Data Search and Replication in Unstructured Peer-to-Peer Networks

Zhijun Wang

Department of Computer Science and Engineering

The University of Texas at Arlington, Arlington, TX 76019, USA

Email: zwang@cse.uta.edu

Abstract

The Peer-to-Peer (P2P) architectures that are most prevalent in today's internet are decentralized and unstructured. The advantage for this type of networks is that they require no centralized directories and no process control over network topology or data placement. However, the flooding-based query strategy used in these networks does not scale. In this paper, we first introduce three basic types of peer-to-peer system. Among these systems, a decentralized an unstructured system, Gnutella, is discussed in detail. Three types of data search strategies: flooding, expending ring and k-walker are presented and their performances are evaluated in two different network topologies. Also three types of data replication schemes are given and evaluated. Some conclusions and discussions are drawn in the last.

I. INTRODUCTION

The peer-to-peer (P2P) systems [1]-[5], almost unheard of three years age, are now one of the most popular internet applications and a very significant source of internet traffic. The basic premise in such networks is that any one of set of replica nodes can provide the requested content, increasing the availability if interesting content without requiring the presence of any particular serving node.

Currently, there are three basic different architectures for P2P networks:

(1) **Centralized:**

Napster and other similar systems have a constantly-updated directory hosted at central locations. Nodes in the P2P network issue queries to the central directory server to find which other nodes hold the desired files. While Napster was extremely successful before its recent legal troubles, it is clear that such centralized approaches scale poorly and have points of failure (e.g., the system is out of work when the center directory node failure).

(2) **Decentralized Structured:**

In such a system, there is no central directory server, and so are decentralized, but have a significant amount of structure. By $structure$ means that the P2P network topology is tightly controlled and the files are placed not at random nodes but at specified locations that will make subsequent queries easier to satisfy. In loosely structured systems this placement of files is based hints, such as Freenet P2P network [4]. In highly structured systems both the P2P network topology and the placement of files are precisely determined; This tightly controlled structure enables the system to satisfy queries very efficiently. There is a growing literature on highly structured P2P systems which support a hash-table-like interface [5]-[7].

Such highly structured P2P designs are quite prevalent in the research literature, but almost completely invisible on the current network. Moreover, it is not clear how well such designs work with an extremely transient population of nodes, which seems to be a characteristic of the Napster community.

(3) **Decentralized Unstructured:**

These are systems in which there is neither a centralized directory nor any precise control over the network topology or file placement. Gnutella [4], [8]-[9] is an example of such design. The network is formed by nodes joining the network following some loose rules [10]. The resultant topology has certain properties, but the placement of files is not based on any knowledge of the topology. To find a file, a node queries its neighbors. The most typical query method is flooding, where the query is propagated to all neighbors within a certain radius. These unstructured designs are extremely resilient to nodes entering and leaving the system, just as an ad hoc system. However, the current search mechanisms are extremely unscalable, generating large loads on the network participants.

In this paper, we focus on the decentralized and unstructured P2P system, like Gnutella. Because this type of systems are actively used by a large community of internet users [11]. We first quantify the flooding search algorithm, then introduce a k-walker random search strategy, which greatly reduces the load generated by each query. We also present an active based data replication scheme. The performance evaluation are done and compared. At last some conclusions and discussions are given.

II. Modeling and Measurement Metrics

A. Network Topology

There are four network topologies for P2P network:

1) Power-Law Random Graph (PLRG): The node degrees follow a power-law distribution: if one ranks all nodes from the most connected, then the i'th most connected node has ω/i^{α} neighbors, where ω is a constant. Once the node degrees are chosen, the nodes are connected randomly [12]; Many real-life P2P networks have topologies that are power-law random graphs, the particular value is $\alpha = 0.8$.
2) Normal graph (Random): euqal random graph, generated by GT-ITM topology generator [11].
3) Gnutella graph : the Gnutella network topology, as obtained in October 2000. Its node degree roughly follows a two-segment power-law distribution.
4) Two-Dimensional Grid (Grid): a two-dimension grid.

