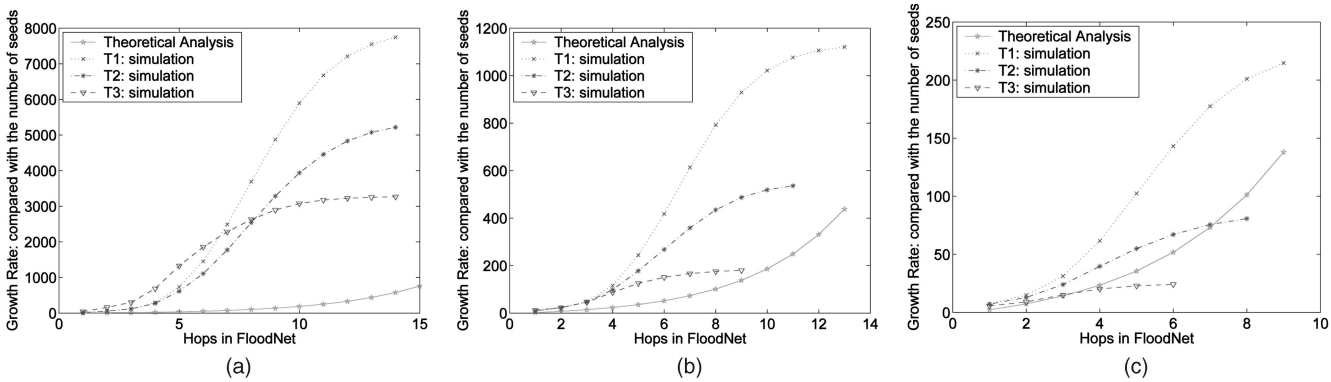


Fig. 6. For flooding arrangements $(1,*)$, $(2,*)$, and $(3,*)$, the actual coverage growth in *FloodNet* is much larger than that from the seed sprouting analytical model. The simulation shows that topology T1 gains almost 8,000 times of coverage growth under the arrangement of $(1,*)$, whereas the analysis only shows that the coverage growth is only about 800 times, which is 1/10 of the simulation result. Therefore, the seed sprouting analytical model is not suitable when the number of *seeds* is very small. At this time, the avalanche effect can well explain what happens in *FloodNet*. The avalanche effect is dominant in the $(1,*)$ and $(2,*)$ flooding for all three topologies. For the $(3,*)$ arrangement, the avalanche effect is apparent only for topology T1. The growth curves for T2 and T3 actually are very close to the seed sprouting analysis result. The reason is that the three-hop pure flooding already generates enough number of seeds in topologies T2 and T3; thus, the avalanche effect is suppressed during flooding. (a) Growth rate comparison: the $(1,*)$ policy. (b) Growth rate comparison: the $(2,*)$ policy. (c) Growth rate comparison: the $(3,*)$ policy.

flooding messages. Thereafter, the avalanche procedure will dominate the coverage growth.

Again, Fig. 6 shows the growth rate for the arrangement of the $(1,*)$ flooding on topology T1. The coverage keeps a consistent growth rate from hop 5 to hop 10, and then, the growth rate starts decreasing after hop 11. This is because the number of new tree nodes reached in each hop starts decreasing after the messages are flooded six hops from the roots. This observation supports our assumption of diamond-shaped trees.

5.3 Comparisons between Seed Sprouting and the Avalanche Effect

We have found that seed sprouting is not the only coverage growth phenomenon in *FloodNet*. The avalanche effect dominates the coverage growth when the number of seeds is small. Both seed sprouting and the avalanche effect can achieve the desired coverage. We are interested in their relative advantages and disadvantages in practice.

With regard to the response time, seed sprouting takes less time than that of the avalanche effect to reach a desired coverage. Although the avalanche effect can achieve a higher growth rate than that of seed sprouting, it takes about five to six hops (according to our tests on existing traces), on the average, for a flooding message to reach the tree roots. Therefore, the avalanche spends more time overall. With regard to reliability, we claim that seed sprouting is much more reliable than the avalanche effect. Recall that the avalanche effect relies on the procedure of finding the tree roots. A problem at either the tree roots or the path to the tree roots will cause avalanche to fail. However, since there are relatively more seeds in seed sprouting, a limited number of faulty seeds will not affect the overall sprouting performance. Finally, the advantage of the avalanche effect is that it almost does not have redundant messages. Recall that the typical avalanche effect happens when we use the $(1,*)$ arrangement for flooding, and in this case, we will not have any redundant messages. Although some redundant messages can be generated in the seed-sprouting scheme, we have shown that several initial hops of pure flooding is highly efficient. Thus, message redundancy is not a problem for seed sprouting. Therefore, we conclude that the seed sprouting method is more preferred over the avalanche effect in practice.

