PercolationNET: a multi-tree P2P overlay network supporting high coverage search

Ruixuan Li[a*], Cuihua Zuo[a1], Haiying Shen[b2], Kunmei Wen[a3] and Xiwu Gu[a4]

[a]*College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P.R.China;* [b]*Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA*

(Received 1 June 2009; final version received 1 August 2009)

Flooding with a time-to-live (TTL) constraint is a popular algorithm in unstructured peer-to-peer (P2P) networks. However, blind flooding may cause a large amount of network traffic. Moreover, it cannot guarantee acquiring all 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.

Keywords: peer-to-peer network; high coverage search; multi-tree architecture; sub-overlay; flooding-based scheme; tree-based scheme

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 tables (DHTs). 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. Consequently, unstructured networks are becoming more and more popular as they are flexible to be optimized for specific applications [5].

The predominating search mechanism in unstructured networks is message flooding with a TTL restriction. This simple method does not provide guarantee that an object existing in the network can be found. Moreover, flooding does not scale

* Corresponding author. Email: rxli@hust.edu.cn

well in terms of message overhead, since each query may generate a significant amount of traffic, especially in a system with a high-connectivity topology. Although some schemes have been proposed to restrict flooding, such as the use of a flexible or alterable TTL value and selective neighbours, these schemes are only beneficial to popular data objects in the unstructured networks since the queries are restricted in a certain scope.

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 ensures 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. The three different kinds of approaches can be used together to complement each other.

Unlike pure flooding, which starts with a fixed TTL and sends query message to all neighbours, modified flooding takes more dynamic factors into consideration to reduce traffic overhead while maintaining certain search quality. For example, in Directed BFS [6], each peer maintains statistic information based on a number of metrics such as the degree of neighbours. A peer selects a subset of its neighbours, such as the neighbours 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 neighbours, 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 literature [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 layer 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 neighbour with the maximum secondary degree (i.e., the sum of the degrees of a peer's neighbours). 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 neighbours (i.e., the neighbours of a peer's neighbours). 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, since each peer chooses one of its neighbours as parent according to local information, the design of the overlay FloodNet is not optimal in the global. In addition, calculating the number of the secondary neighbours of each peer will consume much bandwidth resource. Furthermore, unstructured P2P networks are so dynamic that the number of the secondary neighbours 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 several principles in constructing PercolationNET: (1) Most research on overlay construction only considered the local information of each peer. Thus, they can achieve preferable effect in the local but not globally. In order to achieve optimal effect globally, the design of PercolationNET relies on the global information of the original overlay. (2) To guarantee that all required data objects can be found, the sub-overlay should include all peers in the original overlay. In other words, each peer exists in one tree of PercolationNET. (3) Because of the high transiency of the unstructured p2p networks, PercolationNET must be efficiently maintained. Therefore, when peers leave, join, or even fail, only local information is needed for overlay maintenance.

Lv et al. [19] showed that a high degree node in Gnutella network would most likely experience high query load. Generally, high degree nodes have high capability to handle a great deal of 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 relatively 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 [14] 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 nodes, the super-peers, as the tree roots of sub-overlay. We can find these high-degree nodes easily by relying on the degree distribution of peers in the original overlay. As shown in Algorithm 1, *DThres* is a threshold value for super-peers, that is, a peer whose degree larger than *DThres* is defined as a super-peer. The set *S* is used to store all super-peers. Since the discovery of super-peers is based on the global information of the original overlay, the construction of PercolationNET is from the overall consideration.

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 value of *DetectTTL* varies with different topology sizes and their average connectivity degrees. The setting of *DetectTTL* value needs to ensure that all ordinary peers can find at least one super-peer. We use the minimum number of 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 neighbour as its parent according to Algorithm 3. A peer *p*'s candidate parent peers are its neighbour 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 this paper, the original overlay is generated directly by the topology generation based on Barabasi-Albert (BA) model. Hence, the degree of peers in this overlay must obey the power-law distribution. We classify the peers in the original overlay into two types: super-peers who have high degree and ordinary peers (peers other than super-peers). As shown in Figure 1(a), there are three super-peers SP_1, SP_2, SP_3, and nineteen ordinary peers *P*a, *P*b, …, *P*x. In Figure 1(b), 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 directly or indirectly connecting with the super-peer, and the original existing links among them. Concretely, super-peers SP_1, SP_2 and SP_3 in ordinary overlay become the tree roots in the sub-overlay, and each ordinary peer detects its level and selects a neighbour in the original overlay as its parent in the sub-overlay according to Algorithm 2 and Algorithm 3.

For example, the thirteen ordinary peers *P*a, *P*d, *P*e, *P*g, *P*h, *P*k, *P*l, *P*m, *P*j, *P*n, *P*r, *P*t and *P*x have direct links with one of three super-peers. Therefore, their level is 1. *P*a, *P*d and *P*e only have direct link with one super-peer SP_1, so they select SP_1 as their parent undoubtedly. Similarly, *P*k, *P*l, *P*m select SP_2 as their parent, and *P*n, *P*r, *P*t, *P*x select SP_3 as their parent. However, as to peers *P*g, *P*h and *P*j, they have direct links with two super-peers. Thus, they have to choose one as their parent according to the probability *P*r described in Algorithm 3. The six ordinary peers *P*b, *P*c, *P*f, *P*o, *P*i and *P*s need two hops to reach one super-peer. Hence, their level is 2. In all neighbours of peer *P*c, only the level of peer Pa is just one lower than peer *P*c, Hence, *P*c selects *P*a as its parent undoubtedly. Similarly, *P*s selects *P*t as its parent. However, as to peers *P*b, *P*f, *P*o and *P*i, they have more than one choice, because they all have two neighbours whose levels are just one lower than them. Hence, they need to choose one as their parent according to the probability *P*r as well.

