# Self-learning Routing in Unstructured P2P Network

Haitao Chen, Zhenghu Gong, and Zunguo Huang

School of Computer Science, National University of
Defense Technology, 410073 Changsha, Hunan, China

nchrist@163.com

## Abstract

Since the appearance of Napster in 1999, peer-to-peer networks which don't distinguish clients with servers, have become one of the fastest growing and most popular Internet applications. Content location is a key function, but it presents a very difficult and challenging problem for large-scale peer-to-peer systems. A number of different methods are currently in use. However, new mechanism which gains high success rate with low search scope and retains the simplicity and robustness of Gnutella-like systems is still under research. In this paper, we introduce a new self-learning algorithm- SLPS. SLPS learns interest similarity of nodes from history search results, and computes friend relations which can be used to locate content effectively. Simulation results show that, compared to the flood-based algorithm, SLPS improves query efficiency by up to ten times without a significant increases in load.

**Keyword**: peer-to-peer, file sharing, search, learning, interest

## I. Introduction

Today computing and storage capacity of ordinary PCs are much higher powerful than those of former mainframes. The advances of fiber transmission and high-speed network switching technology provide possibilities for sufficiently utilizing the re-sources of ordinary PC. As a revolutionary technology, P2P technology devotes to reasonably and sufficiently organize and utilize Internet edge's huge decentralized resources of computing, storage and information. P2P networks have become, in a short period of time, one of the fastest growing and most popular Internet applica-tions. The big challenge of constructing P2P application is how to implement effi-cient and distributed search in complex environments which contains decentralized and huge-amount users, uncontrollable nodes with unbalanced computing capacity and network connection.

This paper presents a self-learning semantic search method-SLPS. SLPS learns the interests of nodes by passive learning. It learns the file sharing relations from the search results and constructs friend relations. In SLPS, queries are routed to friend nodes in an un-structured pattern. If the searches in friend nodes fail, broadcast search will be executed.

The rest of this paper will be structured as follows. Section 2 reviews important related works. Section 3 presents a self-learning algorithm in unstructured peer-to-peer network. In section 4 we make sufficient simulation tests on performance of SLPS algorithm. The last section offers our conclusions and future work.

## II. Related Work

We can categorize the content location algorithms used in current P2P systems into four groups by overlay network architecture. The earliest design, used in Napster [1], is the most similar to the client/server model. It depends on a central server (or cluster of servers) which maintains an index with the location of all shared files available in the system. In this approach, content location is centralized, but the download of contents is decentralized.

The other three approaches are based on the distribution of the indices to the shared files among all participant peers: structured, unstructured and hybrid. These approaches differ as to whether the content location mechanism relies on any special structuring of the peers. Different architectures support different search capacities like fuzzy search in different degrees.

Structured overlays, based on the Distributed Hash Table (DHT) abstraction, [2] [3][4][5] have been proposed to address the scalability problem inherent to flooding. In these protocols, each content is identified by a key and nodes organize themselves into a well-defined graph that maps each key to a responsible node. These constraints provide efficient support for exact-match queries. Some researches devote to realize complex search in DHT network. In [6], keywords are hashed into DHT space and nodes are responsible for neighboring keywords. In PGroup, nodes with the same classes of contents are connected to each other by building semantic peer-to-peer networks in CAN network [7]. Arpeggio is a peer-to-peer file-sharing net-work, based on the Chord lookup primitive, in which queries for data whose Meta-data matches a certain criterion are performed efficiently by using a distributed key-word-set index, augmented with index-side filtering [8]. Indeed, the obvious approaches to distributed full-text document search scale poorly [9]. It is not clear how to perform searches without sacrificing scalability or query completeness, and DHTs incur larger overhead than unstructured architectures in the presence of peer failure or disconnection and inherently, cannot efficiently support partial-match queries [10].

Unstructured content location is used in systems such as Gnutella [11]. Gnutella relies on flooding of messages. Peers organize themselves into a random overlay. In order to find content, a peer sends a query to all its neighbors on the overlay, which, in turn, forward the query to all of their neighbors and so on, until the query time-to-live has expired. While this solution is simple and robust, it does not scale well. Several works tried to solve the scalability problems inherent to Gnutella networks, but none thoroughly solve the problem. The authors of [12] proposed a content location algorithm for such networks based on multiple random walks that resolves queries almost as quickly as flooding while considerably reducing the network traffic. BFSFlood [13] is a variation of the flooding scheme, with peers randomly choosing only a ratio of their neighbors to forward the query to. Iterative Deepening [14] in-creases the TTL of flood only when search with smaller TTL fails.