As discussed in the previous section, different flooding arrangements will lead to two different coverage growth patterns, and we want to take advantage of the seed sprouting coverage growing pattern to quickly obtain a satisfying coverage. One may have a question on how the *LightFlood* parameters can be chosen to achieve an appropriate searching coverage and short response time. Based on the observations of our tests on more than 20 Gnutella traces, *LightFlood* with arrangement $(4,*)$ achieves superior performance in all traces (the three examples are shown in Fig. 7). Considering the fact that 95 percent of the peers in a Gnutella system could be reached within seven hops $(\mathrm{TTL}=7)$ by flooding [11], we are confident that arrangement $(4,*)$ is close to the optimal *LightFlood* parameter for Gnutella networks. In the next section, we will have a more detailed discussion of the effect of arrangement parameters on latency and coverage.

6 Performance Evaluation

The overhead of flooding can be quantified by the number of redundant messages generated. Broadcasting a message causes processing and traffic overhead. Redundant message transmission and the associated message processing at both ends of a link are sheer overhead. In the ideal case, where all such redundant messages are eliminated, it takes only N messages to reach N peers from a message initiator.

The goal of our evaluation is to study the performance of *LightFlood* compared with pure flooding. Because the general *LightFlood* scheme (M,N) represents a class of arrangements, we will also investigate the implication of timing to start *FloodNet* on performance. There are several questions that we are particularly interested in:

1. With a given TTL, how does the increase in M affect the performance of *LightFlood*?

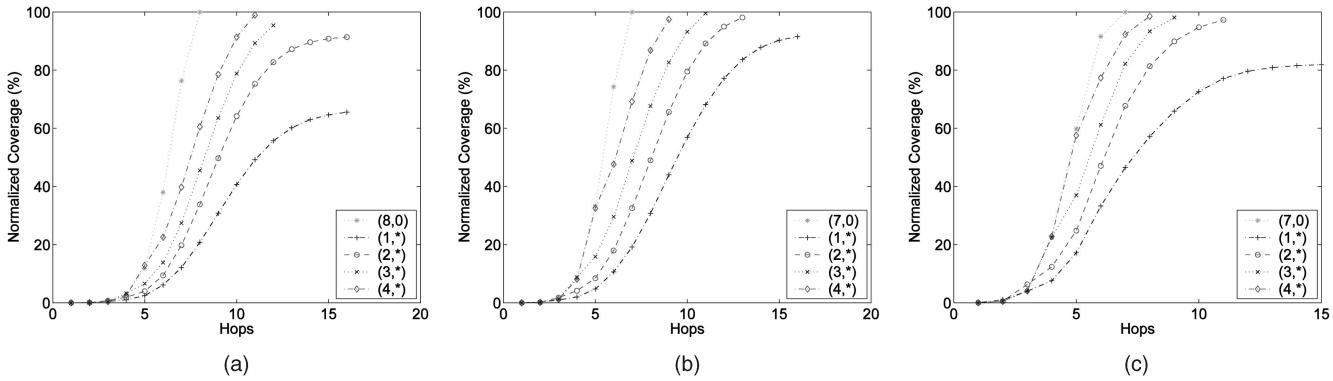


Fig. 7. Simulations on three different topologies: T1, T2, and T3. For each experiment, every peer, in turn, starts a searching procedure and broadcasts a query message to the network by using *LightFlood*. All statistics shown are average values. We normalize the coverage to that of eight-hop pure flooding for T1 and seven-hop pure flooding for T2 and T3. (a) Topology T1. (b) Topology T2. (c) Topology T3.

2. How does the connectivity of a topology affect the performance of pure flooding and *LightFlood*?
3. With a coverage similar to that of pure flooding, how much reduction of redundant messages in *LightFlood* is there, and how many additional hops does *LightFlood* have to travel to achieve a similar coverage?
4. How does the departure or failure of peers affect the performance of pure flooding and *LightFlood*?
5. How well do the existing flooding improvement techniques such as expanding ring benefit from *LightFlood* when it is built into these schemes?

For each search method, we run a series of simulations to collect the statistics in which we are interested. We use the three topology graphs listed in Table 1. For each experiment, we broadcast a message from each peer in a topology. Statistics are collected from the broadcasts, and we report their average values.

6.1 The Effects of Different Arrangements (M, N) on the Latency for a Given Coverage

For the *LightFlood* scheme, we not only need to understand how the parameters M and N of the arrangement of (M, N) would affect flooding patterns but are also interested in learning how the connectivity of a topology affects the performance of *LightFlood*. We run a series of simulations on the three different topologies T1, T2, and T3.

