AChord: Topology-Aware Chord in Anycast-Enabled Networks

Le Hai Dao and JongWon Kim
Networked Media Lab., Department Information and Communications,
Gwangju Institute of Science and Technology (GIST), Gwangju, Korea
E-mail : {daolehai, jongwon}@gist.ac.kr

Abstract

Chord is a scalable peer-to-peer (P2P) lookup protocol that solves the problem of efficiently locating nodes with designed data items. However, Chord is known to be limited in routing efficiency since all nodes are placed on the overlay without taking into account the underlying network topology. In this paper, we introduce AChord, a topology-aware version of Chord that utilizes anycast, along with additional modifications, to achieve higher routing efficiency. Simutation results show that the proposed approach achieves better performance than the original Chord in the sense of end-to-end lookup latency and average path length.

1. Introduction

Distributed Hash Table (DHT)-based peer-to-peer (P2P) overlay networks have drawn much research attention in the past few years. They promise to be substrates for future P2P applications in the Internet because of their salient features of scalability and resilience. One of the most popular DHT-based systems is Chord [1]. Although this overlay system can guarantee that any data object can be located in *O(logN)* overlay hops on the average, where *N* is the number of peers in the system, the underlying path between two peers can be significantly different from the path on the overlay [9]. This is because the construction of a P2P network in Chord does not take into account underlying network topology. Therefore, the end-to-end latency of the path can be quite large even though the number of application-level hops is small.

Anycast is a new "one-to-nearest-one-of-many" communication method in IP networks [2]. It promises several important applications and will become indispensable in the future Internet. Anycast has been included explicitly in the IPv6 protocol, so compliant IPv6 implementations must support anycast. It can be assumed, then, that when IPv6 is fully deployed, anycast will be readily available. IPv6 is the next generation IP protocol that promises to solve the shortcomings of current IP protocol, IPv4, and its deployment has significantly increased in the past few years. There is also an increased interest and research surrounding retrofitting anycast to IPv4, and this warrants an early deployment. Thus, we expect the availability of a global anycast-enabled network in the near future.

In this paper, we apply anycast to address the above problem of Chord and help nodes in the overlay network to be more aware of underlying network topology, thus achieving a better routing efficiency. Our approach can also be applied to other DHT systems with little modification. This new version of Chord is named AChord, meaning “topology-aware Chord running on anycast-enabled networks”. Since anycast is designed to deliver a packet to the nearest one of many hosts, this kind of network can be used to help a new node discover an overlay network in which all nodes are in one anycast group.

The major difference in AChord is that the identifier of a node is computed based on the relative locality information of this node to the others but not by hashing the IP address as in Chord. By this mechanism, the positions of nodes on the overlay are associated with the underlying network topology. However, only assigning nodes in the overlay network closely to underlying topology is not enough, since we cannot appropriately map the *n*-dimensional space of a real network into the one-dimensional identifier space of Chord. For this reason, nodes need additional information to understand more about their surroundings. In our approach, each node maintains one more table, called a ‘neighborship table’, to store information about its closest known nodes. The routing decision will be made by both neighborship table and finger table. The neighborship table is first built by adding the nearest node and its selected neighbors when a new node joins the overlay network. This table is updated many times during the lifetime of the node:

- By exchanging its neighborship table with its neighbors, and
- When it is contacted by a new joining node.

AChord has several advantages over current topo·

2006 International Conference on Hybrid Information Technology (ICHIT'06)
0-7695-2674-8/06 $20.00 © 2006 IEEE

logy-aware solutions for Chord such as ChordPNS (proximity neighbor selection) [1], PChord [3], Chord6 [4]:

- It is simple and lightweight. New joining nodes just have to go through fewer steps initially to join the network.
- There is no need to arrange any privileged nodes to help new joining nodes. Thus, the overlay network is flat and can avoid centralized points of failure.
- It achieves high accuracy of network proximity giving high routing efficiency.
- There is little change to Chord and its advantages are kept unchanged.