B. Measurement Metrics

Performance issues in real P2P systems are extremely complicated. In addition to issue such as load on the network, load on network participants, and delays in getting positive answers, there are a host of other criteria such as success rate of research, the bandwidth of selected provider nodes, and fairness to both the requester and the provider. It is impossible for us to use all of these criteria in evaluating search and replication algorithms.

The following measurement metrics are discussed in the paper:

User Aspects::

1) $Pr(success)$: the probability of finding the queried object before the search terminates.
2) $number\ of\ hops$: delay in finding an object as measured in number of hops.

Load Aspects:

1) $Average\ number\ of\ messages\ per\ node$: overhead of an algorithm as measured in average number of search messages each node in P2P network has to process. The motivation for this metric is that in P2P systems, the most notable overhead tends to be the processing or message processing, is directly proportional to the number of messages that the node has to process.
2) $Number\ of\ nodes\ visited$: the number of P2P network participants that a query's search messages travel through.
3) $peak\ number\ of\ messages$: to identify hot spots in the network, i.e., the maximum messages a node need handle.

Aggregate Performance:

The performance of each query convoluted with the query probability. That is , for each object i, if the performance measure is $\alpha(i)$, then the aggregate is $\sum_i q_i * \alpha(i)$.

III. Search Algorithm

In this section, we describe three types of search strategies: flooding, expanding ring and k-walker.

A. Flooding

Flooding strategy is the simplest strategy in P2P networks. When a node has a query, it sends the query to all its neighbors, when the neighbor node gets the query message, they check if they have the queried

