A Novel Search Technique in Peer-to-Peer Networks

Sabbir Ahmed[†] and Kyoji Kawagoe[‡]

[†]Graduate School of Science and Engineering

[‡]College of Information Science and Engineering

Ritsumeikan University

Nojihigashi 1-1-1 Kusatsu, 525-8577 Shiga-ken, Japan

E-mail: [†]sabbir_001@yahoo.com, [‡]kawagoe@is.ritsumei.ac.jp

Abstract Sufficient uptime is the main concern in a distributed system where there is no centralized organization and control. Therefore searching in P2P file sharing systems is still a big problem due to the high churn rate of nodes. To find a particular piece of data within the network P2P systems explicitly or implicitly provide a lookup mechanism, which largely depends on the availability of nodes. In this paper we propose a stable node based ranking technique by detecting reliable nodes with adequate uptime by nature to ensure efficient searching and un-interruption during the data receiving. We have simulated an unstructured file-sharing environment to evaluate our proposed technique.

Keyword Peer-to-Peer systems, Information retrieval, Searching, Node ranking, Overlay network

1. Introduction

As a distributed system in Peer-to-Peer (P2P) systems nodes play equal role in a decentralized mode to exchange information and services directly with each other. In recent years, with the enormous increase of Internet use these systems have opened the way of storing large volume of information in a cost-effective way.

Unlike a client-server system, these systems have no dedicated server to serve others. Millions of computers usually called peers or nodes, all over the world build this network, which creates huge Internet traffic during communications. These peers are very unstable in nature and they only care about their own purpose. Our approach is to collect the stability value of each node to rank them and route the query only to those nodes based on the rank. From our observation we have seen that the node, which has good stability record, has more resources than the nodes those are relatively unstable.

The volume of a P2P systems increase day by day, therefore ranking of nodes is a basic approach to filter these millions of nodes and to pick up the better quality nodes to ensure scalability of the system. The state of the art systems filter these nodes considering different attributes, which theoretically may be very smart approach but practically are not very efficient. Some times considering a lot of attributes of a single node creates huge calculation overheads.

Nodes join or leave the network frequently hence these networks are dynamic in nature. There are two major phases should be taken into account in this unstable situation; first the searching phase and second the data acquisition phase. In this paper we propose a stable node based ranking technique considering a unique parameter called the *Stability Ratio* of nodes, which will help to construct a stable overlay for efficient searching and data retrieval functions. For evaluation and testing of this method we have applied this method on Gnutella, the largest P2P architecture in operation.

2. Related Works

Basically there are three types of P2P systems; these are unstructured systems, loosely structured systems [16][17] and highly structured systems. Among highly structured P2P networks such as Chord [9], Tapestry [11] and Pastry [10] implement a distributed hash table (DHT) where the data placement is specified and they use

identifier based searching. Although, these systems are more precise but the implementation and maintenance are comparatively difficult and cost intensive. In an unstructured P2P systems, no rules exist that strictly defines where data are stored and which nodes are neighbors of each other. For easy maintainability and inexpensiveness these systems are widely used and popular.

Our searching technique basically solves the existing searching problems in the unstructured file sharing systems. To find a specific data item, early work such as the Gnutella [8] uses flooding, which is inefficient and creates a lot of network traffic. Many alternative schemes have been proposed to address the problem of the original flooding.

In *k*-walker random walk [3] algorithm query messages are called walker and this message is forwarded node to node. Here randomly selected *K* copies of query messages are deployed and they take their own random walk. Each walker periodically checks the original requester, if query is satisfied then terminate searching or keep walking up to TTL limit. The value of K is between 16 to 64 gives good results. In two-level *k*-walker random walk [12] method query node send k_1 random walkers with TTL being l_1, when TTL l_1 expires each walker forges k_2 random walkers with TTL being l_2. Popularity-biased random walk [6] system probes object with probability proportional to the square root of its query popularity.

