University of Surrey

An Efficient Fault Tolerant Approach to Resource Discovery in P2P Networks

J. Salter and N. Antonopoulos

May 2004

Computing Sciences Report

CS-04-02

One of the authors (JS) gratefully acknowledges the support of this work by the EPSRC under its Doctoral Training Grant scheme.

An Efficient Fault Tolerant Approach to Resource Discovery in P2P Networks

James Salter
University of Surrey
j.salter@surrey.ac.uk

Nick Antonopoulos
University of Surrey
n.antonopoulos@surrey.ac.uk

Abstract

Current research into resource discovery in peer-to-peer networks is largely focussed on the use of Distributed Hash Tables and multi-layered topologies. In this paper we present a resource discovery system capable of resolving keyword-value pair queries, based on a two-layered Chord ring architecture. We show how the base topology augmented with shortcuts between layers and selection of ring member nodes provides performance benefits and the potential for increased fault tolerance over other similar resource discovery systems.

1. Introduction

The use of Distributed Hash Tables (DHTs) for resource discovery in peer-to-peer networks has become a highly studied research area. More recently, systems utilising multiple DHT structures have begun to be explored.

We present a two-layered multi-ring architecture based on Chord [11] supporting keyword-value pair queries. For the majority of topology configurations we demonstrate a reduction in messages required to route a query in worst-case scenarios compared with a single-layered Chord ring.

We show that using techniques that allow for shortcuts between layers and selection of high quality nodes, fault tolerance of our system can be improved and under certain conditions become comparable with Chord and better than other multi-ring topologies.

The remainder of the paper is structured as follows: In section 2 we review related work, before describing the main architecture and algorithms of our system in section 3. Analyses of query routing performance and fault tolerance are presented in sections 4 and 5 respectively. We outline future work in section 6 and present our conclusions in section 7.

2. Related Work

Peer-to-Peer networks utilising flooding protocols such as Gnutella [7] provide a level of fault tolerance that can only be bettered by increasing numbers of connections between nodes, such as in fully connected meshes. However, the amount of routing information that must be stored and updated is unscalable in large-scale P2P environments. Similarly, it has been proven [8] that traffic generated by Gnutella queries can also become unscalable.

Several systems based on Distributed Hash Tables have been proposed to substantially reduce query and update message traffic. Chord [11] organises nodes into a ring and can route a query in O(log N) messages while maintaining routing information about only $\log_2(N)$ nodes on every node in the network. Updates following a node joining or leaving the network require $O(\log^2 N)$ messages.

Other widely used examples of DHTs in P2P networks include CAN [6], Pastry [9] and Tapestry [15].

Multi-layered DHT structures are perhaps the next logical development, aiming to improve further upon flat DHTs. Several researchers have proposed systems similar to ours involving multiple rings, but to our knowledge none have so far been able to show substantial improvement in the number of messages used to route a query in worst case scenarios compared to a similarly sized Chord ring. The separation of keywords and values that allows us to utilise Preference Lists and cache keyword ring entry points for faster query resolution also appears unique.

PLANES [12] partitions a single Chord ring into multiple smaller rings, but little regard is given to the information stored on each node in individual clusters. Although the top layer of the PLANES topology allows the cluster holding the target key to be located quickly, there are no guarantees that the same cluster will also contain similar keys to the one originally searched for.

Garcés-Erice et al [3] present Hierarchical Peer-to-Peer Systems and give a case study instantiation of a modified version of Chord. The lower level overlay is split into a number of sub-rings. One or more nodes from each sub-ring are inserted into the top-level overlay. A reduction in routing messages is demonstrated by assuming powerful machines with greater availability and connectivity are

used in the top-level overlay, thus providing a smaller probability of node failure. However, ignoring probabilities of failure, reduction in messages is not possible (see section 4).

In HIERAS [13] and Brocade [14], a single overlay contains all nodes. HIERAS rings co-exist in different layers, but each node is a member of a ring in each layer. Although average routing latency is less than Chord, HIERAS requires a larger average number of routing hops. In Brocade, a second overlay layer is used to improve performance of long range routing. However, nodes are grouped only by administrative domain rather than by the information contained within them.

In Coral [2], searches begin in a sub-group containing nodes in the same geographical region, continuing on a cluster with continental coverage before eventually switching to a single planet-wide cluster if the lookup has not been resolved in the intermediate stages. While searches that are resolved within the regional cluster may be faster than those in our system, more messages will be consumed to route a query in the worst-case.