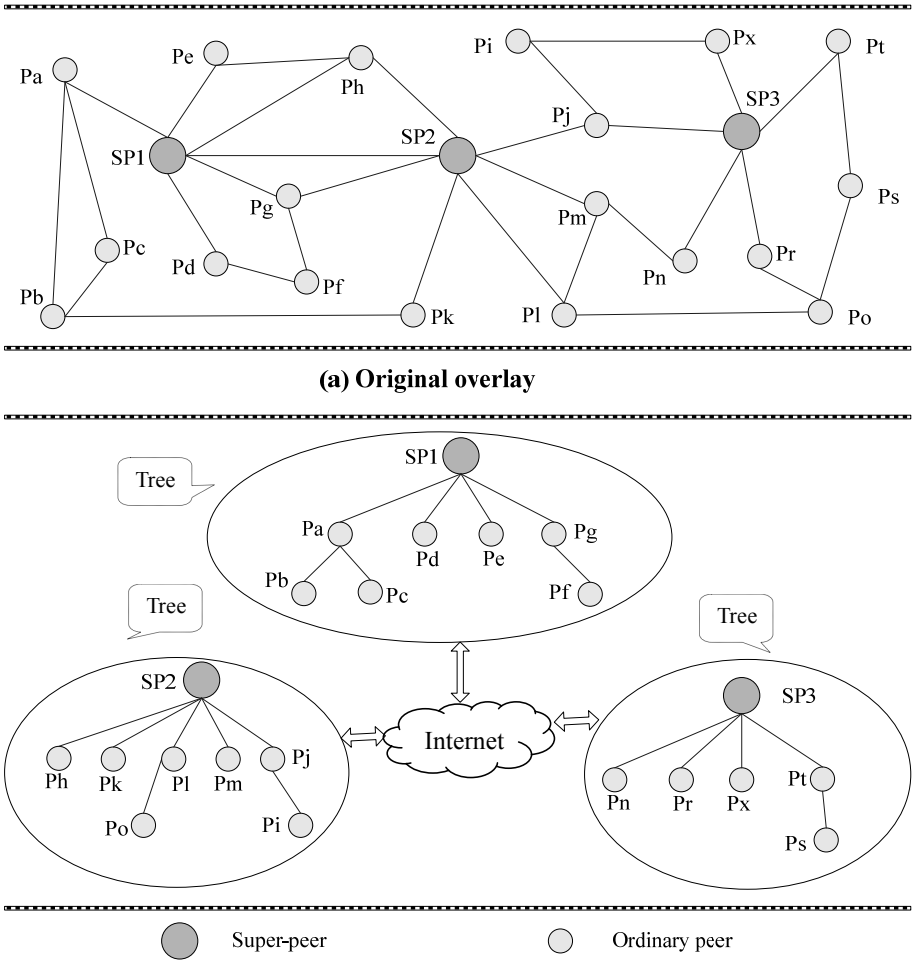


Figure 1. Original overlay structure of P2P network and sub-overlay structure of PercolationNET

Algorithm 1. SELECT_SUPERPEERS

1. N is the set of peers in the original overlay.
2. S is the set of super-peers with the initial value of 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 neighbours.
   M is a null set. //M is used to store the temporary information
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 peer p is the circular parameter j and the cycle process terminates*
7. **else** put the neighbours of peer q into set M
   //*If q is not a super-peer, the neighbours of q will be detected, so M is used to express the set of the neighbours 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 neighbours.
   M is a null set. //M is used to store the candidate parent peers of peer p
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 neighbour 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 $\Pr_k = \dfrac{Degree(k)}{\sum_{i \in M} Degree(i)}$

10.**End For**
11.Select a neighbour *j* from *M* as its parent with probability $\Pr_j$

3.2 Overlay maintenance

3.2.1 Joining

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 neighbours in original overlay as its parent by Algorithm 3. The peers (except the father of the new peer) who are connected by the new peer will detect their level over again. Once the level of a peer is updated, the level of its children will be updated accordingly. The algorithm of a new peer joining is shown in Algorithm 4.

Figure 2(a) and 2(b) show the process of a new peer joining the original overlay and sub-overlay, respectively. The peer *P* chooses four peers *P*1, *P*2, *P*3 and *P*4 to establish the connection in the original overlay. In the sub-overlay, it select peer *P*1 from the four peers as its father according to Algorithm 3. Additionally, the rest peers *P*2, *P*3 and *P*4 need to detect their level afresh.

3.2.2 Leaving

There are two types of peers in PercolationNET: super-peer and ordinary peer. The ordinary peer can be further classified into leaf peer and middle peer. When a super-peer leaves the network, the sub-overlay needs to be reconstructed according to the construction of PercolationNET. When a middle peer leaves the network, it has to inform its parent and children by sending a leave message. Each peer of the informed children detects its level by Algorithm 2 over again, and then selects another

Algorithm 4. Joining(P)

1. Randomly choose m nodes {*P*1, *P*2, …, *P*m} in original overlay as the new peer *P*'s neighbours;
2. Detect the level of peer *P* using the DETECT_LEVEL function in Algorithm 2;
3. Peer *P* selects a neighbour peer *P*i from {*P*1, *P*2, …, *P*m} as its parent according to Algorithm 3;
4. Define a set $S = \{P1, P2, \ldots, Pm\} - \{Pi\}$;
   //*S is the set of the peers (except the father Pi of the new peer P) who are connected by the new peer*
5. **For** each peer $Pj \in S$
6. **If** (Level(*P*j)>Level(*P*)+1) **then**
7. Parent(*P*j) = *P*;
   //*alter the parent node of Pj*
8. Level(*P*j) = Level(*P*)+1;
   //*alter the level information of Pj*
9. Alter the level of descendants of peer *P*j;
10. **End If**
11. **End For**