The Adaptive probabilistic search [5] is based on the k-walk and probabilistic forwarding. The querying node simultaneously sends *k* walkers to the best neighbor that has the highest probability value. The probability values are calculated based on the results of the previous query hits. Iterative deepening [2] method uses BFS with the maximum depth limit *D* and there is a system wide policy that defines the iteration method.

In local indices based search [2] each node keeps an index of metadata of all neighbor nodes within *r* hops where *r* is the system-wide specified variable. Unlike Gnutella all nodes do not process the query, a system-wide policy *P* determines in which depth nodes will process a query. In this way, processing the query at few nodes can search the data of many nodes. Routing indices based search [7] the nodes store information of neighbors about the content of the documents instead of the filename or file identifiers. The objective is to select the best neighbors to forward queries.

In dominating set based search [1] approach instead of every node routing indices are stored in a selected set of nodes that make a Connected Dominating Set (CDS). All nodes other than CDS can be reached to CDS within one-hop. Searching is done through a random walk on the dominating nodes. There is a mechanism to select the dominating nodes those create the CDS overlay. In Directed BFS [2] the querying node forward the query to a subset of neighbors, which are selected based on some heuristics, while from next hop nodes still broadcast that as usual.

Intelligent search [4] is a similar approach of directed BFS. This technique consists of four components: a search mechanism, a profile mechanism, a peer ranking mechanism, and a query similarity function. When the querying node initiates a query, it evaluates the past performance of all its neighbors and forwards the query only to a subset of its neighbors that have answered similar queries before and therefore that most likely will answer the current query. In modified random BFS [4] technique, randomly selects a subset of its peers to propagate the search request. The fraction of peers that are selected is a parameter to the mechanism.

The very recent paper on this searching method [15] is slightly different from these previous works, where instead of measuring relative capability of the peers it focuses on how to find the best combination of metadata management technique and query result ranking strategy from the perspective of clients. Another protocol named TBRPF Neighbor Discovery (TND) [14] designed for mobile ad-hoc networks uses “differential” HELLO message that report the changes in the status of neighbor links. Thus topological changes in a dynamic network are always a problem for information retrieval (IR) processes.

Although our approach is based on node ranking technique but it exploits a completely new parameter, thus this technique is totally different from previous probabilistic searching technique in the file sharing P2P systems.

3. Problem Analysis

Efficient searching techniques consider two major attributes, the cost and the quality of results. The cost includes average cumulative bandwidth costs, which is required to propagate a query through the network. Another one is average cumulative processing cost, which is required to propagated a query through the network to process and forward them (i.e., cycles). Our intension is

to reduce the cost of query, but it must be ensured that the techniques do not degrade the user experience.

The quality of results refers to the number of results that satisfy a query. The quality also means precision and reliable source of data. But in P2P environment the quality often refers to the host's performance that return the result. Since we propose stable node based ranking, which ensures good quality results from reliable sources.

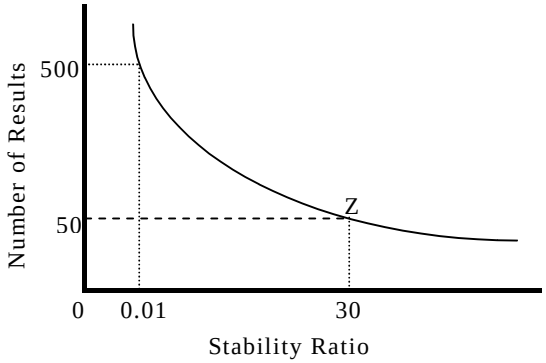

Fig. 1. Relationship between number of stable nodes & query results, which shows the node stability, filters the quality results.

The P2P network systems have two major problems. Firstly it creates huge network traffic during searching and secondly due to nodes instability it causes interruption at time of data retrieval. Only successful searching does not ensure successful retrieval of data if those sources are not stable.