Diminished Chord [5] is a new version of Chord in which virtual sub-rings are created within a single Chord ring. Query routing requires O(log *N*) hops as in standard Chord, but the authors argue their approach requires fewer resources than creating a separate P2P network for each subgroup. Our Lazy Ring Creation scheme maximises usage of both node capacity and query and update traffic by only creating rings for subgroups (keywords) when necessary.

3. The System

Our P2P network is based on a set of Chord rings and is designed to provide a list of machines hosting matching resources in response to a query composed of keyword-value pairs. Each ring is responsible for a single keyword. Within each keyword ring, nodes are responsible for indexing resources matching one or more values associated with the ring's keyword. For example, if a web service was registered against 'distribution=binomial' then 'distribution' would be the name of the keyword ring and 'binomial' would be hashed to give the ID of the node in the ring holding a reference to the service. Nodes in rings do not necessarily host the actual resources related to keywords and values they are responsible for, but hold a list of IP addresses of machines hosting the resources.

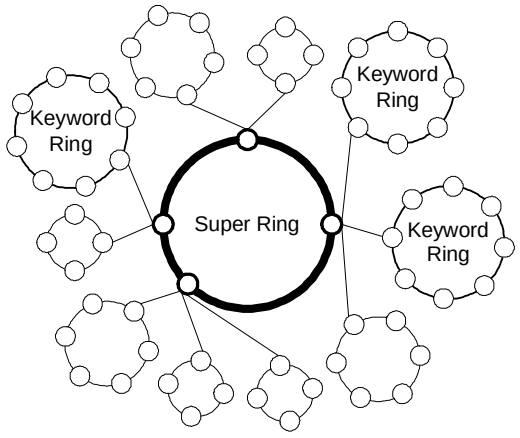

Figure 1. Central super ring pointing to multiple keyword rings

A further ring named the super ring organises a group of super nodes that host pointers to each keyword ring (see Figure 1). Each super node can be responsible for pointing to multiple keyword rings.

A node can become a member of multiple keyword rings, for up to as many as it has capacity for. In addition to storage capacity, this capacity also refers to the workload a machine can handle. Nodes which are members of popular keyword rings must be able to handle more query throughput than a node in a less popular keyword ring.

Our system provides a true hierarchy of labels, rather than partitioning groups of nodes into clusters that has been the focus of most previous work. In effect, it looks up two independent but related keys, while other systems look up a single key in two stages.

3.1. Query Processing

For simplicity, throughout this paper we will assume that a query will be made for a single keyword-value pair. Multiple pair queries are possible within our system, but will be described in detail in future literature.

The machine on which a query originates should know the address of at least one super node and uses this as an entry point into the super ring. The query keyword is hashed to give the ID of the super node responsible for that keyword. The query is routed to this super node using the standard Chord routing algorithm. Addresses of multiple nodes that are members of the required keyword ring are listed on the super node. One of these is selected and the query is forwarded into the keyword ring.

The value of the keyword is hashed to give the ID of a node in the keyword ring and the query is sent to it using the standard Chord routing algorithm. The node hosts a list of providers of resources matching the keyword and value specified. This list is returned to the query instigator.

3.1.1. Preference Lists. The list of providers returned in response to a query is cached on the machine originating the query, so that future queries for the same keyword-value pair can be resolved more quickly. As the query is repeated, the providers are ordered by those most likely to supply the required resource, based on how often they

have done so in the past. If no provider listed in a preference list can supply the required resource, our system can naturally be used to retrieve a more up to date resource list. Several criteria may affect the likelihood a provider listed will actually allow access to the resource (or whether it is desirable to access the resource from the provider), such as compatible security policies, workload or connectivity between it and the consumer.

Preference lists can also be used as shortcuts to bypass the super ring for queries where the keyword has been previously searched for but the value is different. In addition to the list of providers, the addresses of the keyword ring node that hosted the list and the super node that pointed to the keyword ring are also stored in the preference list. The keyword ring node can be used as an entry point into the keyword ring for the repeated keyword, eliminating the need to route the query through the super ring to find a keyword ring entry point.

Use of preference lists not only reduces the number of messages needed for query processing, but also increases the fault tolerance of the system. For repeated keyword queries, no super ring node need be present for the query to be processed as long as the preference list can provide at least one address of a node still active and a member of the relevant keyword ring. In ideal situations, the indexing structure can be bypassed entirely for repeat queries, providing a direct link between consumers and providers.