We normalize the coverage (the number of peers reached) to that of eight-hop pure flooding for T1 and seven-hop for T2 and T3, which are presented as arrangements (8, 0) and (7, 0). Fig. 7 lists the coverage growths with the increase in hops for different arrangements in the three topologies.

Based on the figures, we can make following observations. First, among different arrangements $(M, *)$, not every M value can reach the coverage that (8, 0) or (7, 0) obtains. For example, $(1, *)$ on T3 can only achieve 81 percent of coverage, whereas $(1, *)$ on T2 only has about 90 percent of coverage. The reason is that *FloodNet* is made of multiple tree-like components. If the number of seeds is not large enough, seeds cannot be dispersed into all trees during the first stage. Second, different values of M require different TTLs to obtain a comparable coverage to that of (8, 0) or (7, 0). For topology T1, it takes $(2, *)$ 13 hops to reach 80 percent of the coverage by (8, 0), whereas it takes arrangement $(4, *)$ only eight hops to reach a similar coverage. Recall that the coverage growth for $(4, *)$ is contributed by seed sprouting, and that of $(2, *)$ is mainly made by the avalanche effect. We may conclude that the avalanche effect has a longer delay than seed sprouting to achieve a similar coverage. Third, as shown in Fig. 7, for T1, it only takes $(4, *)$ nine hops to reach the same coverage as the one that costs the pure flooding seven hops. Simulation shows that the redundant messages generated during four-hop floodings in topologies T1, T2, and T3 are only 1.9 percent, 2.9 percent, and, 10.5 percent, respectively. This means that *LightFlood* on topology T1 is able to eliminate over 90 percent of redundant messages by only introducing a two-hop delay.

6.2 How Does M of *LightFlood* (M, N) Affect Its Efficiency?

Fig. 8 shows the message efficiency at different stages of flooding with various arrangements. We define message efficiency for a specific hop h as the ratio of the number of peers reached and the number of the messages forwarded in the initial h hops. The message efficiency for the last hop is the efficiency of the arrangement. The optimal efficiency is 1 if there are no redundant messages. In the experiments, we use the coverage of (7, 0) as the baseline to set the stopping hops for all *LightFlood* arrangements in the three topologies. In the figure, we can see that although (7, 0) uses the least number of hops to achieve its coverage, its message efficiency deteriorates sharply with the increase in the number of hops. This is consistent with what we have observed in Section 2, where more redundant messages are generated with a smaller coverage growth rate within high hops than those within low hops. However, once the flooding is switched from pure flooding to flooding over *FloodNet*, the efficiency curves immediately stop dropping and start rising. This results in a large gap of message efficiency for the final hop between (7, 0) and $(M, *)$ for $(M \leq 5)$. The smaller the value of M is, the larger the efficiency improvement becomes. For example, for $M = 1$, 2, 3, the final efficiencies are close to 100 percent. At the same time, the increase in TTL time for a similar coverage is modest for $M \geq 3$. Generally, the arrangements of $(3, *)$,

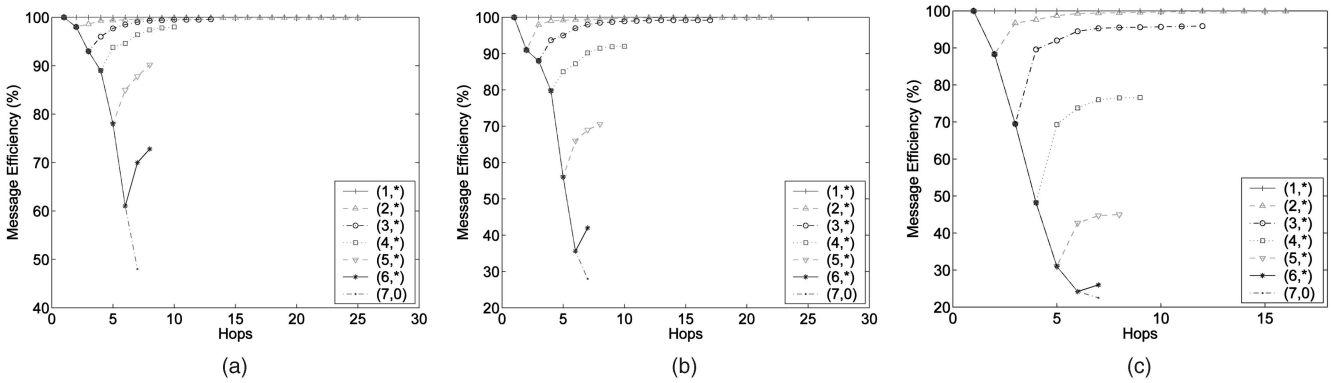


Fig. 8. Message efficiency of various *LightFlood* arrangements in the three Gnutella topologies. Message efficiency is the ratio between the number of peers reached and the number of forwarded messages. For example, with arrangement $(4,*)$ in topology T1, on the average, the broadcast messages reached 1,067 peers with 1,194 forwarded messages within the initial four hops; thus, the efficiency shown in the figure for $(4,*)$ for the fourth hop is $1{,}067/1{,}194 \times 100 \text{ percent} = 89 \text{ percent}$. (a) Topology T1. (b) Topology T2. (c) Topology T3.

