High Coverage Search in Multi-Tree Based P2P Overlay Network

Cuihua Zuo[1], Ruixuan Li[1+], Haiying Shen[2], Zhengding Lu[1]

[1]College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P.R.China

[2]Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA

E-mail: zuocuihua@smail.hust.edu.cn, rxli@hust.edu.cn, hshen@uark.edu, zdlu@hust.edu.cn

Abstract

The blind flooding algorithm under a time-to-live (TTL) constraint used in unstructured peer-to-peer (P2P) networks, such as Gnutella, may cause a large amount of network traffic. The algorithm cannot guarantee for acquiring the required data objects, especially for rare ones. To mitigate these problems, this paper proposes PercolationNET, a multi-tree sub-overlay, which is built on top of an existing P2P overlay (named original overlay). PercolationNET organizes peers in a tree-based structure which facilitates reliable and efficient message dissemination for search. The search process is divided into two stages. A query message is first propagated on the original overlay, and then broadcast along the sub-overlay PercolationNET. PercolationNET combines the advantages of fast coverage speed in flooding-based scheme and low traffic cost in tree-based scheme. The experimental results of PercolationNET compared with FloodNet confirm the superiority of PercolationNET in achieving faster coverage speed and lower message cost.

1. Introduction

P2P networks have become a dominant part of the Internet traffic due to the tremendous success of P2P file-sharing systems such as Gnutella [1] and KaZaA [2]. P2P overlay networks can be classified into two categories: structured and unstructured. Structured overlays [3, 4] tag the peers with peer identifiers. The placement of shared data and topology characteristics of the networks are tightly controlled based on distributed hash table (DHT). In contrast to structured overlays, unstructured overlays do not follow any specific topology characteristics. Therefore, they don't apply any clue as to where queried content is located. In spite of the absence of location clue, unstructured P2P networks have several desirable properties not easily achieved by structured counterparts — they are highly resilient to node failures and incur low overhead for peer arrivals and departures. In addition, they are simple to implement and nearly incur no overhead in topology maintenance [5]. The predominating search mechanism in unstructured networks is message flooding with a fixed TTL restriction. This simple method does not provide guarantee that an object existing in the network can be found. Moreover, flooding does not scale well in terms of message overhead.

Realizing the importance of flooding in unstructured P2P networks and its problems, our work focuses on overlay construction for search with data retrieval guarantees as well as low traffic cost. We build PercolationNET, a multi-tree sub-overlay, upon the original overlay. Correspondingly, the search process is divided into two stages. In the first stage, a query message is flooded with an appropriate TTL value in the original overlay network so that the message can spread to all trees of the sub-overlay with smaller redundant messages. Then in the second stage, the query message is broadcasted along the sub-overlay, which has low connectivity but can ensure that any object existing in the network can be found. The experimental results show that PercolationNET offers high probabilistic guarantees of the accessibility of data objects, while incurring minimal overhead.

The rest of the paper is organized as follows. Section 2 describes a survey of related work. Section 3 details the design of PercolationNET in terms of overlay construction and maintenance. Section 4 evaluates the performance of the PercolationNET in comparison with FloodNet [18] through simulation experiments. Section 5 presents the conclusion and future work.

2. Related work

Many efforts have been devoted to avoid the large volume of unnecessary traffic incurred by the flooding-based search in unstructured P2P networks. In general, they can be categorized into three types: modified flooding, caching index or content, and overlay optimization.

Unlike pure flooding, which starts with a fixed TTL and sends query messages to all neighbors, modified flooding takes more dynamic factors into consideration to reduce traffic overhead. For example, in Directed BFS [6], each peer maintains statistic information based on a number of metrics such as degree information of neighbors. A peer selects a subset of its neighbors, such as the neighbors that have large degrees, to send its query. In the expanding ring [7], flooding is initiated with increasing TTLs. A peer starts a flooding with a small TTL, and waits to see if the search is successful. If it is, the node stops. Otherwise, the node increases the TTL and starts another flooding operation. The process repeats until the queried object is found. Adaptive Flooding [8] combines the above two schemes. It not only relays a query message to limited neighbors, but also adjusts TTL value. Although these schemes can save traffic overhead to some extent and reduce the latency of popular data objects, their performance could be uncertain for rare or distant ones when the search scope is deepening. In contrast, our approach can reduce the network traffic with high coverage speed.