Since in peer-to-peer system there is neither any dedicated server nor any client is responsible to provide services to others, therefore in such a system it is more important to consider the node stability than to consider the location. The main motivation of our research is unlike others whose aim is to find nearest neighbors or just randomly filtering the nodes. Our aim is to find the stable nodes among the millions of nodes.

The figure 1 shows that if the numbers of nodes are reduced for searching on the basis of stability ratio then the number of query match is reduced proportionately. Thus there is a tradeoff between number of searching nodes and quantity of results return. It is a common problem of filtering nodes however we are concern about the quality of results instead of the quantity. Hence quality results are always more appreciated than the huge inferior ambiguous results.

4. Proposed Method

In a dynamic network where the node churn rate is very high, stable node based searching is more realistic technique. From our observation we have also seen that stable nodes contains more data than unstable nodes. Our technique is to collect the *Stability Ratio* of participants to build a stable overlay network. Each node will calculate their stability value at their respective node and will broadcast the value as they broadcast their IP, number of file shared, number of kilobyte shared, etc.

4.1 Definition of Stable Node

The ratio that shows the duration of active mode of a node in percentage is called the *Stability Ratio* (*SR*). Here stability refers to the time. The node, which is available in the network, is called the active node. The application needs to find out which nodes are more active in the network in respect of time. To calculate this uptime nodes need to maintain a log file locally, which will log the data of each starting and ending time of node activation. It will also add the timestamp with each record. From these log records of last few days (system-wide specified variable) each node will calculate its own *SR* using the following equation

$$SR = C\sum_{i=1}^{n} d_i \ .$$

Here d_i is the duration of a node in active mode at time t_i, t is time interval and n is the system-wide specified day count value. C is a constant; the value will depend on n. If n is 30, then each node in that network needs to preserve last 30 days login and logout records. From these previous records nodes can calculate d_i at each time. If we assume the time interval in hour and guess 30 random value of d, using the previous equation the *SR* calculation will be as follows

$$SR = \left(\frac{100}{30 \times 24}\right)\sum(5 + 6 + 3.7 + 10 + 19 + ...d_{30})$$

For example if *SR* = 30%, which defines the stability property of this node. A node may be ranked as stable if its *SR* is over the threshold. The threshold also will be system-wide specified or by user specified depends on the

application program.

4.2 Extension of Gnutella Protocol

For evaluating and testing we have considered the Gnutella [8] environment, a decentralized P2P system. It allows the participants to share resources from their system for others to see and get, and locate resources shared by others on the network. Resources can be anything: mappings to other resources, cryptographic Keys, file of any type, meta-information on key-able resources, etc. The Gnutella protocol defines the way of communicate over the network. It consists of a set of messages used for communicating data between nodes and a set of rules governing the inter-node exchange of messages [18].

In this protocol during communication some sequential structured messages or descriptors are employed. Messages are the entity in which information is wrapped and transmitted over the network.

- **Step1.** A node *n* sends a *Ping* message to its neighbor nodes
- **Step2.** Each node that receives the ping message sends a reply, which is called *Ping Response or Pong* and then forwards the message to their respective neighbors.
- **Step3.** After receiving all pong messages *n* flood the *Query* to all the active neighbors.
- **Step4.** Node *n* receives *Query Hits* from active nodes that has match and sort them up.
- **Step5.** Then the user of node *n* selects a resource to retrieve the data using HTTP protocol

The steps show the Gnutella communication scenario. For flooding Gnutella uses a breadth-first traversal (BFS) with depth limit D, where D is the system-wide maximum time-to-live of a message in hops.