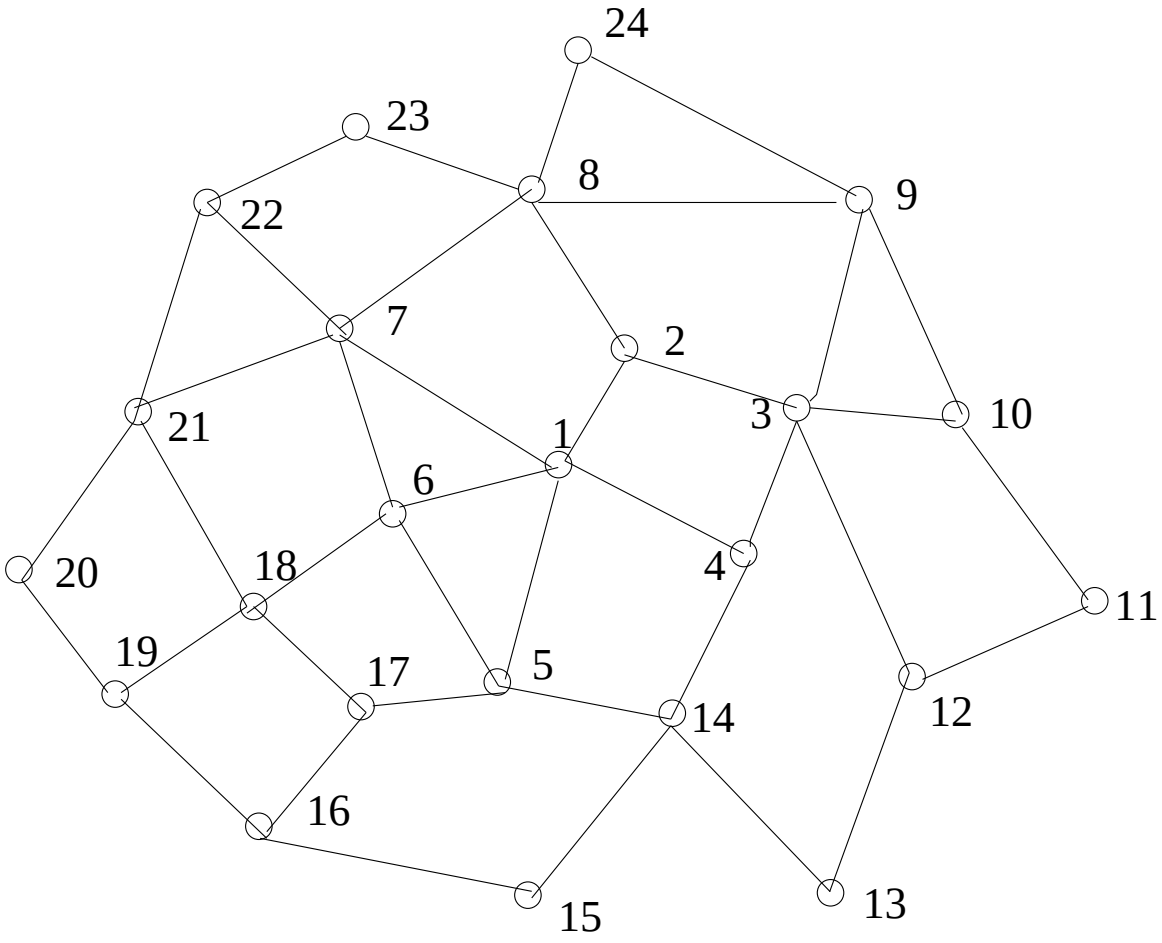

Fig. 1. Flooding Search Strategy

data object, if a node has the data object, it sends the data object back to the previous node. Otherwise, the node sends the query message to all its neighbor nodes except the node which sent the message. A TTL (Time-To-Live) is used to control the number of hops can be propagated.

Now we explain how the flooding search algorithm works. Figure 1 shows a P2P network configuration. Assume node 1 has a query, it sends a query to all its neighbors (node 2,4,5,6,7); When these neighbor nodes receives the query message, they check if they have the queried data object. If a node has the data object, it sends the data object back to node 1. Otherwise the node sends a forward message to its all neighbor node except node 1. The TTL in query message is reduced by 1. For example, node 6 sends a query message to node 5, 7 and 19. When node 5 and node 7 receives the query message, it just