The second approach is caching index or content. In Local Indices [6] policy, each peer maintains an index of files available in the nodes within given radius r. When a peer receives a query, it can process the query on behalf of all nodes within the radius r. Caching file contents (replication) [9, 10] has also been studied. The work in [11] evaluates and compares different replication strategies. The article [12] researches on how many replications should be made and where to locate these replications. In Uniform Index Caching (UIC) [13], each peer stores IP addresses of the peers that have the contents whose queries passed the peer. If the same objects are queried again, the peer stops the flooding and replies with the location stored in its memory. In this paper, we mainly consider search scope rather than searching a concrete object. If we integrate the caching strategy into our approach, the performance in terms of message overhead and response time for object search can be improved further.

The third approach is overlay optimization [14, 15] that is closely related to what this paper presents. Mismatch between logic overlay and physic overlay is a well-known problem in P2P networks. Recent efforts including Location-aware Topology Matching (LTM) [16] and Scalable Bipartite Overlay (SBO) [17] have been made to address the mismatch problem without sacrificing the search scope. In LTM, each peer issues a detector so that the peers receiving the detection can record relative delay information as the optimization basis. SBO scheme optimizes the overlay topology by identifying and replacing the mismatched connections, and it distributes optimization tasks in peers with different colors. These two schemes mainly improve one aspect of the performance — response time. In [18], a sub-overlay FloodNet is constructed for the purpose of reducing the number of redundant messages. FloodNet consists of all peers in original overlay and the links between each peer and its parent who is its neighbor with the maximum secondary degree (i.e., the sum of the degrees of a peer's neighbors). Though it can reduce the number of redundant messages effectively, it needs more hops to reach all peers of the network. Additionally, the secondary degree of each peer is volatile due to the dynamic characteristic of P2P networks.

Different from the aforementioned approaches, the proposed sub-overlay in this paper is constructed depending on the overall characteristic of the original overlay. Since the search can span the entire network along the sub-overlay with an appropriate TTL value, we can regard this sub-overlay network with percolation characteristic. Hence, we call the sub-overlay as PercolationNET. In this paper, we use flooding as an example of searching in the first stage of our approach. However, other search schemes can be also used as long as they can spread message to all trees of PercolationNET with a limited TTL.

3. PercolationNET design

FloodNet [18] is built based on the number of each peer's secondary neighbors (i.e., the neighbors of a peer's neighbors). When flooding runs over FloodNet, it can eliminate a large number of redundant messages. However, the design of FloodNet has the following disadvantages in practice. Firstly, calculating the number of the secondary neighbors of each peer will consume much bandwidth resource. Furthermore, unstructured P2P networks are so dynamic that the number of the secondary neighbors of each peer is volatile. Last but not least, in FloodNet, the level of the sub-overlay is very deep, leading to long latency. Inspired by the pros and cons of the flooding-based search scheme and FloodNet, we propose PercolationNET — a sub-overlay for providing the guarantee that any object existing in the network can be found with low cost. In the following section, we will describe the overlay structure and maintenance of PercolationNET.

3.1. Overlay structure

There are three principles in designing PercolationNET: (1) In order to achieve optimal effect globally, the design of PercolationNET relies on the global information of the original overlay. (2) To guarantee the full attainability of data objects, the sub-overlay should include all peers in the original overlay. (3) Because of the high transiency of the unstructured p2p networks, PercolationNET must be efficiently maintained. Therefore, only local information is needed for overlay maintenance.

Lv et al. [23] showed that a node with high degree in Gnutella network would most likely experience high query load. Thus, we make two logical assumptions for the sub-overlay. Firstly, the peers with high degree are high capacity ones, called super-peers in this paper. Super-peers take over more responsibilities in the sub-overlay. Secondly, super-peers do not leave the network frequently. Thus the sub-overlay is comparatively stable.