Table 1. Gnutella Message Header

Fields	Descriptor ID	Payload Descriptor	TTL	Hops	Payload Length
Byte offset	0...15	16	17	18	19...22

Table 2. Gnutella Ping (0x00) payload

Fields	Optional Ping Data
Byte offset	0...L-1

Table 3. Gnutella Ping Response/Pong (0x01) payload

Fields	Port	IP Address	Number of Files Shared	Number of KB Shared	Optional Pong Data
Byte offset	0...1	2...5	6...9	10...13	14...L-1

The message formats are shown in table1, table2 and table3. Each node that receives the *Ping* message, must reply the message. This *Ping Response* message contains port number, IP address of the node, number of file shared, number of kilobytes shared by the node etc. This protocol also allows for extensions inside many messages. The Gnutella Generic Extension Protocol (GGEP) allows arbitrary extensions in Gnutella messages [18]. All nodes pass on GGEP extension blocks inside Gnutella messages and they include a new header in the Gnutella connection handshake indicating this support.

Table3 shows there is an optional field of variable length in *Pong* message, which is reserved for extensions of the current version of the protocol, to give other information about the node. This optional field may contain the value of *SR* of that node in GGEP block. Thus Gnutella can easily inherit our proposed stability based ranking method. Every participant has to maintain a database locally to log its own login and logout time. How many previous records have to be preserved that must be system-wide specified. At the time of *Pong* message when the node calculates its others metadata information at the same time it will calculate its *SR* value. Then through the *Pong* message that node will broadcast this value using GGEP extension. Therefore the querying node will determine the query routing policy considering the collected *SR* values instead of just flooding the query to its entire neighbors.

4.3 Node Ranking in Gnutella

In different communication phases of Gnutella architecture the node ranking technique based on node stability value may significantly improve the system.

Join the Network. A Gnutella node first joins to the network by establishing a connection with another node currently on the network. There are some permanent Gnutella hosts called host-caches whose purpose is to provide a list of Gnutella hosts to any node connecting to them. Each node also keeps a list of few hundred hosts on

the disk. In both the cases node ranking may play an important roll for choosing the list of reliable nodes.

Pong Caches. An important feature of Gnutella 6.0 is to maintain pong caches, which helps to reduce network traffic. When a Ping message is received (TTL>1 and it was at least one second since another Ping was received on that connection), a node respond with a number of Pong (about 10) messages with the same message ID as the incoming ping. To keep the cache fresh, a ping (TTL=7, Hops=0) is sent over all connections at small interval (like every 3 seconds). The problem is how to select which of the stored pongs to send in response to an incoming ping. A good idea is to pick Pong with stable-ranking nodes.

Ultrapeer. The Ultrapeer system has a scheme to have a hierarchical Gnutella network by categorizing the nodes on the network as leaves and Ultrapeer. A leaf keeps only a small number of connections open, and that is to ultrapeers. An ultrapeer acts as a proxy to the Gnutella network for the leaves connected to it, which is very useful to scale the system. For electing a node as ultrapeer the previous record of uptime of that node is very significant. Thus ranking node on the basis of stability value is truly effective for leaf-ultrapeer architecture.

5. Evaluation

For our evaluation we have simulated the Gnutella environment, because Gnutella is one of the most popular P2P systems at present.

5.1 Simulation Background

We have observed several Gnutella applications about 3 months to gather information from the existing systems. The basic information is as follows

- How many files a user in average shares
- What type of files are shared
- Number of neighbors connect to a network each time
- Number of query matched in average
- How many Gnutella messages are received per time interval
- How many kilobytes of messages received per time interval
- How many queries pass through the node
- How many query hits pass through the node
- Number of incoming request received over time

These network statistics have been used to determine the characteristics of an existing Gnutella network participant. In our proposed technique the step 3 as shown in table3 will be modified. Depending on the extended *Pong* message the querying node will route the query to the subset of neighbors(s), which are ranked as stable node.

To evaluate the cost and quality of results of our proposed method we need the data of *Stability Ratio* from all active nodes that we mentioned earlier. Since at present no Gnutella client calculates its stability value, therefore we have designed a simulator. We have used Pentium 4 CPU 3.60 GHz computer with 2GB RAM and Windows XP (SP2) and develop the simulator in JAVA with MySQL. Before we analyze our findings lets take a look into some definitions mentioned bellow:

Cost. When a client sends a query the total cost of this system includes two types of cost, cumulative bandwidth costs and cumulative processing cost.