Hybrid architecture may be a promising direction. It tries to combine the advantages of both unstructured and structured P2P networks. Castro et al proposed Structella, a hybrid of Gnutella built on top of Pastry [15]. Another design used structured search for rare items and unstructured search for massively replicated items [16].

The effective pruning technique is the key problem to complex query, which can greatly reduce the number of involved files and nodes during one search probing. Recent years many researches concentrate on pruning technique by files or by key-words. SETS divides nodes into different themes and searches spread in theme scope [17]. Mercuzy supports for multi-attributes search. It divides attribute space according to scalar attributes [18]. NeuroGrid learns keywords distribution from history search results [19]. It implements the division of keywords space by learning the relations between nodes and keywords. Alpine depends on users' definition to maintain group relation and nodes in same group are all connected with each other [20]. Recently, GS proposed a new content location algorithm that explores common interests among peers [21]. Peers that share similar interests create shortcuts to one another and use them to locate content. When shortcuts fail, peers resort to using the underlying Gnutella protocol. In conclusion, further research is needed on automatic grouping, making full use of file relations and user relations to realize efficient pruning.

Like the shortcut-based algorithm described in GS, our new algorithms also explore the common interests among peers to improve the scalability of the content location mechanisms in Gnutella systems. However, they differ from the previous approach in how common-interests among peers are assessed. By exploring locality of interest in a different way, we expect to reach more significant improvements in the Gnutella performance.

## III. Self-Learning Query Routing

### A. *Design Idea*

Partitioning files or users to limit search scope and improve search efficiency is a common method used in distributed files search researches. Most traditional file grouping methods depend on users to specify attributes or description of a group. But different users may have different descriptions on the same group or have different classification options. This disagreement often leads to inconvenient centralized or predefined group management. SLPS algorithm presents a new user grouping method which requires no users' description and avoids inconsistency of group descriptions. With good grouping method, search-oriented friend relations can be established automatically based on interests similarity between two users. SLPS can greatly optimizes the traditional broadcast algorithm through friend routing and is expected to gain high search success rate with low spend.

As sharing files of each user can stand for his interests, we guess the number of same sharing files between users can be used to compute the similarity of users' interests. The design of SLPS is bases on this assumption. SLPS has to solve two problems. How to collect sharing information in distributed environment? And how to route query based on users' interests?

### B. *Passive Learning Algorithm*

The basic process of learning friend relations includes two self-governed algorithms executing at different opportunities. First algorithm is devoted to learning file sharing relations from history search results. When any node issue search request and get results from some nodes, it will send the search results to those who return successful results. By this process, the nodes can get to know who is sharing the same files with them. The second algorithm is devoted to selecting friend nodes. Each node regards nodes sharing same files with it as friend candidates. Then at regular intervals, nodes ranks friend candidates from high to low according to the

number of sharing same files and chooses the first top k nodes as its friend nodes. As time goes on, each node can learn those friend nodes who share same interests with it.

### C. *Query Routing Based on Friend Relations*

Query Routing based on friend relations includes two steps. First it makes use of friend relations to search. The levels of friend relation can vary from 1 to n. In most cases two levels of friend relation are enough. Most requests can receive responses in first step. If the friend search fails, it adopts flooding search as supplement. The detailed two-levels query routing algorithm is shown as follows.

N is the set of all nodes. $N_i$ denotes $i$ th node. $N_i$.Friend denotes friend set of $N_i$.

```
Program SLPS (int i, int j, string r, int step)
{
  //node i receives the searching request r that node j
  //sends. Step denotes the phase of search process.
  If (step=1){
    //the value of step of original node is one.
    result:= LocalSearch(r);
    // Executing search on local disk including cache.
    if (result≠ nil)  return result ;
    //Process finishes if local search successes.
    for l:=1 to k  SendRequest (r, N_j.Friend[l],2);
    // Search request is sent to all its friend nodes.
    result:=WaitForResponse();
    // Waiting for response from friend nodes.
    if(result=nil )  result= FloodSearch(r);
    //if the search on friend nodes fails, it switches //to flooding search
    return result;
  }
  if(step=2){
    //If the nodes getting requests are the friend nodes
     // of original node, then the value of step is 2.
    result:= LocalSearch(r);
     // Executing search on local disk.
    if (result≠ nil) then return( N_j ,result);
    // Return results and finish search process if local
    // search successes.
    for l:=1 to k SendRequest(r, N_j.Friend[l],3);
    // Search Request is sent to all its friend nodes.
    // This is the use of second friend relations.
    result:=WaitForResponse();
    // Waiting for response of friend nodes.
    return( N_j ,result); //Return search result to node j
  }
  if(step=3){
    // if the node receiving the request is the friend
    // nodes of friend nodes of original node,
    // then the value of step is three.
```

```
        result:= LocalSearch(r).
        // Executing search on local disk
        return( N_j ,result); // Return search result.
    }// end of step3
}// end of program
```