Based on the above principles and assumptions, we construct PercolationNET in three phases. In the first phase, the tree roots of the sub-overlay need to be found. Previous studies [19] have shown that P2P overlay topologies follow the power law properties, which means that a few peers have high degrees. In PercolationNET, we select high-degree peers as super-peers as the tree roots of sub-overlay. We can find these high-degree peers easily by relying on the degree distribution of peers in the original overlay. As shown in Algorithm 1, DThres is a threshold value. That is, a peer whose degree bigger than DThres is defined as a super-peer. The set S is used to store all super-peers.

In the second phase, each ordinary peer probes its level by Algorithm 2. In Algorithm 2, the parameter DetectTTL denotes the number of hops each peer detects. The setting of DetectTTL value needs to ensure that all ordinary peers can find at least one super-peer. We use the minimum hops between each ordinary peer and a certain super-peer as its level. The level of super-peers is zero.

In the third phase, each ordinary peer selects a neighbor as its parent according to Algorithm 3. A peer p's candidate parent peers are its neighbor peers whose level are just one lower than peer p. Each ordinary peer selects one from its candidate parent peers as its parent with probability Pr which can be computed by the degree of peers, as shown in Algorithm 3. Thus, PercolationNET generates multiple unconnected components. In Figure 1, the form of each component in the sub-overlay is a tree, and the root of the tree is a super-peer in original overlay. In the sub-overlay, each tree is composed of one super-peer in the original overlay, ordinary peers (peers other than super-peers) directly or indirectly connecting with the super-peer, and the original existing links among them.

Algorithm 1. SELECT_SUPERPEERS

```
1. N is the set of peers in the original overlay.
2. S is the set of super-peers.
   Its initial value is null.
3. DThres is the threshold value of degree for super-
   peers.
4. initialize DThres according to the degree
   distribution of peers in the original overlay
5. For each peer $q \in N$
6.    If Degree(q) > DThres
      then put peer q into the set S
7. End For
```

Algorithm 2. DETECT_LEVEL(p)

```
1. N is the set of peer p's neighbors.
   M is a null set.
2. S is the set of super-peers.
3. initialize DetectTTL; j = 1; flag = false
4. While j < DetectTTL and flag = false
5.    For each peer $q \in N$
6.       If $q \in S$ then Level(p) = j; flag = true; break;
         //If q is a super-peer, the level of p is the circular
         parameter j and the cycle process terminates
7.       else put the neighbors of peer q into set M
         //If q is not a super-peer, the neighbors of q will
         be detected, so M is used to express the set of the
         neighbors of q
8.    End For
9.    set N = M; M = $\varnothing$; j++
10. End While
```

Algorithm 3. FIND_PARENT(p)

```
1. N is the set of peer p's neighbors.
   M is a null set.
2. obtain the level information of all peers in set N
3. For each peer $q \in N$
4.    If Level(p)-Level(q) = = 1 then
5.       put peer q into set M
6. End For
   // some of peer p's neighbor peers whose level are
   just one lower than peer p are put into set M
7. obtain the degree information of all peers in set M
8. For each peer $k \in M$
9.    compute the probability $\Pr_k = \frac{Degree(k)}{\sum_{i \in M} Degree(i)}$
10.End For
11.select a neighbor j from M as its parent with
   probability $\Pr_j$
```