- **Cumulative bandwidth cost**: refers to the bandwidth required to send and receive query messages through the network. Since every node of a network using bandwidth to propagate a query through the network so we have to consider the average cumulative bandwidth.
- **Cumulative processing cost**: refers to the process-cycles involved in processing and forwarding a query. A query is propagated through a network hence these costs are also distributed over the network.

Satisfaction level. Many P2P systems display only the first Z results, where Z is some value specified by the user or by the systems, called the level of satisfaction [2]. A query is satisfied if Z or more results are returned. Hence, if Z=50, a query that return 500 hits perform no better than a query returning 50 hits. Sometimes 500 results confused the users to select the best one.

Quality of Results. In P2P environment the quality of results are measured by following considerations:

- Number of Results
- Precision of Results

- Level of Satisfaction
- Time to Satisfaction
- Reliable Source of Data

Table 4. Simulation Results

SR%	Total Nodes	Hits (TTL5)	Time (TTL5)	Hits (TTL7)	Time (TTL7)
0~100	3282	**85**	131	128	231
5~100	2242	82	1268	123	223
10~100	1939	81	126	121	221
15~100	1603	80	124	120	217
20~100	1305	79	124	118	214
25~100	1057	78	122	117	214
30~100	826	77	122	116	211
35~100	630	76	121	115	209
40~100	477	75	118	113	205
45~100	347	74	118	112	204
50~100	255	60	100	**89**	173
55~100	192	49	85	71	148
60~100	**144**	**39**	**74**	**57**	**129**
65~100	114	31	66	46	115
70~100	85	24	58	36	105
75~100	63	19	51	28	93
80~100	46	15	46	21	81
85~100	30	10	38	14	74
90~100	15	5	35	7	66
95~100	6	3	32	32	59

5.2 Result Analysis

Our practical experiences on several existing Gnutella systems were quite interesting. We have seen that the nodes stability is the major constraints for information retrieval especially for file searching. We have examined the basic information as mentioned above (Section 5.1) to understand a whole network. Based on these information we have simulated our system.

In the simulation we have considered total 3282 nodes distributed in 7 hops, where each node has 3 neighbors in one hop away. We randomly assigned *SR* to all nodes with a range from 0% to 100%. We propagated 100 queries among the nodes hop by hop. Table 4 shows the average outputs grouped by *SR* value.

The first row of data shows traditional flooding technique where all 3282 nodes have been traversed. For better comparison we have logged search hits twice up to hop 5 and up to hop 7 in each cases. From table 4 we can see that if we consider stability value over 60%, then we are getting 57 hits by searching 144 nodes which is 20 times less cost effective than searching in all 3282 nodes. Another important finding is by flooding up to hop5 we are getting 85 hits in first row, if we expect the same quantity of hits we may get that result by routing the query up to hop7 to the subset of nodes with SR>50%.

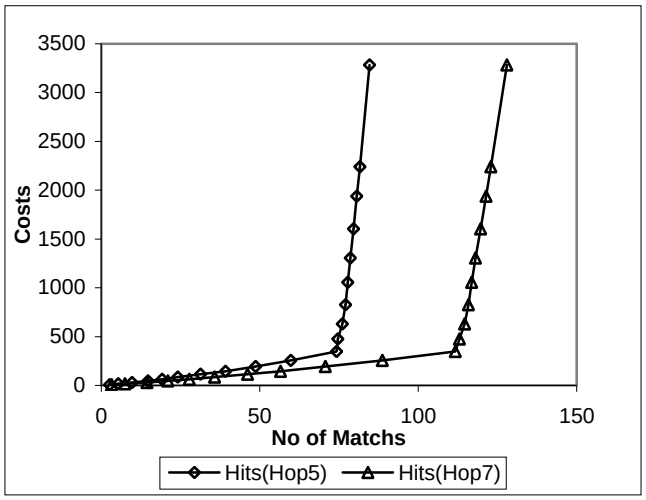

Fig. 2. Average cumulative bandwidth & processing cost increased with the number of nodes that process a query.

