A Comparative Analysis of P2P File Sharing Mechanisms

Anjan Mahanta and Thanaruk Theeramunkong
Sirindhorn International Institute of Technology, Thammasat University
131 Moo 5 Tiwanont Rd., Bangkadi,
Muang, Pathum Thani, Thailand, 12000
anjan@lcct.ac.th and thanaruk@siit.tu.ac.th

Abstract

This paper gives a comparative study on multi-layer file sharing mechanisms in Peer-to-Peer (P2P) systems. Based on a well-known P2P system named Gnutella, two system architectures, *1-layer* and *2-layer N*gnu architectures are proposed in order to reduce network traffic and to gain better scalability. These architectures can be applied to large-scale systems, such as e-government, e-office and e-library, which need an efficient file-sharing mechanism. To examine the efficiency and effectiveness of the proposed methods, a number of simulations are made with respect to three factors: time-to-live's (TTLs), the number of nodes and the number of queries. To compare these systems, the number of messages and the mean query hits are measured and compared with the original flat Gnutella system.

Keywords: peer-to-peer systems, distributed systems, Gnutella, 1L-Ngnu, 2L-Ngnu

1.Introduction

Recently, the peer-to-peer (P2P) systems have emerged to be a focused architecture in both significant social and technical points of views. The P2P architecture refers to a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner [1]. P2P systems usually provide infrastructure for interpersonal collaborative communities that share computing power and storage space (e.g., Gnutella [2], FreeNet [3]).

Currently, P2P systems have been used for a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner. The resources encompass computing power, data (storage/content), network bandwidth, and presence (computers, human, and other resources). The critical function can be distributed computing, data/content sharing, communication and collaboration, or platform services. Typical P2P systems reside either on the edge of the Internet or in ad-hoc networks. The advantage of P2P systems includes (1) valuable externalities, by aggregating resources through low-cost interoperability, (2) lower cost of ownership and cost sharing, by using existing infrastructure and by eliminating and distributing the maintenance costs, and (3) anonymity/privacy, by incorporating these requirement in the design and algorithms of P2P systems and applications, and by allowing peers a greater degree of autonomous control over their data and resources. In the past, P2P gained visibility with Napster's support for music sharing on the Web [4]. As more recent P2P technology, the Gnutella file sharing has been introduced in [5]. In such applications, the Gnutella protocol allows users to search for and download files from other users connected to the Internet. However, the Gnutella has several limitations, such as high download failure, scalability and security. Towards these problem, this paper propose two new architectures, *1-layer* and *2-layer N*gnu systems. By adding the concept of layers into the original Gnutella system, we can improve the download failure, scalability and security. By means of simulation experiments, we examine the efficiency and effectiveness of the proposed methods with respect to three factors: time-to-live's (TTLs), the number of nodes and the number of queries.

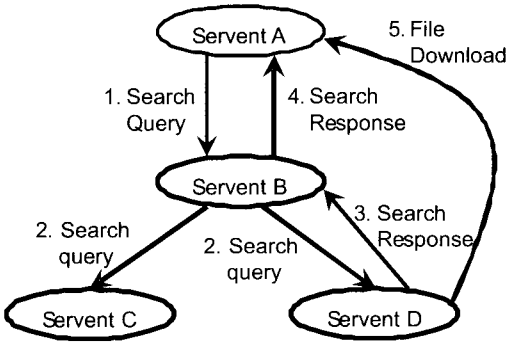

Fig. 1: Gnutella decentralized P2P model.

The number of messages and the mean query hits are evaluated and compare to the original flat Gnutella system. In the rest of this paper, the architecture of Gnutella is described in section 2. The multi-layer *N*gnu is proposed in section 3. Section 4 shows the experimental result of the simulations of multi-layer *N*gnu. The resulting analysis is given in section 5. Finally, the conclusion is made in section 6.

2.Gnutella Architecture

This section gives a brief introduction to Gnutella, the communication protocol used to search for and share files among users [6], using the example shown in Figure 1. Initially, to share files on the Gnutella network, a user (say the node A in Figure 1) starts with a networked computer that runs a Gnutella client. Since this node will work both as a server and a client, it is generally referred to as a (Gnutella) "servent" (i.e., a SERVer and a cliENT). The servent (or node) A will then connect to another Gnutella-enabled networked computer (e.g., B in Figure 1) and then A will announce its existence to B. The node B will in turn announce its existence to all of its neighboring nodes (e.g., C and D in Figure 1) that the node A is alive. This pattern will continue recursively with each new level of nodes announcing to its neighbors that the node A is alive. Once A has announced its existence to the rest of the network, the user at this node can now query the contents of the data shared across the network.