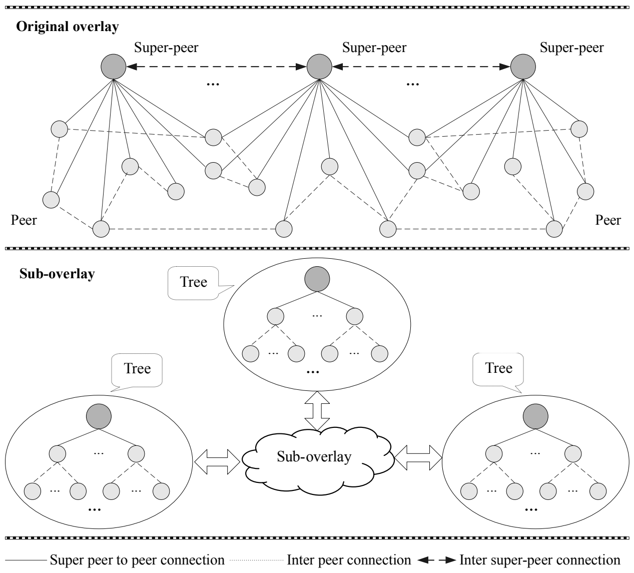

Figure 1. Overlay structure

3.2. Overlay maintenance

3.2.1. Join. A typical unstructured P2P system provides several permanent well-known bootstrap hosts to maintain a list of on-line peers so that a new incoming peer can find an initial host to start its connection by contacting the bootstrap hosts. In an original overlay, a bootstrap host will provide the joining peer a list of active peers with their information. The joining peer tries to construct connections to these peers, and then detects its level by Algorithm 2. In PercolationNET, the new joining peer selects one of its neighbors in original overlay as its parent by Algorithm 3.

3.2.2. Leave or fail. When leaving the network, a peer has to inform its children by sending a leave message. Each peer of the informed children detects its level by Algorithm 2 again, and then selects another neighbor in the original overlay as its parent by Algorithm 3. The failure of a peer in PercolationNET is detected when one of its children misses a sequence of three messages. In the case where a peer detects its parent's failure, it refreshes its level by Algorithm 2, and then selects another neighbor in the original overlay as its parent by Algorithm 3.

The departure or failure of individual peers does not have a disruptive impact on the overlay topology, for messages are routed by many parallel routes in our two-stage search scheme. Therefore, the two-stage search scheme is robust against volatile peers.

3.2.3. Adjust. Since the unstructured P2P networks are self-organized, an individual peer may come, go or fail frequently. As a result, a peer's parent may be not the optimal one in PercolationNET. Therefore, each peer needs to update its parent information periodically. In reality, it is important to find an appropriate interval. However, in our approach, a short update interval will only consume a small bandwidth resource because the adjustment of level and parent for each peer only needs the degree information in the original overlay. At the same time, a large update interval will not cause significant performance degradation due to the two-stage search scheme.

4. Performance evaluation

4.1. Simulation setup

We use the simulator PeerSim [20] for evaluating the performance of PercolationNET. In our simulation, we construct two overlays, original overlay and sub-overlay. Using BRITE [21, 22], we generate the original overlay based on the BA (Barabasi-Albert) model with 10000 nodes and 30000 links. Based on the original overlay, we construct the corresponding sub-overlay, where we define the threshold value DThre as 140. Therefore, there are 8 super-peers for this sub-overlay in our experiments.

For each experiment in the following, every peer, in turn, starts a searching procedure and broadcasts a query message to the network by using flooding with Q(firstTTL, secondTTL). Each peer stores the information of its neighbors in the original overlay and the information of its parent and children in PercolationNET. In the first stage, the message is propagated in the original overlay with firstTTL. When a peer receives the message q(0, secondTTL), the peer will stop broadcasting the message in the original network. Then the message will be broadcasted using flooding along PercolationNET with secondTTL. When a peer receives the message q(0, 0), the peer will stop broadcasting the message in the sub-overlay. The seeds who are the new nodes reached in the last hop of firstTTL are the source nodes of the message in the second stage. We mainly focus on two performance metrics: message overhead and coverage rate within a certain hops. Additionally, we analyze the performance of PercolationNET compared with FloodNet, in terms of coverage rate and message efficiency.

4.2. Seeds and super-peers

We use the term "seeds" to describe the new nodes reached in the last hop of firstTTL. From the previous description, we can see that the number of seeds is an important parameter for message diffusion along PercolationNET. Figure 2 shows the number of seeds with different value of firstTTL in our topology. We can observe that seed amount first increases and then decreases with the increase of firstTTL. This is because flooding in power-law networks is efficient only in earlier stages (with low hops). In the latter stages, the number of the new nodes reached does not increase like the initial stages. This motivates us to use an appropriate firstTTL in the first stage, which can produce enough seeds for the second stage. Moreover, from

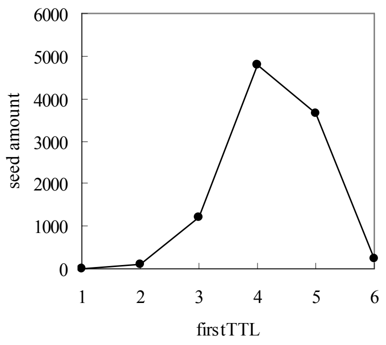

Figure 2. Seed amount in the first stage

the figure, we can see that the optimal value of firstTTL for producing enough seeds is no more than 4 in our topology.