Figure 2 shows the cost and hit metrics. The cost includes average cumulative bandwidth cost and average cumulative processing costs (Section 5.1). For our experiment we assumed that the total cost is proportionate to the number of node searched. Costs are increased with the number of nodes that process a query. The graph shows that there is a tradeoff between costs and quantity of results. If the quantity of results is the main metric of quality of results (Section 5.1) then BFS flooding is the best approach. But it is efficient and overshoots the searching goal. We are concern more about quality than quantity of results and this technique filters the quality results by ranking the stable sources. We consider the reliable source of data (here the stable nodes) as an important issue to ensure the quality of results. This experiment ensures the adequate number of results using this technique. If precision could be guaranteed then we consider that Z=50 is more than enough.

Therefore the systems do not required 200 matches and we can filter only stable sources to get quality results. In figure 2 notices that node number representing the average costs involved in the query. The result shows after filtering 500 stable nodes among 3282 nodes return about 80 matches in TTL=5 and 115 matches in TTL=7, which are good enough to satisfy a Query.

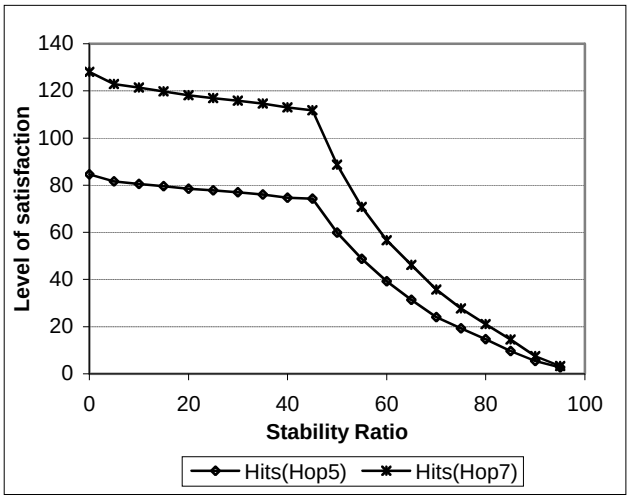

Fig. 3. The probability of satisfaction is ensured up to a certain level of *SR* consideration.

Most of the P2P applications consider a level of satisfaction *Z* (Section 5.1) to measure a query result and the systems only shows first *Z* [2] recalls to its users. Figure 3 shows the level of satisfaction and *SR* metrics. Here we can notice that ranking the stable nodes with stability value over 50 satisfy a query undoubtedly.

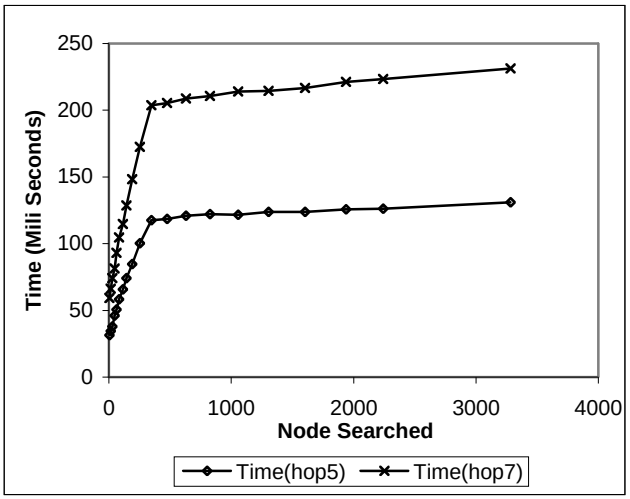

Fig. 4. Stable node filtering reduces overall time consumption for quality recalls.

In both the cases with TTL=5 and TTL=7 overlay with the stable ranked nodes returns more than 50 hits, which ensure the level of satisfaction. It is true that ranking nodes with *SR*=50 may not always applicable. The *SR* value for node ranking may vary system to system.