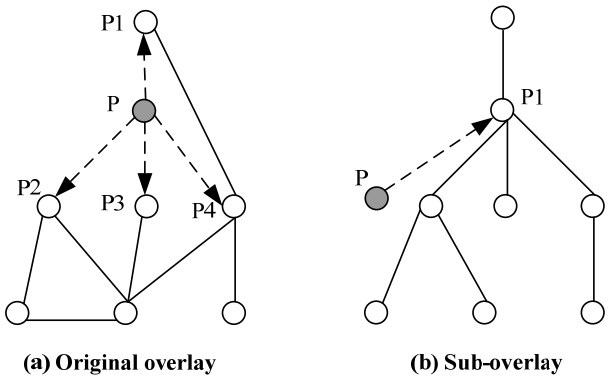


Figure 2. Peer joining process in original overlay and sub-overlay

neighbour in the original overlay as its parent by Algorithm 3. When a leaf peer leaves the network, it has to inform its parent by sending a leave message. The algorithm of leaving network for a peer is shown in Algorithm 5.

Figure 3(a) and 3(b) show the process of a leaf peer and a middle peer leaving the sub-overlay, respectively. In Figure 3, *P*1 is the father of peer *P* and *P*2, *P*3 are its children. We can see that once a middle peer *P* leaves the network, its children *P*2 and *P*3 need to find new peers as their parents by Algorithm 2 and Algorithm 3. Additionally, the children of *P*2 and *P*3 are also need to update their level accordingly.

3.2.3 Failure Recovery

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 neighbour in the original overlay as its parent by Algorithm 3. Additionally, its children need to update their level accordingly.

Although a number of peers may fail at the same time, the query message can still span the entire network. This is because many seeds which will be described in

Algorithm 5. Leaving(P)

1. **If** (*P* is a leaf peer)
2. Peer *P* notices its parent;
3. Its parent deletes the information of the leaving peer;
   //*If the leaving peer is a leaf peer, the peer only needs to notice its parent to update its information.*
4. **Else If** (*P* is a super-peer)
5. The sub-overlay PercolationNET will be reconstructed according to the construction of PercolationNET;
6. **Else**
   // *The leaving peer is a middle peer.*
7. Put the children of peer *P* into a set *M*;
8. **For** each peer $Pj \in M$
9. Detect its level according to Algorithm 2;
10. Select a neighbour peer as its parent according to Algoritm 3;
11. Alter the level of descendants of peer *P*j;
12. **End For**
13. **End If**

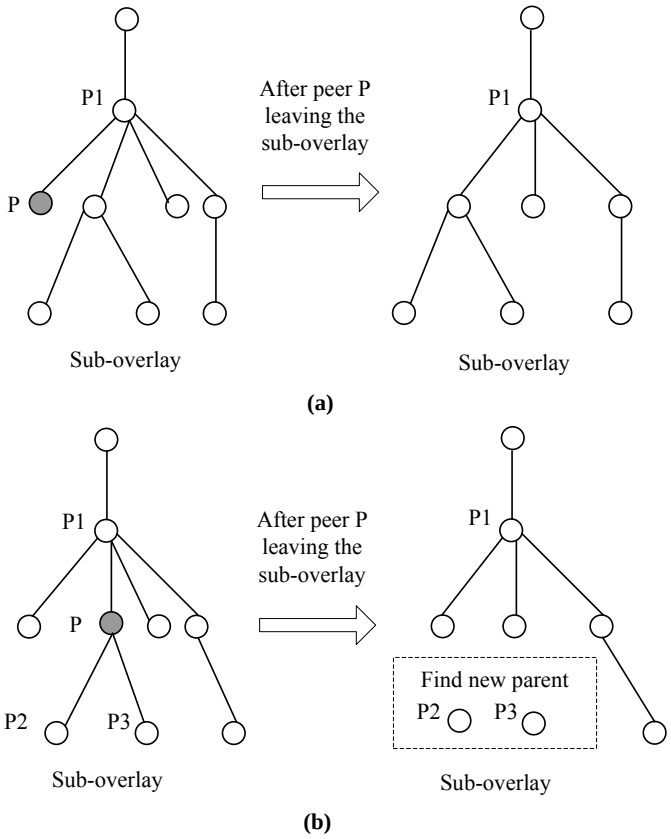


Figure 3. Peer leaving process in sub-overlay

Section 4.2 can be created during the first stage search of a message. The departure or failure of individual peers does not have a disruptive impact on the overlay topology since messages are routed by many parallel routes in the proposed two-stage search scheme. Therefore, the two-stage search scheme is robust against volatile peers.

3.2.4 Adjustment

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. A short update interval can produce accurate information about the PercolationNET topology, but frequently updating messages consumes extra bandwidth resource. A long update interval has lower communication costs but may not reflect the latest network topology. In reality, it is important to find a balanced point since the network condition is dynamic.

However, 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.