The size of preference lists can be constrained to avoid scalability issues, with the least often queried keywords and values being removed as the storage space allocated for them becomes full.

3.2. Joining the System and Registration of New Resources

To join the system, a machine must know the address of a node already in the network. The machine inherits the preference list of the existing node, which should list the address of at least one super node (the minimum information a node needs to provide an entry point into the super ring for query processing). The actual lists of resource providers are not inherited, as they will almost certainly be different or inaccurate for the new machine.

If the machine does not wish to provide any resources, it need not do anything else. There is no expectation that it will register within any structure, which allows for dynamicity in session times for nodes acting exclusively as consumers.

In the simplest case, whenever a node wishes to register a new resource it runs a query for every keyword-value pair the resource is to be listed against, but the query adds the node's IP address to the relevant keyword ring node's list of providers instead of returning a list of results. If the node has not registered a resource before it must also join the physical resource node ring, a special Chord ring containing all nodes with registered services. This ring is used to select nodes available to participate in the super ring and keyword rings.

A resource can be removed from the system in a similar manner to addition, where a query is performed for each keyword-value pair associated with the resource, this time removing references to it from the relevant keyword ring nodes. No attempt is made to remove references to the resource from preference lists. As no information is available about which preference lists reference the resource, a broadcast throughout the network would be necessary. We assume it is less costly to remove references to the resource from a preference list when its removal is discovered following a query.

If a resource were registered against a keyword that does not exist in the system, a new keyword ring would ordinarily need to be created so that a node would be available to list the resource provider. However, this would be wasteful if no queries were ever submitted for the keyword. Instead, we use Lazy Ring Creation, where a keyword ring is only created when the number of queries for the keyword or the number of resources registered against the keyword rises above certain thresholds. Before the keyword ring is created, resource information is held in contention on the super node which ID matches the hash of the new keyword.

If the super node does not have sufficient capacity (storage and workload) for the new keyword, a new super node is selected from those present in the physical resource node ring and inserted into the super ring. The identifier of this new super node is chosen such that it will host the new keyword in addition to part of the workload of the existing super node.

When one of the thresholds above which a keyword ring is created has been exceeded, nodes with available capacity are selected to become members of the ring from those listed in the physical resource node ring. This gives the opportunity of, in addition to capacity, selecting a node with a small probability of failure.

The search for machines starts from the physical machine hosting the super node holding the keyword in contention. An identifier of another machine in the ring is randomly selected and a Chord-style query routed from machine to machine towards the target. This initial randomisation stops situations in which the 'best' available machines are targeted by multiple concurrent requests. The machine at each hop is checked to see if it matches the capacity and failure requirements. The query continues until either sufficient machines have been found or the target machine is reached. If the target machine has been reached, the query can continue to be forwarded between neighbouring machines until it is resolved or the target machine is visited again (implying there are not enough machines in the ring with the required capacity).

This selection process and demonstrations of its scalability are presented in [1].

4. Query Routing Message Costs

As we use the standard Chord routing algorithm, our system uses the same volume of messages to route a query inside our rings. [11] has shown the worst case number of messages to route a query through a standard Chord ring is $\log_2(N)$ and $\frac{1}{2}\log_2(N)$ in the average case, where N is the number of nodes in the ring. In the worst case, it follows that the number of messages required to route a query through our structure would be $\log_2(s)+\log_2(k)+1$ where s is the number of super ring nodes and k is the number of nodes in the keyword ring. The extra message is needed to jump between the two rings, because each is independent unlike in systems such as [3] where a node in the top-level topology is also a member of the sub-ring to which it is related.

We can show improved performance for queries containing repeated keywords. Since an entry point into the keyword ring is already known, the worst case number of messages becomes $\log_2(k)$.

A comparable standard Chord ring would utilise the same number of physical machines as used in our system, but the total number of nodes in the ring would be less than the number of nodes in all our keyword rings, because each machine in our system can be a member of multiple rings (creating multiple logical nodes for each machine). If each machine in our system is a member of r keyword rings, assuming each keyword ring had the same number of nodes and each super node was responsible for the same number of keywords, the standard Chord ring would contain $\frac{wsk}{r}$ nodes, where w is the number of keywords for which each super node is responsible. For all systems where $\frac{w}{r}>2$, we can show a reduction in worst-case number of messages over standard Chord taken to route a query for a keyword not searched for previously: $\log_2(\frac{wsk}{r}) > \log_2(s)+\log_2(k)+1$.