$(4,*)$, and $(5,*)$ strike a good balance between efficiency and latency. For example, the efficiency of pure flooding (7, 0) is improved from 48.2 percent to 97.2 percent by arrangement (4, 6) in topology T1, whereas (4, 6) only increases TTL by 3. Another observation is that the efficiency becomes worse with the increase in the average node degree of the topologies. For example, the efficiency for (7, 0) on T1 with the average degree of 3.40 is 48.2 percent, whereas the efficiency on T2 with the average degree of 4.72 and T3 with the average degree of 5.43 are 28.1 percent and 22.6 percent, respectively. The efficiencies are improved to 97.2 percent, 91.3 percent, and 75.8 percent in T1 and T2, and T3, respectively. This is because higher connectivities cause more redundant connections among links, thus generating more redundant messages. This makes *LightFlood* more attractive for systems with high-topology connectivities.

In summary, for overlay topologies with different connectivities, arrangements of $(3,*)$, $(4,*)$, and $(5,*)$ provide 1.8-4.3 times of message efficiency improvement, which means that 44.5 percent to 76.7 percent of flooding cost reduction, whereas they require a modest increase in TTL time from 1 to 10 for a similar coverage as that of pure flooding. Comparatively, $(3,*)$ improves the flooding efficiency at the cost of reduced coverage, whereas $(5,*)$ improves the flooding coverage at the cost of lowered efficiency. A lower connectivity of topologies leads to a higher flooding efficiency, whereas a higher connectivity of topologies leads to a larger flooding coverage. Considering various topology connectivities, $(3,*)$ can be used in topologies with high connectivity like T3, whereas $(5,*)$ can be used in topologies with low connectivity like T1. There is a spectrum of the arrangement of $(\mathrm{M},*)$ with pure flooding $(7,*)$ and pure tree broadcast $(1,*)$ on the two extreme sides. According to the observation made in [11], the changes in the average connectivity are small over a long period of time. An arrangement of $(\mathrm{M},*)$ with a reasonably chosen M value (3, 4, or 5) apparently performs better than pure flooding in terms of search efficiency and strikes a good balance between systemwide traffic consumption and user-perceived latency. Because *LightFlood* is intended to be a substitute for pure flooding and its search coverage is compared with the corresponding pure flooding coverage, the actual P2P network size does not affect the selection of M. We tested $(4,*)$ on all 48 Gnutella topologies in [3], with their average connectivity degrees ranging from 2.37 to 6.73, and we found that it consistently performs well in terms of efficiency and coverage compared with (7, 0). Thus, $(4,*)$ is a good representative of the family of arrangements, regardless of system connectivity and network size in terms of traffic and latency. In the remainder of this paper, we refer to $(4,*)$ when we mention *LightFlood* if not stated explicitly.

6.3 Message Coverage from Individual Peers

We have reported the statistics after averaging them over all peers. The average efficiency characterizes the cost on the overlay, and the average coverage reflects the scope that an average peer can reach, which implies its quality of service, that is, the number of results returned. However, the specific coverage size of an individual peer could be sacrificed, even though the average size is satisfactory. One might be concerned that the use of *FloodNet* could shrink the flooding coverage for certain peers, whereas the harm to these peers are not reflected in the average statistics. If this is the case, this shrinkage could discourage these users from staying in the systems.

To investigate the issue, we compared the distributions of the coverage of all peers in the topologies between *LightFlood* (4, 6) and pure flooding (7, 0). Fig. 9 gives their Cumulative Distribution Function (CDF) curves of coverage distributions, which shows the percentage of peers from which the flooding coverage is below a certain coverage in the percentage of the total number of peers. In the figure, we can see the impact of the connectivity of topologies on peer coverage distributions. The larger the average degree, the fewer the peers with small coverage. For example, for pure flooding (7, 0), there are 15 percent of peers whose coverage is less than 50 percent of the total peers in topology T1, whereas there are almost no such peers in T2 and T3 because of their relatively high connectivity degrees. The coverage of *LightFlood* (4, 6) on T1 is much better than that of pure flooding (7, 0) on T1 due to the three additional flooding hops. For topologies T2 and T3, the difference of coverage distributions between *LightFlood* and pure flooding is small. In the figures, we see that *LightFlood* not only