Figure 3 shows the average number of super-peers reached in the first stage. We can see that all super-peers are covered in the first stage when the value of firstTTL is 4. If the coverage scope of flooding in the first stage includes all super-peers, seeds will spread over all trees of sub-overlay unquestionably. Consequently, the search can spread to all peers with a low secondTTL in the second stage. However, though not all super-peers are included in the coverage scope of the first stage, the seeds can also spread over all trees as long as they are sufficiently decentralized. We will further test it by experiments in the following.

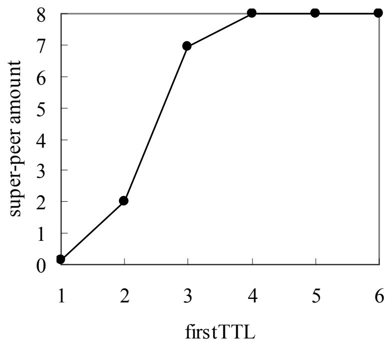

Figure 3. Super-peer amount in the first stage

4.3. Message overhead

The goal of PercolationNET is to reduce the message overhead as much as possible while retaining the same coverage scope. Figure 4 lists the average message overhead per query with the increase of secondTTL in the second stage for different arrangements (firstTTL, *). According to analysis of seeds and super-peers in the above section, we know that the optimal value of firstTTL is not more than 4. Hence in the following experiments, we use (1, *), (2, *), (3, *), (4, *). Furthermore, the whole coverage of the network is used as the baseline to set the stopping hops in the second stage (except for (1, *) and (2, *)).

From the figure, we can make the following observations. First, as the firstTTL increases, the average message overhead

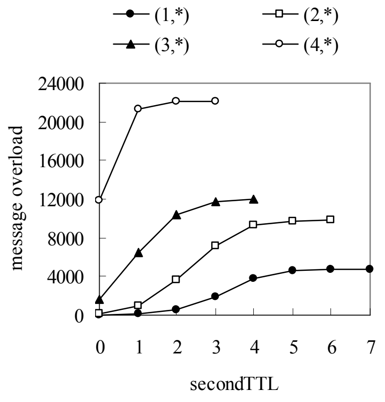

Figure 4. Message overhead in the second stage

increases exponentially. In contrast, once the flooding is switched from the original overlay in the first stage to PercolationNET in the second stage, the rising speed of message overhead is slow. Therefore, this means that PercolationNET is able to eliminate a large number of redundant messages by introducing several additional hops. Second, the bigger the value of firstTTL is, the larger the message overhead becomes. For example, the simulation shows that the final message overhead of (4, *) is twice as much as that of (3, *). Third, at the latter hops of all arrangements in this figure, the message overhead nearly stops rising. The reason is that PercolationNET is made of trees. When the message reaches the leaf nodes, it will not be broadcasted anymore. Lastly, at the first hop of (4, *), the increment of message overhead is comparatively large. This phenomenon can be explained with the reason that all of seeds produced in the first stage broadcast the query message to all of their neighbors along PercolationNET at the initial one hop in the second stage, thus generating many message overheads. In fact, the phenomenon can also be observed in (3, *). However, the seeds in (3, *) are much less than that of (4, *), as shown in Figure 2, so it is not obvious.

4.4. Coverage rate

Coverage rate measures the ratio of the number of visited peers to the whole number of peers in the network. Figure 5 lists the coverage rate growth with the increase of secondTTL for different (firstTTL, *) arrangements. Based on the figure, we can carry out the following observations. First, the smaller value of firstTTL is, the bigger value of secondTTL is needed to obtain a similar coverage. This is because a mass of seeds produced in the first stage can saturate the network quickly.