Improvement in message cost for repeat keyword queries is not possible in standard Chord, since there is no logical ordering of the nodes into keyword groups and there is no way to determine a segment of the ring that contains possible answers. Comparing with our system in the worst case gives $\log_2(\frac{wsk}{r}) > \log_2(k)$. By using shortcuts provided by preference lists, we can realise savings of $\log_2(\frac{ws}{r})$ messages in the worst-case for every repeated keyword query.

Comparing with the Chord instantiation of a Hierarchical Peer-to-Peer system shown in [3], to create a topology containing the same number of keyword rings and nodes within each keyword ring as in our system, the number of superpeers required in their system would be ws since each superpeer can only be responsible for a single subgroup. In the worst-case (assuming a stable network without node failure) the number of messages required to route a query in their instantiation would be $\log_2(wsk)$, which is greater than in our system for $w>2$.

For repeated-keyword queries, we again show improvement. Hierarchical Peer-to-Peer does not contain a mechanism similar to our shortcuts, so queries will always require $\log_2(wsk)$ messages in the worst case. In comparison with our system, we can reduce the required worst-case messages by $\log_2(ws)$: $\log_2(wsk)>\log_2(k)$.

Depending on how many keywords each super node holds and whether a query is for a new keyword-value pair, a repeated keyword or a repeated keyword-value, our system can give varying degrees of message savings in comparison with both Chord and multi-ring topologies.

5. Fault Tolerance Measurements

To measure the success of a Peer-to-Peer network a single metric must not be analysed in isolation or researchers may be tempted to design systems with, for example, good performance whilst ignoring fault tolerance. We have shown above how our system can route queries in an efficient manner utilising a scalable number of messages. Efficiency in routing information updates is inherited from the base Chord protocol and in [1] we discuss in detail our Lazy Ring Creation procedure and show how it yields results in a scalable number of messages. We now present some initial calculations to show the level of fault tolerance our system can provide. We assume that the nodes on which queries begin and their final targets are sufficiently randomly distributed, and that nodes are uniformly distributed throughout the ring. For simplicity, we also assume that rings operate without successor lists.

Suppose the probability of a node failure is p. The probability that a single finger table entry on a node has failed is therefore also p. Since each node holds $\log_2(n)$ finger table entries in a ring of n nodes, the probability that a node's finger table has completely failed is $p^{\log(n)}$. It follows that the probability that not all finger table entries on a node have failed is $1-p^{\log(n)}$. In the average case a query will be routed in $\frac{1}{2}\log_2(n)$ hops, so the probability that no node encountered in the query path will have a completely failed finger table is $(1-p^{\log(n)})^{\frac{1}{2}\log(n)}$. The probability of query routing failure would therefore be $1-(1-p^{\log(n)})^{\frac{1}{2}\log(n)}$.

However, the above formula assumes that queries can always be routed if at least one finger table entry on each node exists, and the number of hops will not increase if finger table entries have failed. In reality, queries may fail if none of the entries in a node's finger table which point *towards* the target node still exist. In addition, if an

alternative finger table entry must be used because the correct one points to a node that has failed, the number of hops taken to resolve the query may increase.

We must adjust the $p^{\log(n)}$ in our formula to take into consideration the probability that a query may fail on a node with a partially complete finger table. For example, for a finger table containing 8 entries, where i is the number of finger table entries that have failed:

$P(i=8) = p^8$
$P(i=7) = 8p^7(1-p)$
$P(i=6) = 28p^6(1-p)^2$
$P(i=5) = 56p^5(1-p)^3$
etc…

It can be seen that the probability of partial finger table failure follows a binomial distribution, Bin(8,p) for the above example.

The probability that a finger table entry is valid given a certain number of entries have failed depends on how many of the entries have failed and how near the query is to its target. If, for instance, one entry had failed, the chance of finding an alternative node to route the query to is higher than if three entries had failed. From a comparison with experimental data from [4], a negative exponential function appears to give a good approximation of the distribution of these probabilities. For the data available, $e^{-i/2}$ yields the best results. We are in the process of establishing whether the denominator (2) is constant or will vary depending on network size.