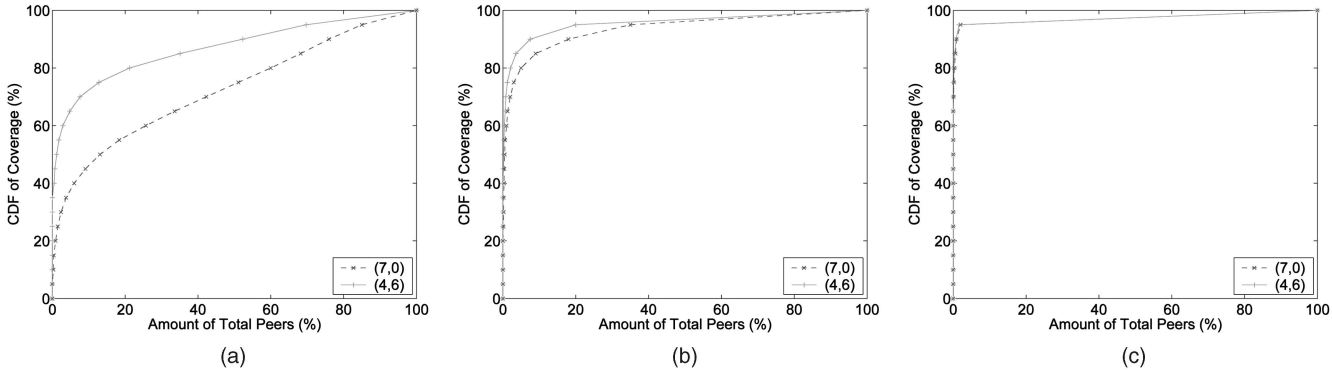


Fig. 9. CDF curves of coverage distributions in *LightFlood* and pure-flooding arrangements, which show the percentage of peers from which the flooding coverage is below a certain value in the percentage of the total peers. For example, with arrangement (4, 6) in topology T1, there are only 20 percent of total peers whose (4, 6) coverage is less than 80 percent of all peers. (a) Topology T1. (b) Topology T2. (c) Topology T3.

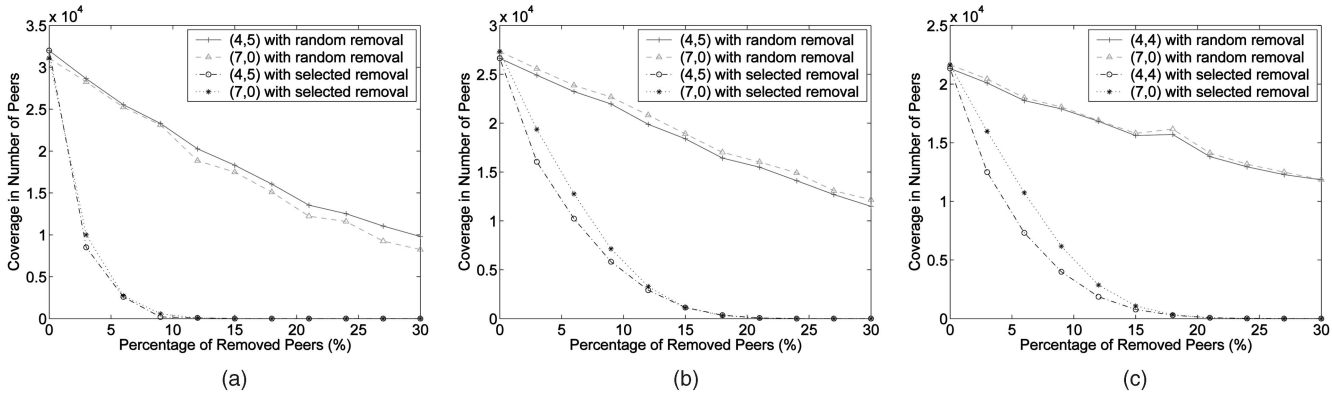


Fig. 10. Changes in coverage size with the percentage of removed peers in the three topologies for the selected *LightFlood* and pure-flooding schemes. There are two options for selecting the removed peers: 1) randomly chosen peers (random removal, as shown in the legend) or 2) the best connected peers in the topologies (selected removal, as shown in the legend). (a) Topology T1. (b) Topology T2. (c) Topology T3.

keeps the coverage of pure flooding with small additional TTL time collectively but also performs as well as pure flooding for individual peers.