3.3 Message routing

Each peer needs to store two aspects of information: the information of neighbour peers in the original overlay and the information of its father and children in the sub-overlay PercolationNET. Correspondingly, the routing process for query messages is divided into two stages. In the first stage, a query message is flooded in the original

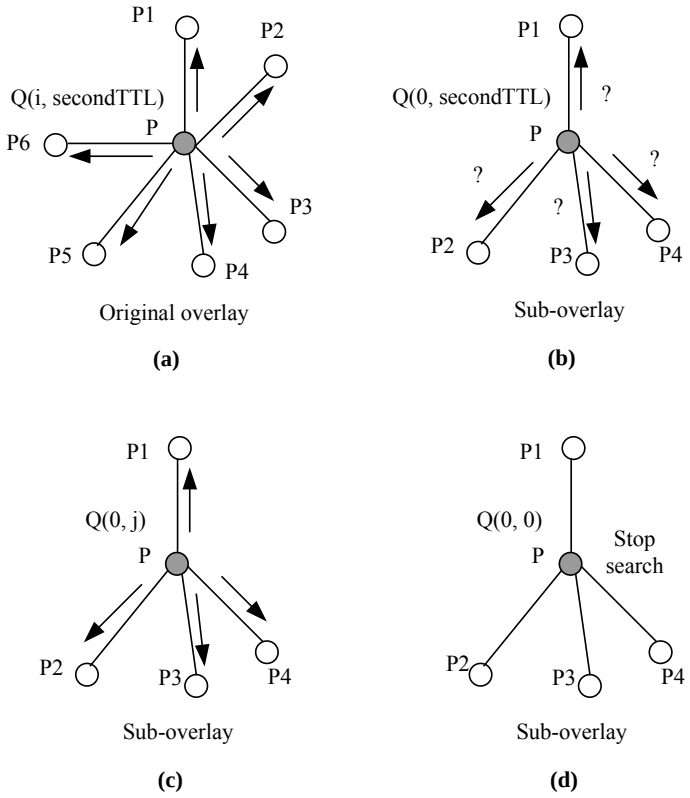


Figure 4. Search process in original overlay and sub-overlay

overlay network. Then in the second stage, the query message is broadcasted along the sub-overlay. Hence, each query message needs two initial TTLs: *Q*(*firstTTL*, *secondTTL*), where *firstTTL* denotes the broadcasting hops of message in the first stage and *secondTTL* means the propagating hops of message in the second stage. There are four situations as described in the following when a peer receives a query message.

(1) When a peer receives a message *Q*(*i*, *secondTTL*), 0<*i*≤*firstTTL*, it will reduce i by 1, and transmit this message to all its neighbours along original overlay. As shown in Figure 4(a), *P*1, *P*2, *P*3, *P*4, *P*5 and *P*6 are all neighbours of peer *P* in original overlay. If peer *P* receives a message *Q*(i, *secondTTL*), 0<*i*≤*firstTTL*, it will transmit the message *Q*(i-1, *secondTTL*) to *P*1, *P*2, *P*3, *P*4, *P*5 and *P*6.

(2) When a peer receives a message *Q*(0, *secondTTL*), it will stop the propagation of this message in original overlay and judge whether the message is a new message. If it is the first time that *P* receives the message *Q*, the peer will become a seed (we use the term "seed" to describe the new node reached in the last hop of *firstTTL*) and further transmit the message along the sub-overlay. Otherwise, the search will be terminated. As shown in Figure 4(b), *P*1 is the father of peer *P* and *P*2, *P*3, *P*4 are its children in the sub-overlay. If peer *P* receives a message *Q*(0, *secondTTL*) and it is the first time to receive the message *Q*, it will transmit the message to its father *P*1 and children *P*2, *P*3, *P*4. However, if *P* has ever received the message Q, the search will be terminated.

(3) When a peer receives a message *Q*(0, j), 0<*j*<*secondTTL*, it will transmit this message to all its neighbours along the sub-overlay, as shown in Figure 4(c).

(4) When a peer receives a message *Q*(0, 0), it will stop the search process, as shown in Figure 4(d).

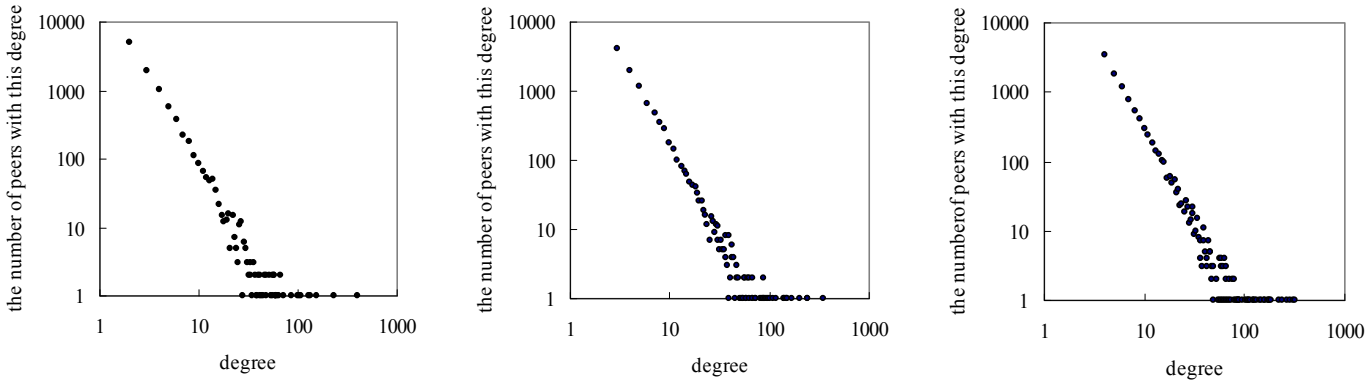
4. Performance evaluation

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 Barabasi-Albert (BA) model with 10000 nodes. In order to learn how the connectivity of an overlay topology affects the performance of PercolationNET, we use different average number of links 2, 3, 4, named Top1, Top2 and Top3, respectively. The node degree distributions of the three overlay topologies are shown in Figure 5. Based on each original overlay, we also construct the corresponding sub-overlay by the scheme described in Section 3, where we define the threshold value of degree for super-peers' *DThre* as 100 in Top1, 141 in Top2, and 178 in Top3. Therefore, there are 8 super-peers for these three topologies 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 (*firstTTL*, *secondTTL*). Each peer stores the information of its neighbours in the original overlay and the information of its parent and children in PercolationNET overlay. In the first stage, the message is propagated in the original overlay with *firstTTL*. When the value of hops in the first stage is equal to the value of *firstTTL*, the peers who receive the message in the last hop will stop broadcasting the message in the original network. Then the message will be broadcasted using flooding along the sub-overlayPercolationNET with *secondTTL*. 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. When the hops in the second stage reach the value of *secondTTL*, the search process will stop. It doesn't need to search a concrete object while broadcasting the message in the network, for the purpose of our evaluation is to obtain the statistics of search scope and message overhead. All data shown in the following figures are average values.