From above, we replace $p^{\log(n)}$ with:

$$\sum_{i=0}^{\log_2(n)} \binom{\log_2(n)}{\log_2(n)-i} p^{\log_2(n)-i}(1-p)^i e^{-i/2}$$

The percentage increase in mean path length of a query with p probability of node failure is approximated by $p^{1.7}$, derived from calculations given in [11]. Again, this gives a good approximation of experimental results from [4]. The mean path length becomes $\frac{1}{2}\log_2(n)(1+p^{1.7})$.

The probability of query routing failure therefore becomes:

$$1-\left[1-\sum_{i=0}^{\log_2(n)} \binom{\log_2(n)}{\log_2(n)-i} p^{\log_2(n)-i}(1-p)^i e^{-i/2}\right]^{\left[\frac{1}{2}\log_2(n)(1+p^{1.7})\right]}$$

In Figure 2 we plot the probability of query routing failure in different sized single Chord rings for different probabilities of node failure. It is clear that larger rings yield increased fault tolerance. Since new queries in our system are routed through two rings that are likely to be small compared to an analogous system's single Chord ring, the fault tolerance of our system will be worse than standard Chord.

However, our Lazy Ring Creation gives the opportunity to select which machines are used as nodes within rings. As all the machines within the physical resource node ring are resource providers, it may also be true that they are also higher quality compared to other machines purely acting as consumers. Figure 2 shows that the probability of node failure is the most important

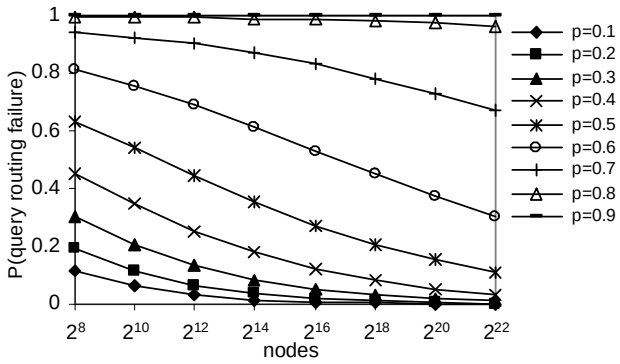

Figure 2. Probability of routing failure

factor in determining the fault tolerance of a ring. For example, to achieve a 20% chance of query routing failure, approximately 256 nodes with 20% probability of failure must be present in the ring. Alternatively, to achieve the same probability of query routing failure, ~1024 nodes with 30% probability of failure or even ~16384 nodes with 40% probability of failure must be used. Reducing probability of node failure by a few percent has a similar effect on probability of query routing failure to substantially increasing the number of nodes in the ring. [10] shows that heterogeneity between nodes in current peer-to-peer systems can be extreme. Therefore, selection of a small number of high quality nodes should lead to a lower average probability of failure of nodes within the ring than in a larger ring containing more nodes that are heterogeneous where probability of node failure would be more variable.

If we can select higher quality nodes in our smaller sized rings, our fault tolerance could be competitive with much larger rings containing nodes with higher probabilities of failure. If, however, all nodes have very similar probabilities of failure, it is unlikely our system could compete with standard Chord in terms of fault tolerance, even for repeated-keyword queries where our super ring would be bypassed completely (the equivalent of saying the probability of query failure within the super ring is zero). It is worth noting that it is better to construct a ring with small number of high quality nodes than a ring with a larger number of poorer quality nodes, because smaller rings also provide performance benefits.

In comparison with the Chord instantiation of Hierarchical Peer-to-Peer Systems [3] for topologies containing similar numbers of nodes, our system will exhibit higher probability of query routing failure for new-keyword queries. Their primary ring (super ring) will contain more nodes, because they need one primary ring node per subgroup, thus leading to greater fault tolerance. However, their primary ring may contain nodes more prone to failure, as they select the best nodes from each subgroup to become members of the primary ring. If some subgroups contained only unreliable nodes,

these would be promoted to the primary ring, increasing overall probability of primary ring node failure and thus lowering fault tolerance.

Our system can show improvement over Hierarchical Peer-to-Peer Systems' fault tolerance for repeated-keyword queries by bypassing the super ring and through our selection procedure for keyword ring nodes. In their system, nodes are formed into clusters and then a cluster is represented on the primary ring by one or more nodes. Our system works from the primary ring, building a keyword ring through selection of nodes from the physical resource node ring. We can therefore choose high quality nodes for keyword rings in addition to high quality super ring nodes.