In our simulation we calculate time in milliseconds. Figure 4 shows average time consumption rate. Our objective is to reduce the number of nodes those process a query and to choose a few stable nodes that can return enough hits. If the number of subset of nodes that processes a query is selected intelligently then it will not only ensure quality results but also will reduce the recall time. Here notice that in TTL=5 searching 500 nodes take about 120 milliseconds where TTL=7 take about 2200 milliseconds. Time increases by the number of node search. Therefore if we can reduce the number of node that process a query then it reduces recall time to satisfaction as well.

6. Conclusions

Our research objective is to reveal a unique parameter to evaluate the nodes ranking status and to build a stable overlay for efficient information retrieval process. At the same time our aim is to design the technique in a less complicated way so that the existing systems can comprise it easily. Since we cannot force a peer to remain online until a file is completely downloaded, therefore it is important to choose a node, which is trustworthy by nature. Nowadays Internet accessibility becomes faster, bandwidth being cheaper and PCs processing power has improved a lot; therefore, in case of information retrieval stability of a node is more important than the location of the source. Our searching technique is faster and reduces network traffic, thus it improves the present searching techniques.

From our experience we have seen that each time searching brings a lot of ambiguous results unexpectedly. To alleviate the issue we are working with the efficient use of the cosign similarity a well-known distance function to enhance the precision of query results.

References

[1] C. Yang and J. Wu, "A dominating-set-based routing in peer-to-peer networks," *Proc. of the2nd International Workshop on Grid and Cooperative Computing Workshop (GCC'03)*, 2003.

[2] B. Yang, and H. Garcia-Molina, “Improving search in peer-to-peer networks” *Proc. of the22nd IEEE International Conference on Distributed Computing (IEEE ICDCS’02)*, 2002.

[3] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and replication in unstructured peer-to-peer networks”, *Proc. of the 16th ACM International Conference on Supercomputing (ACM ICS’02)*, 2002.

[4] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-yazti, “A local search mechanism for peer-to-peer networks”, *Proc. of the 11th ACM Conference on Information and Knowledge*

Management (ACM CIKM'02), 2002.

[5] D. Tsoumakos, and N. Roussopoulos, "Adaptive probabilistic search in peer-to-peer networks", Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003.

[6] Ming Zhong and Kai Shen (University of Rochester), "Popularity-Biased Random Walks for Peer-to-Peer Search under the Square-Root Principle", The 5th International workshop on Peer-to-Peer Systems, IPTPS 2006.

[7] A. Crespo, and H. Garcia-Molina, "Routing indices for peer-to-peer systems", *Proc. of the 22nd International Conference on Distributed Computing (IEEE ICDCS'02)*.

[8] http://www.gnutella.com

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", *Proc. of the 2001 ACM Annual Conference of the Special Interest Group on Data Communication (ACM SIGCOMM'01)*, 2001.

[10] A. Rowstron, and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", *Proc. of the 18th IFIP/ACM International Conference of Distributed Systems Platforms*, 2001.

[11] 11. Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment", *IEEE journal on selected areas in communications, vol. 22, no. 1, January* 2004

[12] I. Jawhar and J. Wu, "A Two-Level Random Walk Search Protocol for Peer-to-Peer Networks", *Proc. of the 8th World Multi-Conference on Systemic, Cybernetics and Informatics*, 2004.

[13] S. C. Rhea, and J. Kubiatowicz, "Probabilistic location and routing", *Proc. of the 21stAnnual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, 2002.

[14] http://rfc3684.x42.com

[15] W. G. Yee and O. Frieder. "On search in peer-to-peer file sharing systems", In *Proc. ACM SAC, Santa Fe, NM, Mar.* 2005.

[16] 16. G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world", *Proc. of 4th USENIX Symposium on Internet Technology and Systems (USITS'03)*, 2003

[17] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system", *Proc. of ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[18] http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html