In this paper, we mainly focus on two performance metrics: message overhead and coverage scope within a certain hops. Additionally, we analyze the performance of PercolationNET compared with FloodNet, in terms of coverage rate, coverage growth rate, and message efficiency.



(a) Top1 (b) Top2 (c) Top3

Figure 5. Degree distribution in different topologies

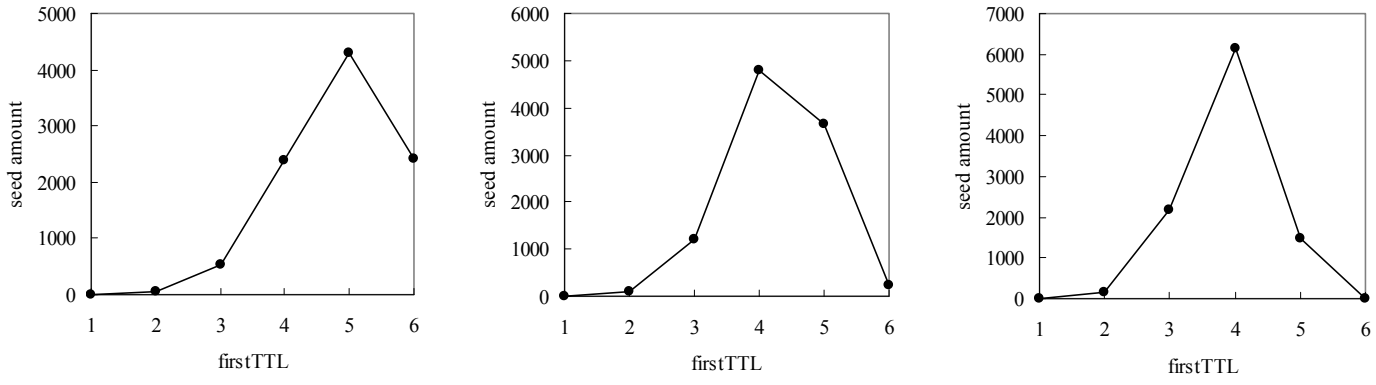
4.2 Seeds and super-peers

From the previous description, we can see that the number of seeds is an important parameter for message propagation along PercolationNET. Figure 6 shows the number of seeds with different value of *firstTTL* in the three topologies. 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, the higher the connectivity of the overlay topology is, the smaller the value of *firstTTL* for producing the most seeds will be. From the figure, we can see that the optimal value of *firstTTL* for producing enough seeds is no more than 5 in Top1, 4 in Top2 and Top3.

Figure 7 shows the average number of super-peers reached in the first stage. From the figure, we can see that the higher the connectivity is, the quicker the super-peers are covered. In addition, all super-peers are covered in the first stage when the value of *firstTTL* is 5 in Top1, 4 in Top2 and Top3. If the coverage scope of flooding in the first stage includes all super-peers, seeds will spread over all trees of PercolationNET 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.

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 8 lists the average message overhead per query with the increase of *secondTTL* in the second stage for different arrangements (*firstTTL*, *) in three topologies. According to analysis of seeds and super-peers in the above section, we know that the optimal value of *firstTTL* is not more than 5 in Top1, 4 in Top2 and Top3. Hence in the following experiments, we



(a) Top1 (b) Top2 (c) Top3

Figure 6. Seed amount in the first stage

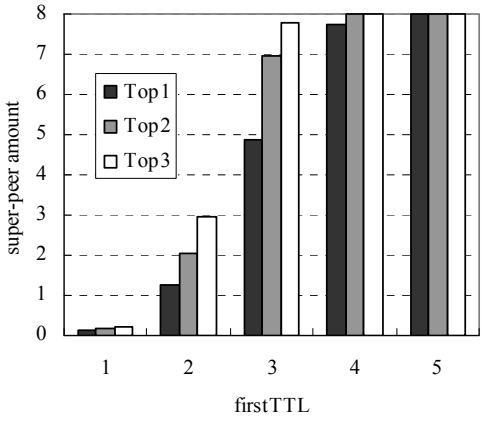
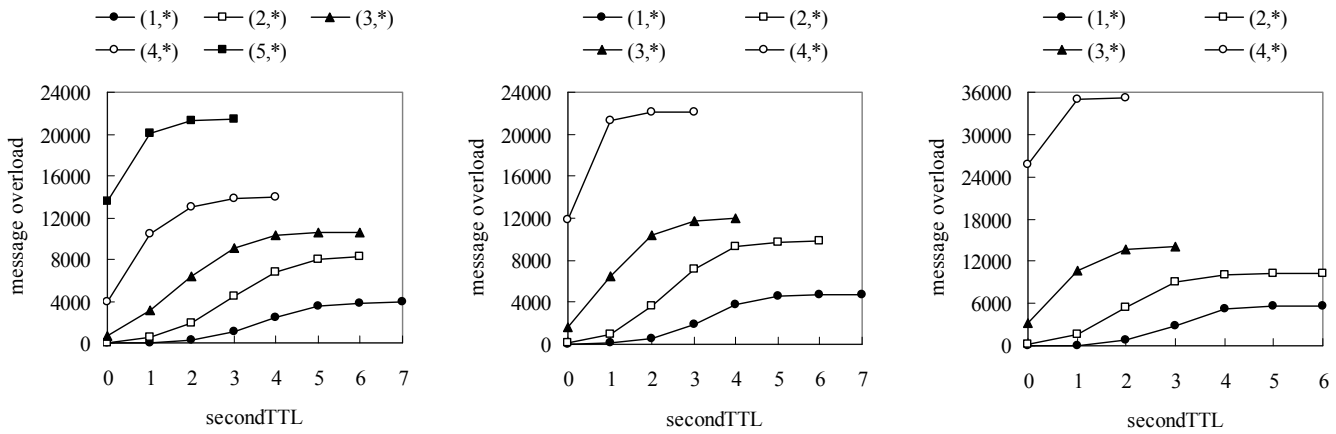