The rest of this paper is structured as follows. Section 2 discusses related works. Section 3 presents our design. Section 4 evaluates AChord including qualitative analysis, simulation results, remaining issues and solutions. The last section, Section 5, gives concluding remarks.

2. Related Work

Chord [1] has many salient features over other P2P systems: it is simple and robust; it has guaranteed retrieval, low path length and efficient recovery schemes. However, this system is well-known for the lack of topology-awareness since nodes get their identifiers by hashing IP addresses. For this reason, there has been significant research aimed at improving the routing efficiency of Chord.

In [1], the authors of Chord also proposed a method called Proximity Neighbor Selection (PNS). The idea is to maintain a set of alternative nodes for each finger and then route the lookup query by selecting the closest node among the alternative nodes. With this method, the lookup stretch is significantly reduced; however, in the process of finding the closest fingers and maintaining the routing table, ChordPNS creates a heavy load to the network and more overheads to nodes.

PChord [3] achieves better network proximity than the original Chord by using proximity lists, i.e. the list of the close nodes that a node discovers in its lifetime. The next hop is decided by the entries in both the proximity list and finger table. Although this approach achieves better routing efficiency than Chord while keeping lightweight maintenance costs, it has problems of slow convergence and inefficiency in the case of churn, where the lifetime of a node in the overlay is relatively short.

Chord6 [4] utilizes the hierarchical structure of IPv6 addresses to produce a node's ID so that the nodes in the same domain have close IDs, but nodes in two close domains may be far away in the overlay network. This approach is quite simple and can reduce the average path length from *O(logN)* to *O(logM)* where *N* is the number of nodes and *M* is the number of domains. However, because of the large number of domains in the Internet and the number of overlay nodes in the same domain is small, this approach doesn't seem efficient.

There has been some successful research on using anycast to improve current Internet services. Agarwal [5] proposed scalable, incrementally deployable architecture, Content Distribution Architecture using Anycast (CDAA), which provides a network-layer support for anycast to solve the problem of selecting the "best" server among several replicated servers. For building efficient shared multicast trees, Katabi [6] introduced the Anycast-Based Tree (ABT) which does not suffer from the traditional problems exhibited by core-based trees, such as traffic concentration or poor core placement. Moreover, ABT exhibits greater robustness and lower bandwidth consumption than other shared multicast trees.

Anycast can also be used to improve other P2P overlay systems and we believe that this could be the new research direction for the future of the Internet.

3. AChord: Proposed Topology-Aware Chord

3.1 Topology-Awareness Problem of Chord

Chord [1] assigns keys to its nodes by using the consistent hash function. All nodes are arranged in an one-dimensional circular space based on their keys which are created by hashing nodes' IP addresses. The data is assigned to a node whose identifier closely follows the key. This decentralization tends to balance the load on the system, since each node receives roughly the same num-ber of keys, and there is little movement of keys when nodes join and leave the system. For routing purposes, each node maintains a list of nodes whose keys follow this node's ID in the key space, called a successor list, and a list of other *O(logN)* far nodes, called a finger table. When a node needs to lookup a key, it checks all the lists and finds out which node will be the next hop. The selected destination node is the node that has the closest ID preceding the data key. The lookup query is then forwarded in this manner hop-by-hop until it reaches the node that stores the key. The number of hops that a lookup query should pass is called routing path length. Chord achieves a routing path length of

O(logN) hops in the average.

The two lookup styles in the Chord protocol are iterative and recursive. In the iterative style, the node that wants to do the lookup makes direct connections to all of the next-hop nodes to the destination. It asks a series of nodes for their finger tables, starting from the closest node to the lookup key in its finger table. After each step, it moves closer to the designed successor of the lookup key. In the second style, the recursive style, the lookup request is forwarded hop-by-hop so that each next hop is decided by the intermediate node until it reaches the destination. The destination node then replies by sending the lookup result directly to the lookup node. Experimental results in [1] show that the recursive style achieves better performance than the iterative style. For this reason, in this paper we assume the recursive style of Chord.