Gnutella [7] uses Time-To-Live (TTL)-based flooding to search for objects. This announcement broadcasting will end when the TTL packet information expires; that is, at each level the TTL counter will be decreased by one from some initial value until it reaches zero at which point its broadcasting will stop. To prevent users from setting this initial TTL value too high, the majority of the Gnutella servents will refuse packets with excessively high TTL values. However, from the user perspective, the maximization of the chances to find the required file, means using a as high as possible, TTL value. This is the trade-off point for this network. A low TTL means to minimize the usage of the network resources whereas a high TTL value will maximize the QoS provided to the users of the network. The Gnutella protocol consists of five descriptors that are used by the servents or the clients to communicate with each other over the network. The descriptors follow a set of rules as to exchange these descriptors. The descriptors that Gnutella uses are defined in Table 1.

Descriptor	Description
Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.
Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.
Query	The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a Query Hit if a match is found against its local data set.
Query Hit	The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.
Push	A mechanism that allows a firewalled servent to participate in the file sharing.

Table 1: Gnutella Descriptor Overview.

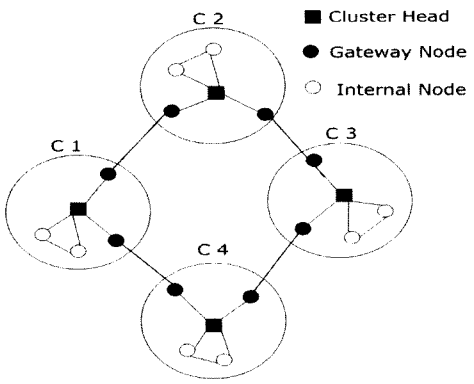

Fig. 2: The 1L-*N*gnu architecture.

3. 1-layer *N*gnu vs. 2-layer *N*gnu Architectures

In Gnutella, the communication is high due to the fact that the communication will propagate from a node to all of its neighboring nodes. Therefore, when time passes, the number of issued messages grow exponentially. In the worst case, the number of messages transferred is $k \times (k-1)$ pairs, where k is the number of nodes in the system. In order to solve this exponentially growing communication, we propose the concept of multi-layer file sharing mechanisms, called *N*gnu. Instead of a flat structure like Gnutella, the nodes are arranged into layers. In this paper, we propose two versions of *N*gnu, called 1-layer (1L-*N*gnu) and 2-layer (2L-*N*gnu). The *N*gnu architecture is dictated by geographical relationships between nodes and "logical" hierarchy of clusters and sub-clusters in which the members move as a group.