Besides, among different arrangements (firstTTL, *), not every firstTTL value can reach the whole coverage. For example, (1, *) can only achieve 47 percent of the whole coverage. The reason is that PercolationNET is formed by multiple trees. If the number of seeds is not large enough, seeds can't be dispersed into all trees during the first stage. Finally, combination of this figure and the above figure 4, we can see that the arrangements of (3, *) strike a good balance between message overhead and search coverage.

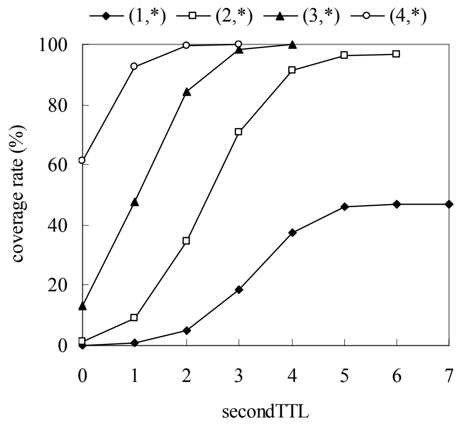

Figure 5. Coverage rate

To compare the performance of PercolationNET with FloodNet, we carry out the contrasting experiments using (3, *), as shown in Figure 6. Obviously, we can see that the performance of coverage rate in PercolationNET is superior to that of FloodNet all the time. This is because the level in FloodNet is much deeper than that of PercolationNET. Hence, it is slow for a message to spread to the whole network.

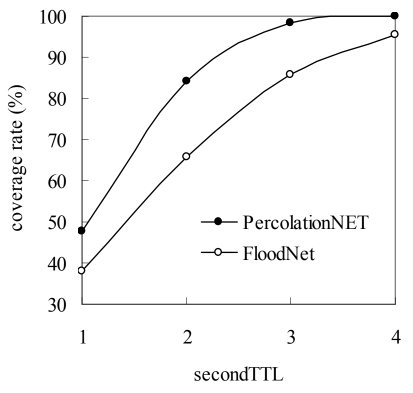

Figure 6. Compare coverage rate with (3, *)

4.5. Message efficiency

Message efficiency is the ratio between the number of peers reached and the number of forwarded messages. The optimal efficiency is one if there are no redundant messages. Figure 7 shows the message efficiency using (3, *). In the figure, we can see that PercolationNET is superior to FloodNet in message efficiency. Another observation is that the efficiency becomes worse with the first hop in the second stage, and then it becomes better and better. For example, the message efficiency is nearly 83 percent in (3, 0), whereas it is less than 74 percent in (3, 1). This is consistent with what we have observed in section 4.3.

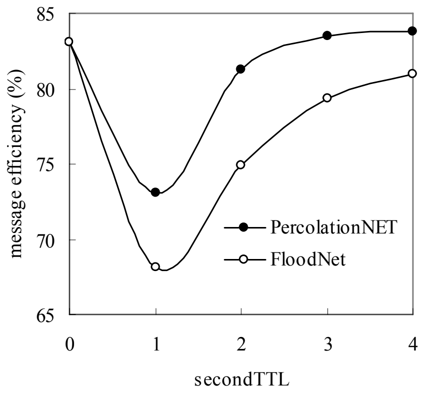

Figure 7. Compare message efficiency with (3, *)

5. Conclusion and Future Work

In this paper, we build PercolationNET upon the existing P2P overlay. We apply the information of super-peers in the existing overlay to construct the multi-tree sub-overlay, while the super-peers are regarded as the tree roots in PercolationNet. Accordingly, the search process is divided into two stages. The experiments show that the proposed sub-overlay structure is more efficient than the existing FloodNet scheme, including coverage speed and message efficiency. Although we propose a promising sub-overlay comparing with FloodNet, and use flooding scheme to investigate its efficiency, there are still further problems to be explored. Firstly, we can take other search schemes into account since we only need to spread query message to all trees of PercolationNET in the first stage. In addition, we research how the parameters firstTTL and secondTTL would affect the performance of our approach, but not exploit how the size of P2P overlay affects its performance. We will address these issues in our future work.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (60773191, 70771043, 60873225), the National High Technology Research and Development Program of China (863 Program) (2007AA01Z403).