## Ⅳ. Performance Evaluation

Integrating the former researches [22][23], we evaluate the performance of P2P query routing algorithms using the following four metrics.

1) Search Success Rate denotes the proportion of success rate of search.

$$Search\ Success\ Rate = \frac{Number\ of\ \ Successful\ Search}{Number\ of\ \ All\ Search}$$

2) Query Scope is the number of peers in the system involved in query processing for each query. A smaller query scope increases system scalability.

$$Query\ Scope = \sum_{Search\ Step} Number\ of\ \ Nodes\ involved\ in\ each\ step$$

3) Query Efficiency denotes the efficiency of search algorithm which is a new metrics introduced by this paper.

$$Query\ Efficiency = \frac{Query\ Success\ Rate}{Query\ Scope}$$

4) Query Hop denotes average search hops which stands for the average search delay.

$$Query\ Hop = \frac{\sum_{Search\ Number} Search\ Hops\ in\ each\ search}{Search\ Number}$$

In this section, we use both simulator-based and trace-based simulation for our performance evaluation. Artificial data later named Simdata are generated by a generic P2P simulator developed by Tokyo University of Japan which has configurable parameters [19]. Simdata contains two thousand nodes, four thousand files, and four thousand keywords. Query nodes and query contents are generated randomly. Because web traces and P2P access data are similar at some ways, trace-based simulation is widely used in P2P search domain [21] [22]. The trace-based test methodology used in this paper is the same as in [21]. For each web trace, we assume that all Web clients participate in a Web content file-sharing system. Query workloads are generated in the following way: if peer P1 downloads file A (or URL A) at time t0, peer P1 issues a query for file A at time t0. We model the query string as the full URL, A, and perform exact matching of the query string to filenames. We assume that P1's intention is to search for file A, and all hosts with file A will respond to the query. To preserve locality, we place the first copy of content at the peer who makes the first request for it. Subsequent copies of content are placed based on accesses. That is, any peer who downloaded a file at time t0 will share it to all other nodes after time t0.

Here we adopt three groups of wildly-used web traces with different scale to test SLPS algorithm. Boston [24] is the web traces of Boston University which contains 558261 records, 538 nodes and 9431 files. Berkeley [24] is the web traces of Berkeley University which contains 1703836 records,

5222 nodes and 116642 files. Boeing [24] is the web traces of Boeing Corporation which contains 4421526 records, 28895 nodes and 254240 files.

In trace-based simulation, each node has at most five friend nodes. Note that peers who have just joined the system do not have any friends on their friend lists, and have no choice but to flood to locate contents. We start counting the search success rate after the search success rate is stable. Figure 1 depicts the average search success rates of SLPS and BFSFLood [13] for different test data. The vertical axis is the search success rate, and the horizontal axis is the search hop.

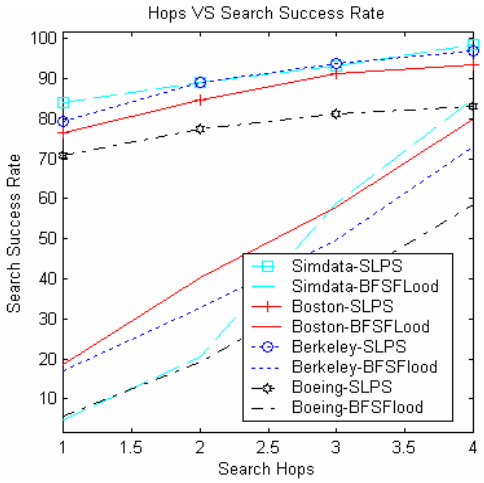


Figure 1 Comparion of Search Success Rate

Table 1 Comparion of Query Scope

| | SimData | BostonData | BerkeleyData | BoeingData |
|---|---|---|---|---|
| BFSFLood | 305 | 388 | 664 | 691 |
| SLPS | 28 | 41 | 37 | 128 |
| BFSFlood/SLPS | 11 | 9.5 | 18 | 5.4 |