6.4 Impact of the Departure of Peers on Performance Degradation

When there are a considerable number of peers simultaneously leaving the system or failing, which is possibly due to malicious attacks, the coverage of a flooding message may be reduced. This is because each peer also serves as a router for forwarding messages. In *LightFlood*, when a peer leaves, its child peers would select another available neighbor with the highest degree as its father to take place of the leaving peer. It has been shown that Gnutella is highly resilient in the face of a random breakdown but is highly vulnerable in the face of the removal of the best connected peers, which could happen in well-orchestrated targeted attacks [13].

To compare the impacts of the situation on *LightFlood* and pure flooding, we selected (4, 5) for topologies T1 and T2 and (4, 4) for topology T2 to show their coverage changes when randomly chosen peers are removed and when the best connected peers are removed. The experimental results are shown in Fig. 10. Our results confirmed that coverage has a graceful degradation with random peer removal, but the network could become mostly unconnected with the removal of a small percentage of the best connected peers. In Fig. 10a, the coverage of (4, 5) with random peer removal is even larger than that of (7, 0) with random peer removal, because in the case without peer removal in topology T1, the coverage of (4, 5) is already much higher than that of (7, 0), as shown in Fig. 7a. Our *LightFlood* cannot improve the worst case with the removal of high-degree peers, because *FloodNet* does not build additional links among peers beyond the originally existing links. However, *LightFlood* does behave almost the same as pure flooding with graceful degradation in the face of random removal.

6.5 How Much Does Expanding Ring Benefit from *LightFlood*?

The expanding-ring scheme has been shown to be an effective approach to achieve a very small stopping TTL[1] and to eliminate most of the redundant messages in flooding when widely duplicated files are searched [7]. For example, it takes only one or two hops to find one result when the files are duplicated at over 10 percent of the peers. In such a case, there is no difference when expanding ring uses either pure flooding or *LightFlood*, because *LightFlood* also uses pure flooding for its initial hops.

The concern with expanding ring comes with its search for less popular files. Lv et al. [7] have shown that the stopping TTL could be much enlarged and the number of messages used could be significantly increased. Our

1. Stopping TTL is the TTL used in the last flooding of its multiple consecutive flooding in the expanding-ring scheme.

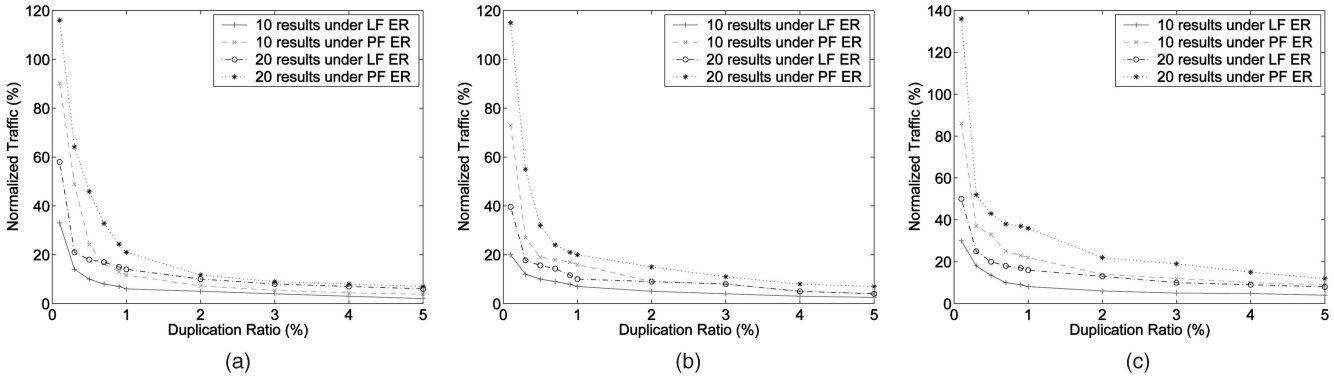


Fig. 11. Traffic of *LightFlood* expanding ring (LF ER in the legend) and pure flooding expanding ring (PF ER in the legend) for queries requesting at least 10 or 20 results on objects with different duplication ratios in the system. The *LightFlood* policy is (4, 6), whereas the pure-flooding policy is (7, 0). Traffic is normalized by the number of messages generated in a (7, 0) pure-flooding operation. (a) Topology T1. (b) Topology T2. (c) Topology T3.