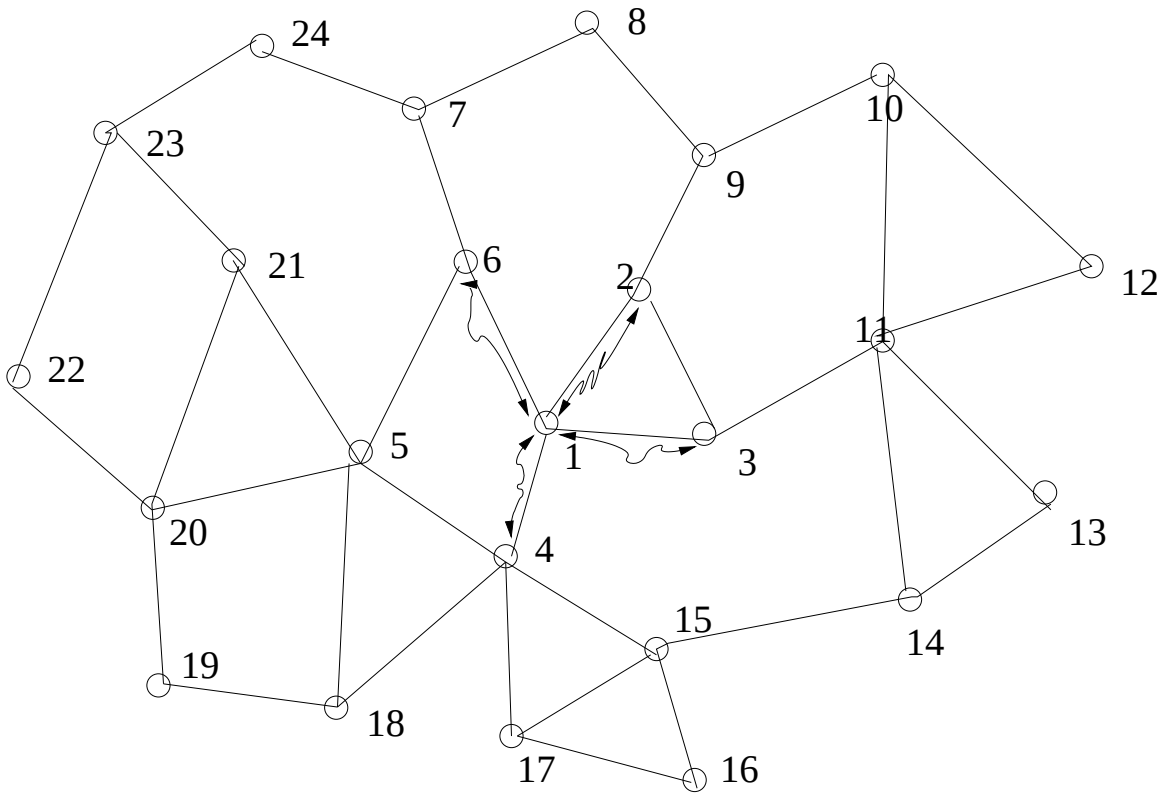

Fig. 2. Flooding Search Strategy

discards the message because they have gotten it. The node 18 processes as the same way as node 6. A node terminates the forward query message either it has the queried data object or the TTL reaches 0.

There are two problems on the flooding strategy. One is the difficult to choose appreciate TTL to control the number of hops per query. If the TTL is too high, the node unnecessary burdens the network. If the TTL is too low, the node might not find the object even though a copy exists somewhere. The other is, there are many duplicated messages introduced by flooding, particularly in high connectivity graphs.

B. Expaning Ring

In this strategy, the node starts with small TTL, and waits to see if the search is successful. If it is, then the node stops. Otherwise, the node increases the TTL and starts another flood. The process repeats until the object is found.

Figure 2 shows how the strategy works. When node 1 has a query, it sends a query message to all its

neighbor nodes (2,3,4 and 6) with TTL=1. Then waits the answer. If the data object is back, it is done. Otherwise, after some timeout, the node 1 doubles its TTL, and sends to all its neighbor nodes again. Then waits the answer. Each time the TTL is double until the maximum TTL is reached or the data object is received.

Expanding ring achieves the savings at the expense of slight increase in the delays to find the object, but reduces message overhead significantly compared with flooding strategy.

C. k-Walker

Random walk is a well-known technique, which forwards a query message to randomly choose neighbor at each step until the object is found. We call this message as a $walker$.

When using the standard random walker, it cuts down the message overhead significantly, by an order of magnitude compared to expanding ring across the network topologies. However, this efficiency comes at an order of magnitude increase in user-perceived delay of successful searches.

To decrease the delay, one can increase the number of walker, k. That is instead of just sending out one query message, a requesting node sends k query messages, and each query message takes its own random walk. The expectation is that k walkers after T steps should reach roughly the same number of nodes as 1 walker after kT steps. Therefore, by using k walkers, the delay can be cut down by a factor of k.

D. Performance Comparison

The simulations are done to compare with the three different search schemes. Two different network topologies: power-law and Gnutella are used. In each topology, there are 10000 nodes are generated. Tables I and II show the number of hops per query, number of messages per node, number of nodes visited per query and the peak number of messages. The node query is followed by a Zipf-like distribution.

TABLE I

THE PERFORMANCE COMPARISON FOR POWER-LAW NETWORK

metrics	flood	expanding ring	32-walker
$\#hops$	2.07	2.93	9.85
$\#message\ per\ node$	2.850	0.961	0.031
$\#nodes\ visited$	7923	3631	136
$peak\ messages$	464.3	98.9	12.7

TABLE II

THE PERFORMANCE COMPARISON FOR GNUTELLA

metrics	flood	expanding ring	32-walker
$\#hops$	2.03	3.05	9.39
$\#message\ per\ node$	3.548	0.423	0.058
$\#nodes\ visited$	4137	810	143
$peak\ messages$	54.5	7.0	1.6

The results show that, compared to flooding, the 32-walker random walk reduces message overhead by roughly $two\ orders\ of\ magnitude$ for all queries across two network topologies, at the expense of slight increase in the number of hops. The 32-walker random walk generally walk outperforms expanding ring as well.