Table 2 Comparion of Query Efficiency

| | SimData | BostonData | BerkeleyData | BoeingData |
|---|---|---|---|---|
| BFSFLood | 0.279 | 0.206 | 0.11 | 0.084 |
| SLPS | 3.52 | 2.28 | 2.62 | 0.65 |
| SLPS / BFSFlood | 12.6 | 11.1 | 23.8 | 7.7 |

Table 3 Comparison of Query Hop

| | SimData | BostonData | BerkeleyData | BoeingData |
|---|---|---|---|---|
| BFSFLood | 4 | 4 | 4 | 4 |
| SLPS | 1.6 | 1.8 | 1.5 | 2.5 |
| BFSFlood/SLPS | 2.5 | 2.2 | 2.7 | 1.6 |

The average search success rate of SLPS at the first hop is as high as 70%-85% for four different groups of data. Yet the average search success rate of BFSFlood at the first hop is as low as 4%-18% for four different groups of data. At the end of search, the search success rate of SLPS algorithm is still higher than BFSFlood. The search success rate of SLPS increases fast at the first two hops but increases slowly at the last two hops. So SLPS is efficient to locate popular contents and depends on expensive flooding to locate rare contents. BFSFlood method depends on query scope to improve

success rate, so its success rate is linearly increasing. It is clear that friend relations can gain high success rate with very low search scope and latency.

Table 1 lists the query scope of SLPS and BFSFlood algorithm. The query scope of BFSFlood is 5-18 times higher than that of SLPS. If we increase the friend node of each node for Boeing traces, also the reduction of search scope will be about a factor of 10. Table 2 lists the query efficiency of SLPS and BFSFlood. The query efficiency of SLPS is 7.7-23.8 times higher than that of BFSFlood. Table 3 lists the average query hop of SLPS and BFSFlood algorithm. The query hop of BFSFlood is 1.6-2.7 times higher than that of SLPS. With the help of friend relations, SLPS can greatly reduce query expense and query delay. We can see that the scalability of SLPS is much better than BFSFlood. The improvements are achieved by using friend relations before flooding so that only a small number of peers are exposed to any one query.

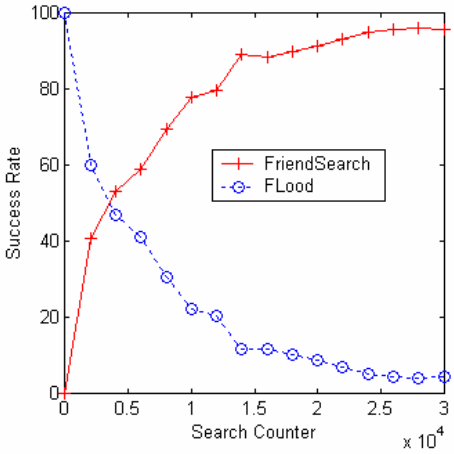


Figure 2 Constringency Test of Berkeley Traces

Table 4 Relations between constringency speed with node number

| | SimData | BostonData | BerkeleyData | BoeingData |
|---|---|---|---|---|
| Node Number | 4000 | 538 | 5222 | 28895 |
| Constringency Speed (query counter) | 20000 | 2000 | 28000 | 42000 |
| Constringency Speed /Node Number | 5 | 3.7 | 5.4 | 1.5 |

Table 5 FriendSearch VS Flood

| | SimData | BostonData | BerkeleyData | BoeingData |
|---|---|---|---|---|
| SLPS | 94% | 92.45% | 96.1% | 80.3% |

Next, we research on the learning process of SLPS. Figure 2 depicts the constringency process of Berkeley traces. As time goes on, the fraction of query finished by friend search increase while the fraction of query finished by flood decrease. The speed of constringency is as high as in table 4. The learning process will be stable only after the average search number of each node exceeding 2-6 times. This means that the learning algorithm is efficient and nodes can learn its friend nodes in a short period of time after joining P2P network. Table 5 lists the fraction of queries finished by friend nodes after the learning process converged. The success rate is as high as 80%-96%. This means the learning process is very successful.

## Ⅴ. Conclusion

This paper researches distributed file search in complex environment and presents a new method of searching sharing files- SLPS. SLPS learns friend relations between nodes based on search results. The queries are forwarded to friend nodes. Only failed requests will continue to broadcast.