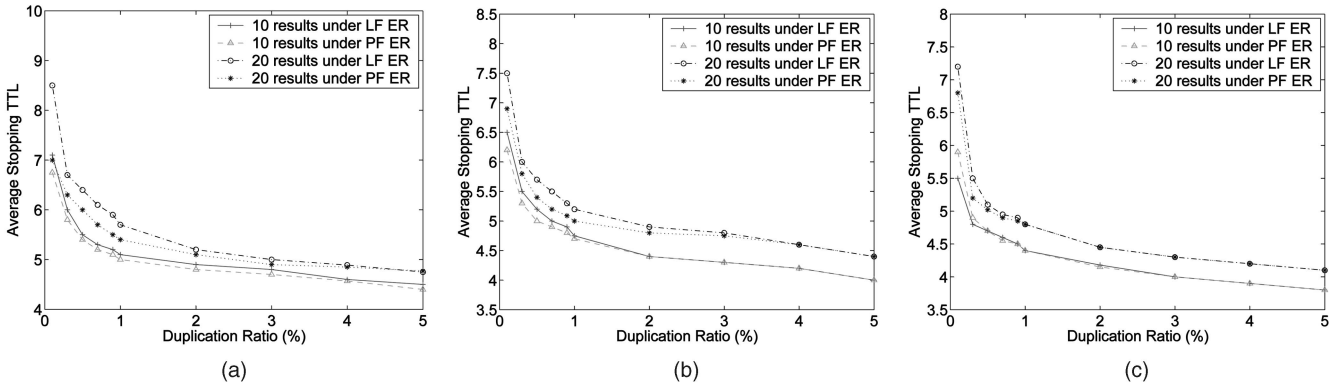


Fig. 12. Stopping TTLs in *LightFlood* expanding ring (LF ER in the legend) and pure flooding expanding ring (PF ER in the legend) for queries requesting at least 10 or 20 results on objects with different duplication ratios in the system. The *LightFlood* policy is (4, 6), whereas the pure-flooding policy is (7, 0). (a) Topology T1. (b) Topology T2. (c) Topology T3.

measurement-based study on Gnutella in [4] and [5] has shown that more than 20 percent of queries can only find less than 10 results in a seven-hop flooding with more than 50,000 peers in the system. For these queries for unpopular files, the expanding ring has to considerably increase its stopping TTLs, which could adversely deteriorate the broadcasting efficiency and thus inflict a heavier burden on the systems than pure flooding does.

To investigate the benefits that expanding ring could obtain by using *LightFlood*, instead of pure flooding, in its repeated broadcastings, we ran the simulations to compute the average number of forwarded messages used and stopping TTLs when the number of satisfactory results are 10 and 20, respectively. We set the starting TTL to 1 and increase the TTL by 2 each time, as suggested in [7]. We use a range of duplication ratios, the one between the number of peers with the requested objects and all the peers in a topology, from 0.1 percent to 5 percent. The *LightFlood* arrangement used is (4, 6), whereas the pure-flooding arrangement used is (7, 0). We randomly select 10 percent of nodes in each overlay topology as the initiator of queries and compute the average number of generated messages and the average stopping TTLs of these queries. Fig. 11 shows the traffic (messages produced by forwarding) generated in both arrangements with various duplication ratios. We see that the traffic of pure flooding expanding ring is extremely heavy, even exceeding the traffic of pure flooding (7, 0), when duplication ratios are low in all three topologies, especially when 20 results are required, although the traffic is reduced sharply with the increase in duplication ratios. Considering the prolonged TTL time caused by repeated broadcastings, practitioners would be discouraged from implementing expanding ring due to the worst scenarios. However, the expanding-ring scheme can significantly reduce its traffic to well below (7, 0) traffic in all cases when it is built on *LightFlood*, especially with low duplication ratios, as shown in the solid and dashed-dotted lines in Fig. 11. At the same time, the increase in TTL time is small compared with the time spent on iterative floodings in the pure flooding expanding ring, as shown in Fig. 12. For example, the average number of forwarded messages is reduced from 117 percent to 40 percent of (7, 0) traffic by the arrangement of (4, 6), whereas the average stopping TTL is only increased from 6.9 to 7.5 on topology T2 when the number of satisfiable results is 20 and the duplication ratio is 0.1 percent. In summary, although expanding ring and iterative flooding are promising searching techniques to replace pure flooding, they only become practical when they are built on a low-overhead flooding technique such as *LightFlood*.

7 Conclusions

Although flooding is essential in an ad hoc self-organized P2P file-sharing system, its overhead imposed on the

underlying infrastructure is large and threatens the scalability of the distributed systems. Our *LightFlood*, represented by its $(4, *)$ arrangement, provides a simple scheme to conduct flooding in a cost-effective way upon an existing overlay. It effectively combines the advantage of quick coverage increase within low hops from pure flooding and the advantage of low broadcasting overhead from the *FloodNet* suboverlay to significantly reduce the number of redundant messages generated by pure flooding and to keep the merits of pure flooding such as low latency and high reliability. The construction and maintenance of *FloodNet*, which is used in *LightFlood*, are actually a process of the selection of existing links, which rely on only local knowledge, and are of low cost. Simulation experiments show that *LightFlood* can reduce up to 69 percent of flooding messages and associated computing cost, with only a small TTL time increase compared with pure flooding. *LightFlood* can also greatly boost the performance of existing schemes aiming at the reduction of flooding cost.