6. Future Work

We are in the process of gathering data and implementing simulations that will allow us to verify the correctness of our calculations of fault tolerance for different sizes of rings. This will allow us to establish whether both the increase in mean path length and v_i shown provide good approximations for all rings, or whether the constants within them need to be varied with respect to network size.

In the long term, we would like to develop a network capable of dynamic adaptation of topology, based on the probability of node failure. For example, where p is small, performance becomes more important than fault tolerance. If p is large, fault tolerance becomes the more important factor. As mentioned in section 2, while DHTs have high resilience to failure, flooding protocols such as Gnutella have high fault tolerance that could only be bettered by a fully connected mesh.

7. Conclusions

We have proposed a resource discovery system for use in peer-to-peer networks based on an arrangement of multiple Chord rings. We have demonstrated the system's performance and fault tolerance, showing there is a trade-off between the two factors. For example, a smaller ring leads to better performance whereas a larger ring increases fault tolerance.

Our system performs favourably in terms of messages required to route a query compared with Chord and other multi-layered ring topologies such as a Chord instantiation of Hierarchical Peer-to-Peer Systems, but has lower fault tolerance than a large Chord ring.

However, it is clear that Lazy Ring Creation and utilising shortcuts which allow the super ring to be bypassed for repeated-keyword queries can improve both performance and fault tolerance. Lazy Ring Creation selects high quality nodes and builds smaller rings, lowering probability of failure whilst allowing queries to be routed in fewer hops. Shortcuts allow the super ring to be bypassed, again routing queries in less hops thus avoiding more nodes which may have failed.

8. Acknowledgement

The financial support of the UK Engineering and Physical Sciences Research Council (EPSRC) (for JS) is gratefully acknowledged.

9. References

[1] N. Antonopoulos and J. Salter, "Improving query routing efficiency in peer-to-peer networks", *Computing Sciences Report CS-04-01*, University of Surrey, UK, March 2004.

[2] M.J. Freedman and D. Mazières, "Sloppy hashing and self-organising clusters", In *2nd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, Berkeley, CA, February 20-21, 2003.

[3] L. Garcés-Erice, E.W. Biersack, K.W. Ross, P.A. Felber, and G. Urvoy-Keller, "Hierarchical peer-to-peer systems", In *Parallel Processing Letters*, Vol 13, No 4, December 2003, pp. 643-657.

[4] K. Gummadi, R.Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The Impact of DHT Routing Geometry on Resilience and Proximity", In *2003 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer* Communications, Karlsruhe, Germany, August 25-29, 2003, pp.381-394.

[5] D.R. Karger and M. Ruhl, "Diminished Chord: A protocol for heterogeneous subgroup formation in peer-to-peer networks", In *3rd International Workshop on Peer-to-Peer Systems (IPTPS 04)*, San Diego, CA, USA, February 26-27, 2004.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network", In *ACM SIGCOMM 2001*, San Diego, CA, USA, August 27-31, 2001, pp. 161-172.

[7] M. Ripeanu, "Peer-to-peer case study: Gnutella network", *University of Chicago Technical Report TR-2001-26*, University of Chicago, USA, 2001.

[8] J. Ritter, "Why Gnutella can't scale", *http://www.darkridge.com/~jpr5/doc/gnutella.html*, 2001.

[9] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralised object location and routing for large-scale peer-to-peer systems", In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 12-16, 2001, pp. 329-350.

[10] S. Saroiu, K. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems", In *Proceedings of Multimedia Conferencing and Networking*, San Jose, January 2002.

[11] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications", In *IEEE/ACM Transactions on Networking*, Vol 11, No 1, February 2003, pp.17-32.

[12] P. Triantafillou, "PLANES: The next step", In *Peer-to-Peer Network Architectures, SIGCOMM Workshop on Future Directions in Network Architectures (FDNA-03)*, Karlsruhe, Germany, August 27, 2003.

[13] Z. Xu, R. Min and Y. Hu, "HIERAS: A DHT based hierarchical P2P routing algorithm", In *2003 International Conference on Parallel Processing*, Kaohsiung, Taiwan, October 6-9, 2003.
[14] B.Y. Zhao, Y. Duan, L. Huang, A.D. Joseph, and J.D. Kubiatowicz, "Brocade: Landmark routing on overlay networks", In *1st International Workshop on Peer-to-Peer Systems (IPTPS 02)*, Cambridge, MA, USA, March 7-8, 2002.
[15] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing", *Technical Report UCB/CSD-01-1141*, University of California Berkeley, CA, USA, 2001.