Simulation tests show that SLPS algorithm is efficient and stable. SLPS brings the performance to within an order of magnitude of improvement compared with classical algorithms such as BFSFlood and so on. It can relieve the bandwidth consumption black hole of existing P2P file sharing systems and can be used in file sharing, con-struction of scale web caches, resource discovery of grid and so on. And SLPS is incentive-compatible, because only active nodes learn the search results. The more a node services others in P2P network, the higher success rate and the lower latency it gains. The future researches include: 1) more effective algorithm of mining friend relations. For example it can take file rarity into count. 2) Adding semantic description on friend relations to improve the expansibility of system.A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

## Acknowledgements

This research is supported by the National Grand Fundamental Research 973 Program of China under Grant No.2003CB314802, also the 863 National High-Tech Research and Development Plan of China under Grant No.2003AA142080.

## References

[1] The napster.2004. home page. http://www.napster.com.
[2] Ratnasamy, S, Francis, P., Handley, M., Karp, R., and Shenker, S: A scalable content-addressable network. In Proc of ACM SIGCOMM pp161－172 (2001)
[3] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan: Chord: A scalable peer-to-peer lookup service for internet applications. In Proc of ACM SIGCOMM (2001)
[4] A. Rowstron and P. Druschel: Pastry:Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proc of International Conference on Distributed Systems Platforms (Middleware), Nov. 2001.
[5] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph: Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.
[6] Zhou J, Lu HM, Li YD: Using small-world to devise routing algorithm for unstructured peer-to-peer system. Journal of Software, 2004, 15(6):915~923.
[7] Song Jian-Tao, SHA Chao-Feng, YANG Zhi-Ying, and ZHU Hong: Study on Construction and Searching of Semantic Peer-to-Peer Networks. Journal of Computer Research and Development. 2004. Vol.41(4):645~652.
[8] Austin T. Clements, Dan R. K. Ports, David R. Karger: Arpeggio: Metadata Searching and Content Sharing with Chord. In proc of iptps(2005)
[9] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. Karger, and R. Morris: On the feasibility of peerto-peer web indexing and search. In Proc of IPTPS (2003)
[10] E. Cohen, A. Fiat, and H. Kaplan: Associative search in peer to peer networks: Harnessing latent semantics. Infocom (2003)
[11] The gnutella. 2004. home page. http://www.gnutella.wego.com.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker: Search and replication in unstructured peer-to-peer networks. In Proc of the 16th international conference on Supercomputing, pp 84-95. ACM Press (2002).
[13] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In proc of CIKM (2002)
[14] B. Yang and H. Garcia-Molina: Improving Search in Peer-to-Peer Networks. In proc of ICDCS (2002)
[15] Castro,M., M. Costa and A. Rowstron: Should we build Gnutella on a structured overlay? ACM SIGCOMM Computer Communication Review 34(1):131-136.2004.
[16] Boon Thau Loo,Ryan Huebsch,Ion Stoica, Joseph M. Hellerstein: The Case for a Hybrid P2P Search Infrastructure. In Proc of iptps (2004)
[17] Mayank Bawa, Gurmeet Singh Manku, Prabhakar Raghavan: SETS: Search Enhanced by Topic Segmentation. ACM SIGIR (2003)
[18] Ashwin R. Bharambe, Mukesh Agrawal, Srinivasan Seshan: Mercury: Supporting Scalable Multi-Attribute Range Queries. ACM SIGCOMM (2004)
[19] Joseph, S.R.H: NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks. In proc of International Workshop on Peer-to-Peer Computing (2002)
[20] Alpine. 2004. http://www.cubicmetercrystal.com/alpine/
[21] Kunwadee Sripanidkulchai, Bruce Maggs, Hui Zhang: Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems, In proc of infocom (2003)
[22] Yang B, Garcia-Molina H: Improving search in peer-to-peer networks. In Proc. of Distributed Computing Systems. IEEE Computer Society (2002)
[23] Balakrishnan H, Kaashoek MF, Karger D, Morris R, Stoica I: Looking up data in p2p systems. Communications of the ACM, 2003, 46(2):43~48
[24] Webtraces. http://www.web-caching.com/traces-logs.html.

Haitao Chen, PhD degree candidate in national university of defense technology. His research interests include: Peer-to-Peer Computing, Network and Information Security.

Zhenghu Gong, professor in national university of defense technology. His research interests include: Computer Network and Communication.

Zunguo Huang, associate professor in national university of defense technology. His current focus is information security and network survivability.