3.1 The 1L-*N*gnu Architecture

Figure 2 illustrates an example of the physical clustering in the proposed 1L-*N*gnu architecture. In this architecture, the existing nodes (computers) are grouped into a set of clusters. Although it is possible to consider different approaches in clustering the nodes, distance constraints need to be considered in order to reduce high traffic communication. In each cluster, there are three main types of nodes; namely (1) the cluster head, (2) gateway nodes and (3) internal nodes. The cluster head acts as a local coordinator of transmissions within the cluster. Gateway nodes are located at the boundary of the cluster, and performing communication with nodes outside the cluster. In general, each node in the cluster has their unique identifier. When a new internal node joins the network, it will register itself to the cluster head. Therefore, the cluster head keeps tracks of all internal nodes within its responsible cluster. When a node sends out a request (a query) for an object, it initially sends the query to the cluster head. Later the cluster head forwards the query request to all the existing internal nodes present in its responsible cluster. As done in Gnutella, the query request will be propagated according to the reply message; object found or object not found. When the object is found in a particular internal node, the cluster head will be informed and then it will send the information, including its ID, of the node which possesses the object, to the node which made the request. Finally, a direct link between these two nodes is established and the object is transferred or shared. On the other hand, if the requested object does not exist in the local cluster, the cluster head forwards the query request to the neighboring cluster head via the cluster's gateway nodes. If the requested object is found in a neighboring cluster, a "virtual link" between the requesting node and the node holding the object will be established and then the object is transferred or shared. The drawback of the 1L-*N*gnu with respect to Gnutella is the need to continuously update information in the cluster head when internal nodes join and leave the network. On the other hand, 1L-*N*gnu provides security with node registration and node identification and can handle scalability because the entire system is divided into a set of clusters with respect to its geographical location. This feature is desirable since it is more likely that a requested object tends to be available in the same area, i.e., users belonging to the same area normally possess same likes and dislikes. At the same time, the 1L-*N*gnu still holds the P2P property within the cluster by propagating a query from a node to its neighboring nodes. However, for a remote area, the query is processed in a centralized manner via the cluster head. Hence, the logical partitioning of the nodes into separate clusters could provide scalability, security, improve query performance and minimize download failures and reduce or

```
1 : EXECQUERY(query, respond)
2 : begin
3 :   pid = GETPID();
4 :   if( FINDOBJECT(query) ) then
5 :   respond = {pid};
6 :   else
7 :    if (TTL < threshold) then
8 :      nlist = GETNEIGHBOR(pid);
9 :      foreach nid in nlist
10 :       EXECQUERY(query, respond1)@nid;
11 :       respond = respond ∪ respond1;
12 :     else
13 :       respond = ∅;
14 :     end if
15 :   end if
16 :   if( ISCLUSTERHEAD(pid) AND
17 :      respond == ∅ ) then
18 :    clist = NEIGHBORCLUSTER(pid);
19 :    foreach cid in clist
20 :      EXECQUERY(query, respond1)@cid;
21 :      respond = respond ∪ respond1;
22 :   else
23 :      respond = ∅;
24 :   end if
25 : end
```

```
26 : INITIALQUERY(query, respond)
27 : begin
28 :   pid = GETPID();
29 :   if( ISCLUSTERHEAD(pid) ) then
30 :     EXECQUERY(query, respond)@pid;
31 :   else
32 :     cid = CLUSTERHEAD();
33 :     EXECQUERY(query, respond)@cid;
34 :   end if
35 : end
```

Figure 3: The pseudo code for 1L-*N*gnu.

separate network support cost. The algorithm for the 1L-*N*gnu mechanism is shown in Figure 3.

When a node dispatches a query to find a resource, the INITIALQUERY function will be executed. The GETPID function returns the ID of the node (*pid*) executing the function. The ISCLUSTERHEAD checks whether the node is the cluster head of that cluster or not. The EXECQUERY function transfers the query for the requested object to the node specified by its ID (i.e., *@pid* or *@cid*). The CLUSTERHEAD function outputs the ID of the cluster head node.

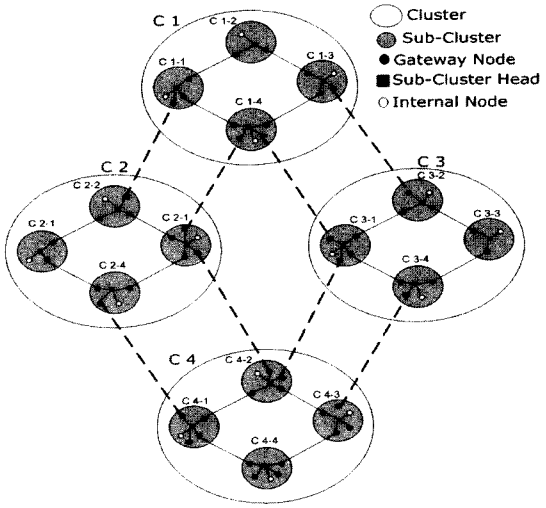

Fig. 4: The 2L-*N*gnu architecture.

The FINDOBJECT function returns 1 when the object matching with the query is found inside the node. The GETNEIGHBOR function returns the list of neighboring nodes (*nlist*). The NEIGHBORCLUSTER returns the list of cluster head nodes which are neighbors to the current node.