E. Principles of Scalable Searches in Unstructured Networks

The above results show that the k-walker random walk is a much more scalable search method than flooding.

The key to scalable searches in unstructured network is to cover the right number of nodes as quickly as possible and with as little overhead as possible. In unstructured network, the only way to find objects is to visit enough nodes so that, statistically speaking, one of the nodes has the object. The summarization are as following:

1) $Adaptive\ termination$ is $very\ important$. TTL-based mechanism does not work. Any Adaptive/dynamic termination mechanism must avoid the implosion problem at the requester node. The checking method described above is good example of adaptive termination.
2) $Message\ duplication\ should\ be\ minimized.$ Preferably, each query should visit a node just once. More visits are wasteful in terms of the message overhead.
3) $Granularity\ of\ the\ coverage\ should\ be\ small.$ Each additional step in the search should not significantly increase the number of nodes visited. This perhaps is the fundamental difference between flooding and multiple-walker random walk. In flooding, an additional step could exponentially increase the number nodes visited; In random walk, an additional step increases the number of nodes visited by a constant. Since each search only requires a certain number of nodes to be visited, the extra nodes covered by the flooding merely increase the per-node load.

Use these constraints, a search algorithm should reduce the latency as much as possible.

IV. Data Replication

For P2P systems, one node may have some space to replicate some data object to reduce the search cost. How many copies of each object should be so that the search overhead is minimized if the total number of storage is fixed for the whole network.

Let us consider a simple model where are n nodes and m data objects. Each object i is replicated at r_i random nodes; $R = \sum_i r_i$ is the total storage capacity for the network. We assume that the objects are requested with relative rates q_i, where we normalize this by setting $\sum_i q_i = 1$. For convenience, we assume that query and replication strategies are such that $1 << r_i \leq n$ and that searches go on until a copy is found. Search consists of randomly probing sites until the desired object is found. Thus the probability $Pr(k)$ that the object is found on the k'th probe is given by:

$$Pr_i(k) = \frac{r_i}{n}(1 - \frac{r_i}{n})^{k-1} \tag{1}$$

The average search size A_i is merely the inverse of the fraction of sites which have replicas of the object:

$$A_i = \frac{n}{r_i} \tag{2}$$

We are interested in the average search size A, where $A = \sum_i q_i A_i = n \sum_i \frac{q_i}{r_i}$. The average search size essentially caputures the message overhead of efficient searches.

If there were no limit on the r_i then clearly the optimal strategy would be to replicate everything everywhere.; If $r_i = n$ then all searches become trivial. Instead, we assume that the average number of these replicas per node, $\rho = \frac{R}{n}$, is fixed and less than m. How to allocate these R replicas among the m objects is a question.

There are three replication strategies: $uniform$, $proportional$ and $square\text{-}root$.

A. Uniform Replication

This is the simplest strategy. Each data object has the same number of replicas in the network. Hence, $r_i = \frac{R}{m}$. The average search size $A_{uniform}$ is given by:

$$A_{uniform} = \sum_i q_i \frac{m}{rho} = \frac{m}{\rho} \tag{3}$$

which is independent on the query distribution. It is clear that uniformly replicating all objects, even those that are not frequently queried, is not efficient.

B. Proportional Replication

A more natural policy, one that results from having the querying nodes cache the results of their query, is to replicate $proportional$ to the querying rate: $r_i = Rq_i$. This should reduce the search sizes for more popular objects. However, a quick calculation reveals that the average remains the same:

$$A_{proportional} = n \sum_i \frac{q_i}{Rq_i} = \frac{m}{\rho} = A_{uniform} \tag{4}$$

C. Square-Root Replication

Given that uniform and proportional have the same average search size. What is the optimal way to allocate the replicas so that the average search size if minimized? A simple calculation [8] reveals that Square-Root Replication is optimal; That is, A is minimized when $r_i = \lambda\sqrt{q_i}$ where $\lambda = \frac{R}{\sum_i \sqrt{q_i}}$. The average search size is:

$$A_{optimal} = \frac{1}{\rho}(\sum_i \sqrt{q_i})^2 \tag{5}$$

Table III lists properties of the three replication strategies. Square-Root replication is such that average search size vary per object, but the variance in search size is considerably smaller than Uniform and Proportional replication.