References

[1] Gnutella, http://www.Gnutella.com/

[2] KaZaA, http://www.kazaa.com, 2007

[3] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," Proc. ACM SIGCOMM, San Diego, California, USA, 2001

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," Proc. ACM SIGCOMM, San Diego, California, USA, pp.161-172, 2001

[5] Hongbo Jiang and Shudong Jin, "Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-Peer Networks," Proc. 13th IEEE International Conference on Network Protocols (ICNP'05), 2005

[6] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," Proc. 22nd IEEE Intl' Conf. Distributed Computing Systems (ICDCS'02), July 2002

[7] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," Proc. 16th ACM Int'l Conf. Supercomputing (ICS'02), 2002.

[8] Luo Jiaqing, Zhou Shijie, Wu Chunjiang, Deng Yiyi, and Yang Xiaoqian, "Adaptive Flooding Routing Algorithm in Unstructured P2P," Proc. 4th IEEE Intl' Conf. Communications, Circuits and Systems, pp. 1557-1561, 2006

[9] Iraklis Angelos Klampanos, Victor Poznanski, "Retrieval Efficiency in Peer-to-Peer Networks with Replication Restrictions," Proc. 16th International Workshop on Database and Expert Systems Applications (DEXA'05), 2005

[10] Sabu M. Thampi, K. Chandra Sekaran, "Autonomous Data Replication Using Q-Learning for Unstructured P2P Networks," Proc. 6th IEEE International Symposium on Network Computing and Applications (NCA'07), 2007

[11] Edith Cohen, Scott Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks," Proc. ACM SIGCOMM, Pittsburgh, Pennsylvania, USA, 2002

[12] Taizo Yamada, Kenro Aihara, Atsuhiro Takasu, and Jun Adachi, "Adaptive Replication Method Base on Peer Behavior Pattern in Unstructured Peer-to-Peer Systems," Proc. 21th IEEE Intl' Conf. Data Engineering (ICDE'05), 2005

[13] C. Wang, L. Xiao, Y. Liu and P. Zheng, "Distributed Caching and Adaptive Search in Multilayer P2P Networks," Proc. 24nd IEEE Intl' Conf. Distributed Computing Systems (ICDCS'04), pp. 219-226, 2004

[14] Hasan Guclu, Murat Yuksel, "Scale-Free Overlay Topologies with Hard Cutoffs for Unstructured Peer-to-Peer Networks," Proc. 27th International Conference on Distributed Computing Systems (ICDCS'07), 2007

[15] Mudhakar Srivatsa, Bugra Gedik, Ling Liu, "Large Scaling Unstructured Peer-to-Peer Networks with Heterogeneity-Aware Topology and Routing," Proc. IEEE Transctions on Parallel and Distributed Systems, Vol. 17, No. 11, 2006

[16] Y. Liu, L. Xiao, X. Liu, L.M. Ni, etc, "Location Awareness in Unstructured Peer-to-Peer Systems," IEEE Trans. Parallel and Distributed Systems, Vol. 16, No. 2, pp. 163-174, 2005

[17] Y. Liu, L. Xiao, and L.M. Ni, "Building a Scalable Bipartite P2P Overlay Network," IEEE Trans. Parallel and Distributed Systems, Vol. 18, No. 9, pp. 1296-1306, 2007

[18] Song Jiang, Lei Guo, Xiaodong Zhang, etc, "LightFlood: Minimizing Redundant Messages and Maximizing the Scope of Peer-to-Peer Search," IEEE Trans. Parallel and Distributed Systems, Vol. 19, No. 5, pp. 601-614, 2008

[19] Hasan Guclu, Murat Yuksel, "Scale-Free Overlay Topologies with Hard Cutoffs for Unstructured Peer-to-Peer Network," Proc. 27th IEEE Intl' Conf. Distributed Computing Systems (ICDCS'07), 2007

[20] PeerSim, http://peersim.sourceforge.net/

[21] BRITE, http://www.cs.bu.edu/brite/, 2007

[22] Alberto Medina, Anukool Lakhina, Ibrahim Matta, et al. BRITE: Universal Topology Generation from a User's Perspective. Technical Report BUCS-TR-2001-003, Boston University, April 2001

[23] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002