3.2 The 2L-*N*gnu Architecture

The 2L-*N*gnu architecture aims to provide the same functionality to that of the 1L-*N*gnu but more scalable with one more layers added. For example, as depicted in Figure 4, the inner layer (later called layer 1) of the 2L-*N*gnu system occupies a number of physical sub-clusters in a cluster. In the figure, there are 4 sub-clusters forming a cluster. A cluster is organized in the form of a 1L-*N*gnu architecture. That is, there are a sub-cluster head, gateway nodes and the internal nodes for each sub-cluster. In the outer layer (later called layer 2), a system is virtually formed by a number of clusters. In the outer layer there are no cluster heads to communicate with the other clusters so the Sub-cluster heads communicate with local Sub-clusters as well as with the neighboring Sub-clusters. Sub-clusters and clusters are organized by taking geographical locations into account.

In each sub-cluster, the sub-cluster head node acts as a local coordinator of transmissions within the sub-cluster. Analogous to the 1L-*N*gnu architecture, the sub-cluster head acts as the cluster head in the 1L-*N*gnu architecture. When an internal node queries an object in the network, the process is the same as the 1L-*N*gnu architecture, except the request reaches the boundary of the cluster. At the boundary, the request is transferred to gateway nodes of the neighboring cluster. The pseudo code for the 2L-*N*gnu is shown in Figure 5. The 2L-*N*gnu code is similar to the 1L-*N*gnu code, but also includes the codes for handling boundary nodes, appearing at line numbers 23 to 30. When the request reaches a boundary between any two clusters, the gateway (boundary) nodes dispatch a query to the gateway nodes of its neighboring clusters. The ISBOUNDARYNODE function returns true if the current node (*pid*) is a boundary node. The GATEWAYNODE function returns the list of neighboring gateway nodes (*glist*). Then the request is transferred to the neighboring clusters via these gateway nodes.

```
1:  EXECQUERY(query, respond)
2:  begin
3:    pid = GETPID();
4:    if( FINDOBJECT(query) ) then
5:    respond = {pid};
6:    else
7:     if (TTL < threshold) then
8:      nlist = GETNEIGHBOR(pid);
9:      foreach nid in nlist
10:       EXECQUERY(query, respond1)@nid;
11:       respond = respond ∪ respond1;
12:     else
13:       respond = ∅;
14:     end if
15:   end if
16:   if( ISSUBCLUSTERHEAD(pid) AND
17:      respond == ∅ ) then
18:    clist = NEIGHBORCLUSTER(pid);
19:    foreach cid in clist
20:      EXECQUERY(query, respond1)@cid;
21:      respond = respond ∪ respond1;
22:   else
23:    if ( ISBOUNDARYNODE(pid) ) then
24:      glist = GATEWAYNODE(pid);
25:      foreach gid in glist
26:        EXECQUERY(query, respond1)@gid;
27:        respond = respond ∪ respond1;
28:    else
29:      respond = ∅;
30:    end if
31:   end if
32: end

33: INITIALQUERY(query, respond)
34: begin
35:  pid = GETPID();
36:  if( ISSUBCLUSTERHEAD(pid) ) then
37:    EXECQUERY(query, respond)@pid;
38:  else
39:    cid = SUBCLUSTERHEAD();
40:    EXECQUERY(query, respond)@cid;
41:  end if
42: end
```

Figure 5: The pseudo code for 2L-*N*gnu.

4. Simulation Experimental Results

4.1 Experiment Environment

In order to evaluate the proposed architectures, a simulator is implemented using Visual Basic and its networking functions. Using the simulator, a set of experiments is made to compare the proposed system architectures 1L-*N*gnu and 2L-Ngnu to a Gnutella-style P2P network [8]. The basic scheme of the simulation is to initialize a P2P file sharing system based on observed statistical data and educated assumptions. Each peer of Gnutella, domain and sub-domain of 1L-*N*gnu and 2L-*N*gnu is simulated as a separate process, and each of them maintains a FIFO queue for incoming requests. A workload generator continues generating query and download requests, the intervals between successive requests follow an exponential distribution. A peer process is an infinite loop, which continuously services incoming requests and performs appropriate actions based on the request type. To account for the dynamic nature of the P2P network, we created a process that emulates the behavior of peers leaving and joining the network. The simulation used various statistical distributions to simulate the behavior or resource distributions within the network. These data are derived from measurement studies by Ripeanu et al. and Saroiu et al. [8], and therefore our simulation is close to real Gnutella networks.