Figure 7. Super-peer amount in the first stage

use (1, *), (2, *), (3, *), (4, *), (5, *) in Top1, (1, *), (2, *), (3, *), (4, *) in Top2 and Top3. 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, *) which can not spread to all nodes).

From the figure, we can make the following observations. First, as the *firstTTL* increases, the average message overhead increases exponentially, especially in the high connectivity topology Top3. 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 (5, *) is twice as much as that of (3, *) on topology Top1, and the final message overhead of (4, *) is nearly three times as much as that of (3, *) on topology Top3. 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, *) in three topologies and (5, *) in Top1, the increment of message overhead is comparatively



(a) Top1 (b)Top2 (c)Top3

Figure 8. Message overhead in the second stage

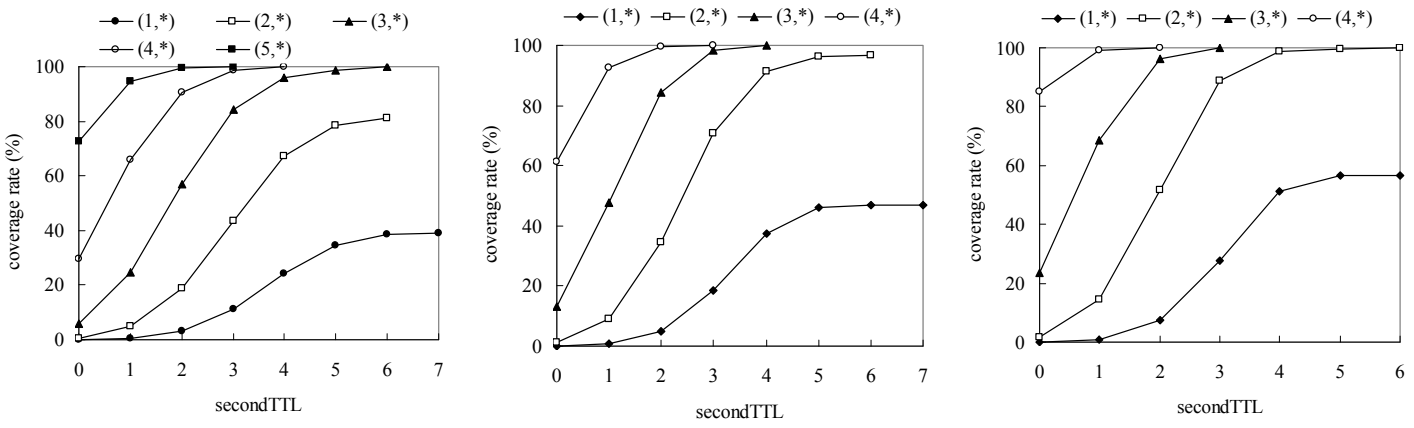
large. This phenomenon can be explained with the reason that all of seeds produced in the first stage broadcast the query message to all of their neighbours along PercolationNET at the initial one hop in the second stage, thus generating many message overheads. In fact, the phenomenon can also be observed in (3, *). However, the seeds in (3, *) are much less than that of (4, *) in these three topologies, as shown in Figure 6, so it is not obvious.

4.4 Coverage rate

Coverage rate measures the ratio of the coverage to the whole number of peers in the network. Figure 9 lists the coverage growth with the increase of *secondTTL* for different (*firstTTL*, *) arrangements. Based on the figure, we can carry out the following observations. First, in each topology, the smaller value of *firstTTL* is, the bigger value of *secondTTL* is needed to obtain a similar coverage. For example, in Top3, (2, *) takes 6 hops to reach the whole coverage, whereas (4, *) takes only 2 hops to retain the same coverage. This is because a mass of seeds produced in the first stage can saturate the network very quickly.