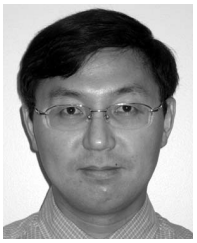
ACKNOWLEDGMENTS

The authors appreciate the constructive and critical comments made by the anonymous referees and thank Bill Bynum for his suggestions. This work is partially supported by the US National Science Foundation under Grants CNS0098055 and CNS0509054.

REFERENCES

[1] http://www.kazaa.com, 2007.
[2] http://www.limewire.com, 2007.
[3] *Clip2 Distributed Search Solutions,* http://www.clip2.com, 2007.
[4] L. Guo, S. Jiang, L. Xiao, and X. Zhang, "Exploiting Content Localities for Efficient Search in P2P Systems," *Proc. 18th Ann. Conf. Distributed Computing (DISC '04),* Oct. 2004.
[5] L. Guo, S. Jiang, L. Xiao, and X. Zhang, "Fast and Low Cost Search Schemes by Exploiting Localities in P2P Networks," *J. Parallel and Distributed Computing,* vol. 65, no. 6, pp. 729-742, 2005.
[6] S. Jiang, L. Guo, and X. Zhang, "*LightFlood*: An Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems," *Proc. 32nd Int'l Conf. Parallel Processing (ICPP '03),* Oct. 2003.
[7] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. 16th Ann. ACM Int'l Conf. Supercomputing (ICS '02),* June 2002.
[8] Q. Lv, S. Ratnasamy, and S. Shenker, "Can Heterogeneity Make Gnutella Scalable?" *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* Mar. 2002.
[9] M.E.J. Newman, S.H. Strogatz, and D.J. Watts, "Random Graphs with Arbitrary Degree Distributions and Their Applications," *Physical Rev. E,* vol. 64, p. 26118, 2001.
[10] M.K. Ramanathan, V. Kalogeraki, and J. Pruyne, "Finding Good Peers in Peer-to-Peer Networks," *Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS '02),* Apr. 2002.
[11] M. Ripeanu and I. Foster, "Mapping Gnutella Network," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* Mar. 2002.
[12] J. Ritter, *Why Gnutella Can't Scale. No, Really,* http://www.monkey.org/~dugsong/mirror/gnutella.htm, Feb. 2001.
[13] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. SPIE/ACM Conf. Multimedia Computing and Networking (MMCN '02),* Jan. 2002.
[14] K. Scipanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proc. IEEE INFOCOM '03,* Mar. 2003.
[15] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network," *Proc. 19th IEEE Int'l Conf. Data Eng. (ICDE '03),* Mar. 2003.
[16] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Systems," *Proc. 22nd IEEE Intl' Conf. Distributed Computing Systems (ICDCS '02),* July 2002.

Song Jiang received the BS and MS degrees in computer science from the University of Science and Technology of China in 1993 and 1996, respectively, and the PhD degree in computer science from the College of William and Mary in 2004. He is currently an assistant professor in the Department of Electrical and Computer Engineering, Wayne State University. From 2004 to 2006, he was a postdoctoral research associate at Los Alamos National Laboratory, where he developed next-generation operating system capabilities for large-scale parallel computing systems. His research interests include operating systems, distributed systems, and high-performance computing. He is a member of the IEEE.



Lei Guo received the BS degree in space physics and the MS degree in computer science from the University of Science and Technology of China in 1996 and 2002, respectively. He received the PhD degree in computer science and engineering from the Ohio State University in 2007. He is currently a senior member of technical staff at Yahoo! Inc. His research interests include distributed systems, P2P systems, multimedia systems, wireless networks, and Internet measurement and modeling. He is a member of the IEEE.

Xiaodong Zhang is the Robert M. Critchfield Professor of Engineering and the chair of the Department of Computer Science and Engineering, Ohio State University. From 2001 to 2004, he was the program director of the Advanced Computational Research, National Science Foundation. He is the associate editor-in-chief of the *IEEE Transactions on Parallel and Distributed Systems* and is serving on the editorial boards of the *IEEE Transactions on Computers*, *IEEE Micro*, and the *Journal of Parallel and Distributed Computing*. He is a senior member of the IEEE.

Haodong Wang is currently working toward the PhD degree in the Department of Computer Science, College of William and Mary. His research interests include pervasive computing systems, security and privacy for wireless low-power devices in wireless ad hoc, sensor, and peer-to-peer networks.