Although Chord has a small average path length, it has a significant problem of topology-awareness. The main reason comes from the mechanism which assigns nodes in the overlay network. Since nodes get their ID by hashing IP addresses, then nearby nodes in the overlay (i.e., nodes with close IDs) can be geographically far away from each other. Thus, although the number of application-level hops is small, the end-to-end latency could be significantly large. To solve this problem, there should be a method that incorporates topology information to nodes' IDs. This is a difficult task for the following reasons. First, we need to have a mechanism to help nodes understand the underlying network. Second, we need an algorithm to map that information to nodes' IDs. Finally, the use of underlying topology to build the overlay may sacrifice the uniform key distribution, so we also need to consider this issue.

3.2. Proposed Approach

In our proposal, AChord, we use anycast(*) as a mechanism to help nodes understand the underlying network. The main feature of anycast is that a message from outside which targets an anycast address would be delivered to the nearest node in that anycast group, and we exploit this for doing node initialization. The key idea is: if all nodes in an overlay network are in the same anycast group, and the source of the outside message is a node that wants to join the overlay, then we can assign this node a suitable position without the need of bootstrapping nodes. To help nodes understand better about the underlying network, each node will maintain an additional routing table, called a neighborship table, to store the updated information about its surroundings.

The algorithm to assign nodes' IDs is presented in the next part of this section. The remaining issue of node distribution and solutions will be discussed in Section 4.3.

3.3. Construction and Operation of AChord

3.3.1. Construction mechanism

Fig. 2 shows the algorithm for constructing the network with some modification of the original Chord. There are two parts: creating a new overlay network and joining the network. The former is done by the first node in the overlay. This node has ID *0* and it also creates a new anycast group.

Fig. 1 illustrates the join procedure with: 1) When a new node, say node *i*, wants to join a P2P network, it first sends a joining request to the anycast address of this overlay network. The underlying network will route this message to the nearest node in the overlay, say node *j*, and 2) node *j* replies to node *i* with its ID and also the ID and IP address of its predecessor, say node *p*. 3) From this information, node *i* computes its own ID so that its position will be between node *j* and node *p*. Node *i* also takes node *j* and node *p* as its immediate successor and predecessor respectively and notifies node *p* about this change. 4) Node *i* then joins the anycast group of this overlay network.

After joining the overlay network, the new node will build its neighborship table as well as its finger table. The finger table is built in the same way as the original Chord. The neighborship table is built by first adding the nearest node. The new node then requests this node's neighborship table and measures distances to all nodes in this received table. All of those nodes are also notified to see if they can add this new node to their neighborship table. The new node only adds up to *k-1* closest nodes among those nodes, where *k* is the maximum number of neighborship entries. By this mechanism, the neighborship table is updated many times in a node's lifetime. A node also requests its neighbors' neighborship tables to have a better view of the surrounding network. Fig.5 illustrates how a node builds its neighborship table.

(*) *Anycast here means the ideal anycast that always routes packets to the nearest host in the underlying network; other cases will be considered in future work. (See Section 6)*