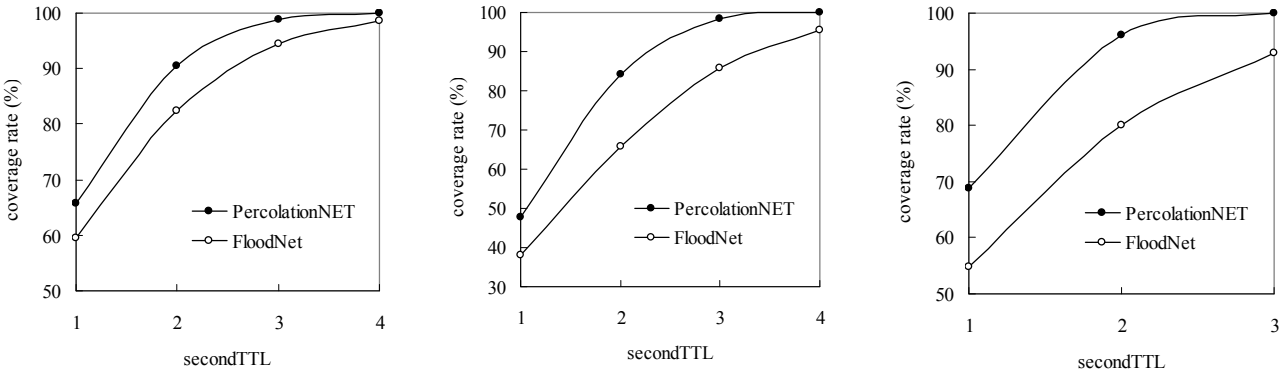
Besides, among different arrangements (*firstTTL*, *), not every *firstTTL* value can reach the whole coverage. For example, in Top1, (1, *) can only achieve 39 percent of the whole coverage, whereas (2,*) only has about 82 percent of the whole coverage. The reason is that PercolationNET is formed by 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 8, we can see that the arrangements of (4, *) strike a good balance between message overhead and search coverage in Top1 and (3, *) that of Top2 and Top3. For instance, in Top3, the message overhead of (4, 2) is 35200, whereas the message overhead of (3, 3) is only 14000 with the same coverage.

To compare the performance of PercolationNET with FloodNet, we carry out the contrasting experiments using (4, *) in Top1, (3, *) in Top2 and Top3, as shown in Figure 10. 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



(a) Top1 (b) Top2 (c) Top3

Figure 9. Coverage rate in the second stage



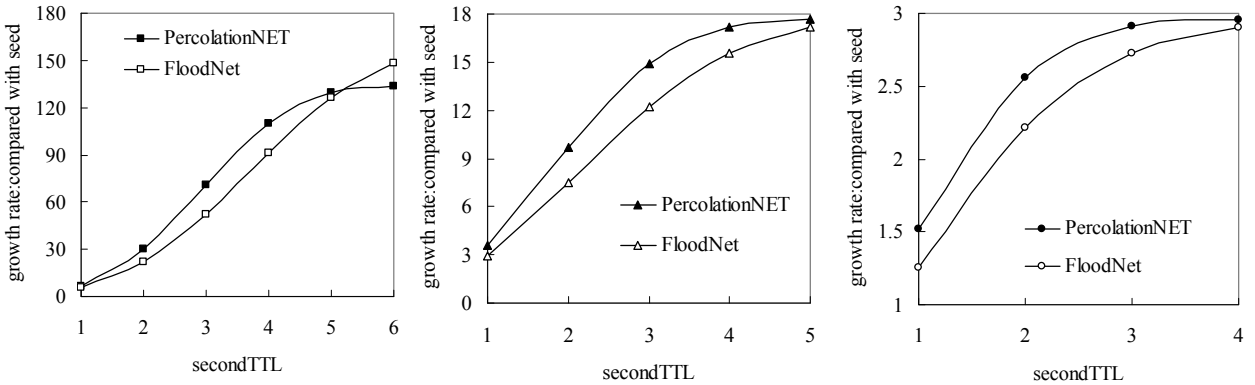
(a) (4, *) in Top1 (b) (3, *) in Top2 (c) (3,*) in Top3

Figure 10. Coverage rate of PercolationNET in comparison with that of FloodNet

message to spread to the whole network. What's more, with the increase of topology connectivity, the performance differences increase between PercolationNET and FloodNet. In other words, PercolationNET is more suitable for the topology with high connectivity than FloodNet.

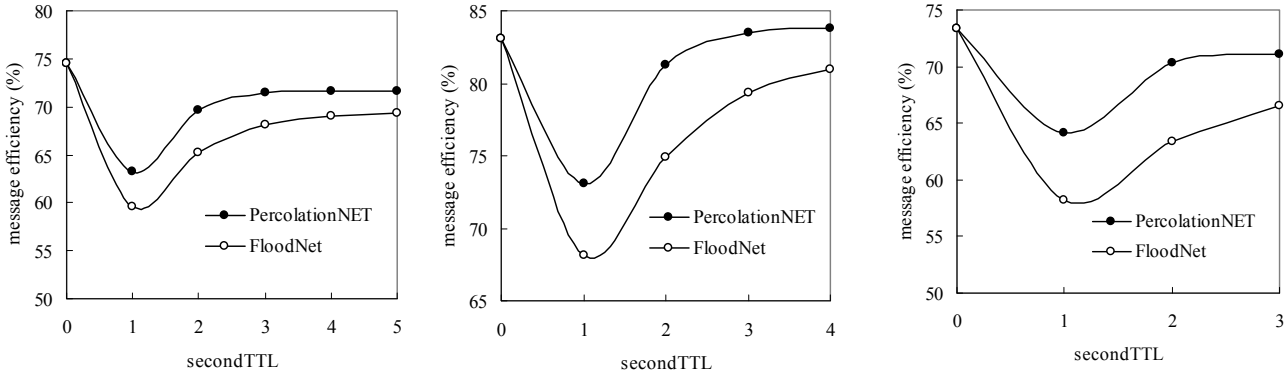
4.5 Coverage growth rate

The coverage growth rate compared with seeds represents the ratio of the growth of the number of peers reached between (i, j) and (i, 0) hops to the number of seeds when firstTTL equals i. The graph in Figure 11 shows the coverage growth rate at various arrangements (2, *), (3, *) and (4, *) in Top1 with low connectivity. The seeds are 60, 525, and 2376 in (2, *), (3, *), and (4, *), respectively, as shown in Figure 6(a). It is clear that the increase in PercolationNET is more obvious than that of FloodNet with (3, *) and (4, *). However, in (2, *), when secondTTL equals 6, the coverage growth rate in FloodNet exceeds that of PercolationNET. This is because seeds are not large enough to disperse into all trees in PercolationNET. However, once the seeds are enough in (3, *) and (4, *), the performance of coverage growth rate in PercolationNET will be superior to that of FloodNet all the time.