TABLE III

THE COMPARSION OF THREE REPLICATION STRATEGIES

strategy	A	r_i	$A_i = n/r_i$	$U_i = Rq_i/r_i$
$Uniform$	m/ρ	R/m	m/ρ	$q_i m$
$Propotional$	m/ρ	$q_i R$	$1/\rho q_i$	1
$Square-Root$	$(\sum_i \sqrt{q_i})^2/\rho$	$R\sqrt{q_i}/\sum_j \sqrt{q_j}$	$\sum_j \sqrt{q_j}/\sqrt{q_i}/\rho$	$\sqrt{q_i}\sum_j \sqrt{q_j}$

D. Performance Evaluation

Fiure 3 shows the average search size for three different replication strategies. Both Uniform and Square-Root allocate to popular objects less than their 'fair share' and to less popular objects more than

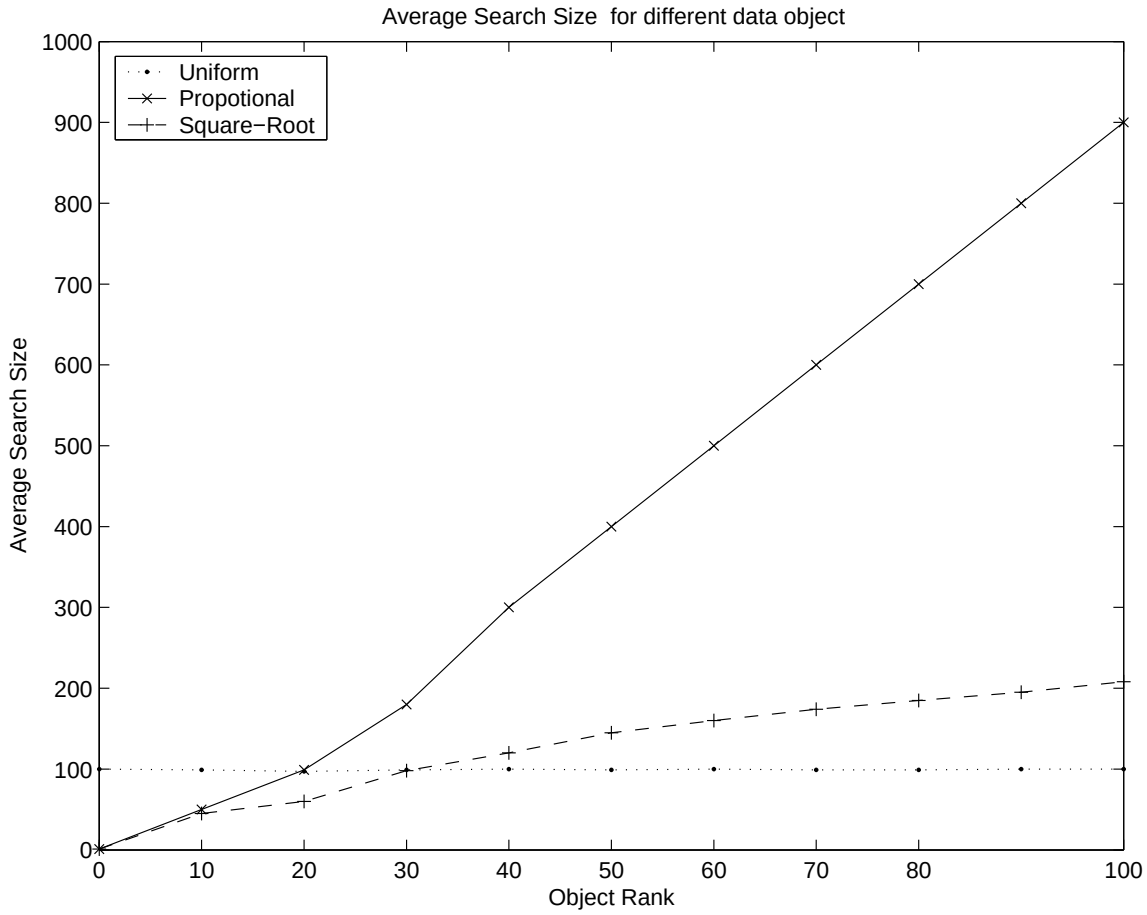

Fig. 3. Performance comparison for three replication strategies

their 'faire share' of replica. The average search size for different objects is almost a constant for Uniform replication. But varies for both Propotional and Square-Root replication strategies. The variance in average search size for different objects with Squre-Root is considerably small than with Proportional.

V. Discussion and Conclusion

This paper introduces the peer-to-peer networks, specially on the decentralized and unstructured P2P system, such as Gnutella. Three types of data search algorithms are discussed. The performance of these strategies are done to evaluate the strategies. Three different data replication strategies are also studied. The theoretical study shows that the Square-Root replication has the minimized average search size. While the proportional replication gives the optimal load balance.

The strategies discussed in the paper is based on the homogeneous networks. What is optimal for heterogeneity networks. One suggestion way is that each node has knowledge of its neighbors, the data

replication can use utility function based scheme. The utility function should combined with its neighbor content to achieve optimal solution.

The data consistency is not discussed in the paper, this is an important problem for P2P networks. How to achieve the strong data consistency with minimum overhead cost is promise reseat topic.