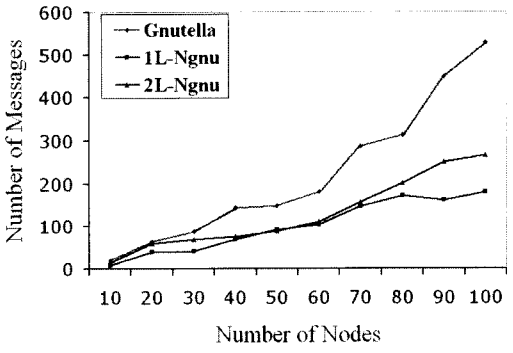

Fig. 6: Total network traffic in 2L-*N*gnu, 1L-*N*gnu and Gnutella (TTL = 7, Time = 10 ms.)

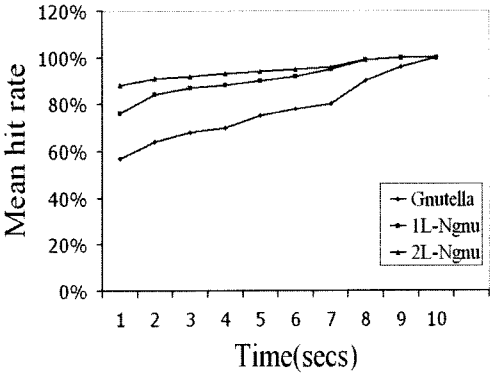

Fig. 8: Comparison of query hit rate.

4.2 Simulation Settings

The following assumptions are applied to all the system architectures; Gnutella, 1L-Ngnu and 2L-Ngnu for all experiments if no other settings are specified.

- There are 10000 nodes with identical capabilities participating in the network.
- The TTL value can be set to any value. The default value is 7 if not specified. It is the common value used in most Gnutella networks.
- Two alternative queries are (1) random query and (2) static query. A random query is a query starting from a random location and searching for some random objects while a static query is one that starting from a fixed location with a fixed time interval and then searching for specified objects
- The simulation duration can be set to any length.

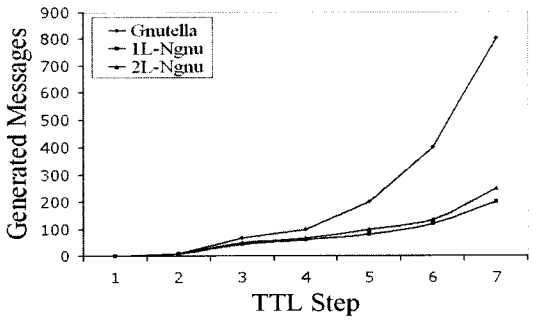

Fig.7: Number of generated messages in each TTL.

4.3 Simulation Results

As the first step, the total number of network messages is examined. With the TTL of 7, the system is simulated by varying the number of nodes. The simulation duration for each test is 10 milliseconds. The total numbers of generated messages are shown in Figure 6. From the result, as the number of nodes increases the network traffic in Gnutella is much higher than that of 1L-Ngnu or 2L-Ngnu throughout the entire simulation. The 1L-Ngnu has the lowest network traffic. This is due to the distribution of nodes into clusters as per their geographical location.

As the second experiment, the network traffic of the three systems is examined with different TTL values starting from 1 to 7. By default, the TTL value is set to 7. The result is shown in Figure 7. When the TTL is small (1 or 2), there are not so many messages generated and all the systems have almost the same number of messages. However, when the TTL value is set to a larger number, it is observed that the Gnutella network generates much more messages than the 1L-*N*gnu and 2L-*N*gnu do. The 1L-*N*gnu achieves the lowest network traffic.

In the third experiment, we compare the query hit rate percentages of these three systems. In this experiment, the TTL is set to its default value (i.e. 7), the number of nodes is set to 10000 and the number of queries is set to 300. The query hit is recorded by varying the time to run the simulation. Initially, the Gnutella network gains the lowest query hit rate of 60%, 1L-*N*gnu has a query hit rate of 80% and 2L-*N*gnu has a query hit rate of nearly 90%. When

the simulation duration is set longer, the query hit rate increases for all the systems and finally their query-hit rates are saturated to 100%, especially after 10 seconds.