(a) (2, *) in Top1 (b) (3, *) in Top1 (c) (4,*) in Top1

Figure 11. Coverage growth rate of PercolationNET in comparison with that of FloodNet



(a) (4, *) in Top1 (b) (3, *) in Top2 (c) (3,*) in Top3

Figure 12. Message efficiency of PercolationNET in comparison with that of FloodNet

Carefully examining the figure, we observe that the coverage growth for the first several hops is faster than that of the latter in PercolationNET, especially in (3, *) and (4, *). This is because Percolation is made of several trees. After several hops of flooding along original overlay in the first stage, the query messages reach most of tree roots in (3, *) and (4, *), as show in Figure 7. Starting from the next hop along PercolationNET in the second stage, the tree roots start flooding the message down to their entire trees.

4.6 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 12 shows the message efficiency using (4, *) in Top1, (3, *) in Top2 and Top3. In the figure, we can see that PercolationNET is superior to FloodNet in message efficiency, especially in the topology with high connectivity. 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, in Figure 12(a), the message efficiency is nearly 75 percent in (4, 0), whereas it is less than 65 percent in (4, 1). This is consistent with what we have observed in section 4.3.

The last observation from figure 12(b) and (c) is that the higher the connectivity is, the worse the message efficiency is. This is because high connectivity can cause more messages in the first stage, so the chance of message collision is higher for (3, *) in Top3.

5. Conclusion and Future Work

In this paper, we build PercolationNET, a multi-tree sub-overlay, 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. In the first stage, a query message is propagated in the original overlay so that the

message can spread to all trees of the sub-overlay with a limited TTL value. Then the message is broadcast by flooding along the sub-overlay in the second stage, which ensures that the message can span the entire network with several additional hops. The experiments show that the proposed sub-overlay structure is more efficient than the existing FloodNet scheme, including coverage scope 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*, *secondTTL*) and the topology connectivity 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 with grants 60873225, 60773191, 70771043, the National High Technology Research and Development Program of China (863 Program) with grant 2007AA01Z403, and U.S. National Science Foundation grants CNS-0834592 and CNS-0832109.

Notes

1. Email: zuocuihua@smail.hust.edu.cn
2. Email: hshen@uark.edu
3. Email: kmwen@hust.edu.cn
4. Email: xwgu@hust.edu.cn

References

[1] Gnutella Network Size, http://www.limewire.com/index.jsp/size, 2007.
[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, pp. 149-160.
[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content-Addressable Network”, Proc. ACM SIGCOMM, San Diego, California, USA, 2001, pp.161-172.
[5] H. Jiang and S. Jin, “Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-Peer Networks”, Proc. 13th IEEE International Conference on Network Protocols (ICNP’05), 2005, pp. 122-131.
[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, pp. 5-14.
[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] J. Luo, S. Zhou, C. Wu, Y. Deng, and X. Yang, “Adaptive Flooding Routing Algorithm in Unstructured P2P”, Proc. 4th IEEE Intl’ Conf. Communications, Circuits and Systems, 2006, pp. 1557-1561.

[9] I. A. Klampanos, V. Poznanski, “Retrieval Efficiency in Peer-to-Peer Networks with Replication Restrictions”, Proc. 16th International Workshop on Database and Expert Systems Applications (DEXA’05), 2005, pp. 758-763.

[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), Cambridge, MA, USA, 2007, pp. 311-317.

[11] E. Cohen, and S. Shenker, “Replication Strategies in Unstructured Peer-to-Peer Networks”, Proc. ACM SIGCOMM, Pittsburgh, Pennsylvania, USA, 2002, pp. 177-190.

[12] T. Yamada, K. Aihara, A. Takasu, and J. 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), 2004, pp. 219-226.

[14] H. Guclu, and M. 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, pp.32.

[15] M. Srivatsa, B. Gedik, L. Liu, “Large Scaling Unstructured Peer-to-Peer Networks with Heterogeneity-Aware Topology and Routing”, Proc. IEEE Transactions on Parallel and Distributed Systems, 2006, 17(11): 1277-1293.

[16] Y. Liu, L. Xiao, X. Liu, L.M. Ni, etc, “Location Awareness in Unstructured Peer-to-Peer Systems,” IEEE Transactions on Parallel and Distributed Systems, 2005, 16(2): 163-174.

[17] Y. Liu, L. Xiao, and L.M. Ni, “Building a Scalable Bipartite P2P Overlay Network,” IEEE Transactions on Parallel and Distributed Systems, 2007, 18(9): 1296-1306.

[18] S. Jiang, L. Guo, X. Zhang, et al, “LightFlood: Minimizing Redundant Messages and Maximizing the Scope of Peer-to-Peer Search”, IEEE Transactions on Parallel and Distributed Systems, 2008, 19(5): 601-614.

[19] Q. Lv, S. Ratnasamy, and S. Shenker, “Can heterogeneity make gnutella scalable”, Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), LNCS 2429, Cambridge, MA, USA, 2002, pp. 94-103.

[20] PeerSim: A peer-to-peer simulator, http://peersim.sourceforge.net

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

[22] A. Medina, A. Lakhina, I. Matta, et al, ”BRITE: Universal Topology Generation from a User’s Perspective”, Technical Report BUCS-TR-2001-003, Boston University, April 2001.