REFERENCES

[1] Admirana Iamnitchi, Matei Ripeanu and Ian Foster, "Locating Data in (small-World?) Peer-to-Peer Scientific Collaborations", In *Proceedings of IPTPS'02*, 2002.

[2] John Byers, Jeffrey Considine, Michael Mitzenmacher and Stanislav Rost, "Informed Content Delivery Across Adaptive Overlay Networks", In *Proceedings of SIGCOMM'02*, August 2002.

[3] Ion Stoic, Daniel Adkins, Shelley Zhuang, Scott Shenker and Sonesh Surana, "Internet Indirection Infrastructure", In *Proceedings of SIGCOMM'02*, August 2002.

[4] Open Source Community. "The free network project-rewriring the internet", In *http://freenet.sourceforge.net/*,2001

[5] Sylvia Ratnasamy, Paul Francis, Mark Handley, RichardKarp, and Scott Shenker. "A Scalable Content-Addressable Network", In *Proceedings of SIGCOMM'2001*, August 2001.

[6] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. "Chord: A Scalable Peer-to-Peer Storage Utility", In *Proceedings of SIGCOMM'2001*, August 2001.

[7] A. Rowstron and P. Druschel. "Storage management and caching in past, a large -scale, persistence peer-to-peer storage utility", In *Proceedings of SOSP'01*, 2001.

[8] Edith Cohen and Scott Shenker. "Replication Strategies in Unstructured Peer-to-Peer Networks", In *Proceedings of SIGCOMM'2002*, August 2002.

[9] Qin Lv, Sylvia Ratnasamy and Scott Shenker. "Can Heterogeneity Make Gnutella Scalable", In *Proceedings of IPTPS'2002*, 2002

[10] Clip2.com. "The Gnutella protocol specification v04", In *http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf*, 2000

[11] Megan Thomas and Ellen W. Zegura. "Gt-itm: Georgia Tech. Internetwork Topology Models", In *http://www.cc.gatch.edu/Ellen.Zegura/graphs.html*, 1997

[12] William Aiello, Fan Chung, and Linyuan Lu. "A random graph model for massive graphs", In *Proceedings of STOC'00*, p171-180, 2000.