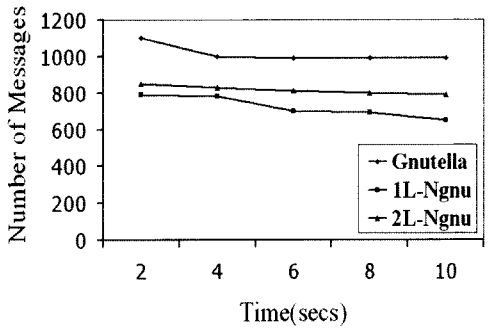

Figure 9: Number of messages with various durations.

Figure 9 displays the simulation result of a stable network of 100 nodes. The TTL is set to the default value (i.e. 7) and the number of queries is set to 300. It is observed that except for the initial state, all the systems have almost a constant number of messages when the simulation is recorded with various runtime durations. This means the traffic is almost stable and independent of the time passed.

5. Conclusion

In this paper, two peer-to-peer (P2P) system architectures for file sharing mechanism, called 1L-*N*gnu and 2L-*N*gnu, are presented. In the architectures, a multi-layer mechanism was introduced to extend a well-known P2P system named Gnutella. A cluster head is established to be a local coordinator of transmissions within the cluster. When a node joins the network, it registers itself to the cluster head (or sub-cluster head). The nodes belonging to the same geographical location are grouped in the same cluster (or sub-cluster). These system architectures provide more scalable alternatives to existing Gnutella algorithms [9], focusing on the search and replication aspects. By simulation experiments, it was found that the total number of network messages in 1L-*N*gnu is considerably less than both Gnutella and 2L-*N*gnu. But if we compare only the two systems, i.e. Gnutella and 2L-Ngnu, the number of messages generated in the 2L-Ngnu are much less than that of Gnutella. The 1L-*N*gnu and 2L-*N*gnu not only have less network traffic than Gnutella but also provide scalability, security by registering the users as well as reduction in download failure due to the unavailability of the cluster head or the sub-cluster head. The 1L-Ngnu and 2L-Ngnu architectures can be implemented as a part of file-sharing applications such as e-government and e-library [10]. In the future, in order for Ngnu to establish itself among individuals and organizations as a file sharing method of choice, we are going to provide *N*gnu with a naming system for its dynamic nodes and objects. It is also interesting to explore how a system with a higher degree of layering performs.

6. Acknowledgement

The authors would like to thank Lampang College of Commerce and Technology (LCCT) for providing financial support through the project. Many thanks are given to Dr. Kritchalach Thitikamol for his useful comments and suggestions. We also would like to thank to Dr. Nimit Jivasantikarn for his encouragement and useful comments.

7. References

[1] Mahanta A., Comparative Simulation Analysis: Proposal of a new system architecture – *N*-Gnu, In Proceedings of International Conference on Telecomputing & Information Technology (ICTIT-2004), 22-24 September 2004, Amman, Jordan, pp. 244-250.

[2] Clip2, The Gnutella Protocol Specification v0.4 (Document Revision 1.2), June15,2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[3] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, Freenet: A Distributed Anonymous Information Storage and Retrieval System. In Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability (LNCS 2009), Berkeley, CA, pp. 46-68, 2000.

[4] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system in designing privacy enhancing technologies. International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, pp. 46-66, 2001.

[5] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, Zhichen Xu, Peer-to-Peer Computing, Technical Report of HP Laboratories Palo Alto, Report No. HPL-2002-57, 8 March 2002.

[6] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, R., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. OceanStore: An Architecture for Global-Scale Persistent Storage. Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), pp. 190-201, November 2000.

[7] Igor Ivkovic – Software Architecture Group (SWAG) Improving Gnutella Protocol: Protocol Analysis and Research Proposals, Department of Computer Science, Computer Networking Group Seminar Series, University of Waterloo, Canada, November 2001.

[8] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," Tech.Rep. 01-06-02, UW-CSE, June 2001.

[9] Kunwadee Sripanidkulchai, The popularity of Gnutella queries and its implications on scalability, Carnegie Mellon University, Featured on O'Reilly's www.openp2p.com website, Technical Report, February 2001.

[10] B.Y.Zhao, J.Kubiatowicz and A.D.Joseph, Tapestry: An Infrastructure for fault-tolerant wide-area locating and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.

[11] Schloss Elmau, Dabek, F., Brunskill, E., Kaashoek, F., Karger, D., Morris, R., Stoica, I., and Balakrishnan, H. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), Germany, pp. 81-86, May 2001.