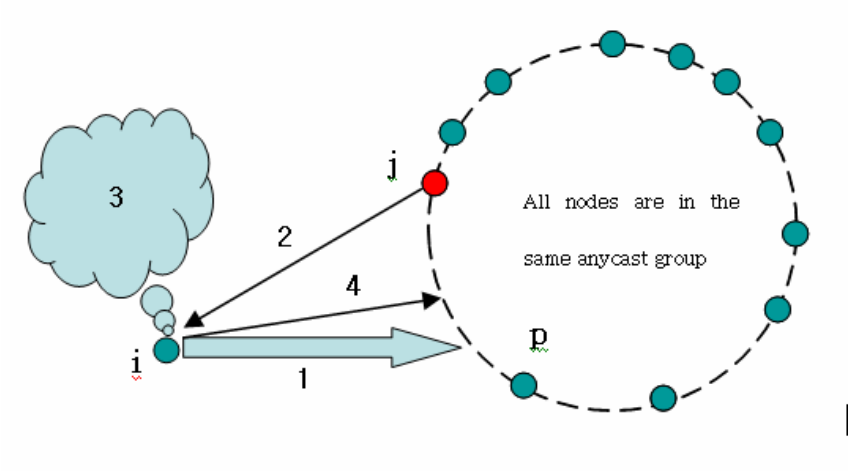

Figure 1. Join procedure in AChord.

```
//create a new Chord ring
// and a new anycast group
n.create()
  n.id = 0;  //the first node has ID = 0
  predecessor = nil;
  successor = n;
  g = new anycast_group;
  anycast_join(g,n);

// join a Chord ring
n.join()
  anycast_send_to(g);
  //receives reply from nearest node
  //containing node's information and
  //predecessor
  info = rcv_from(g);
  nearest = info.node;
  n.ID = between
  (nearest.ID,nearest.predecessor.ID);
  n.successor = nearest;
  if (info.predecessor = nil)
    predecessor = nearest;
  else n.predecessor = info.predecessor;
  anycast_join(g,n);
```

Figure 2. Pseudo code for initialization.

```
//build the neighborship table (nbt)
n.build_nbt ()
  if rcv_nbt.size < k
    n.nbt = rcv_nbt;
  else
    rcv_nbt.remove(furthest_entry);
  //measure distances and notify
  // update to neighbors
  for i = nbt.size downto 1
    nbt[i].distance = distance(n,nbt[i]);
    notify_update(nbt[i]);
  //add the nearest node to nbt
  n.nbt_add(nearest);
```

Figure 3. Pseudocode for building neighborship table.

```
//search the routing table for the highest
//predecessor of ID
n.closest_preceding_node(ID)
  for i = finger.size downto 1
    if (finger[i]∈ (n,ID)
      temp1 = finger[i];
  //searching in neighborship table
  for j = nbt.size downto 1
   if (nbt[i]∈ (n,ID)
     temp2 = nbt[i];
  return max(temp1,temp2);
```

Figure 4. Pseudocode for routing in AChord.

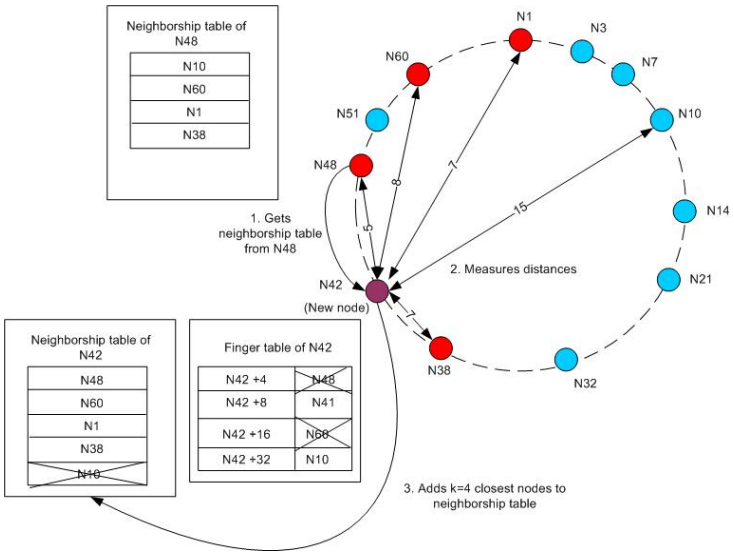

Figure 5. Building neighborship table.

3.2.2 Operation

Fig. 6 shows an example of routing in AChord using the algorithm in Fig. 4. The routing decision is made by using both the neighborship and finger table with the same rule as the original Chord. The closest node to the target *key* in key space will be found as the next hop of the lookup query. The neighborship table helps nodes to route messages to close destinations, while the finger table helps nodes to route messages to wider destinations. If there is an overlap entry in the two tables, the entry in the finger table will be set as inactive and when this entry is replaced in the neighborship table, it is then reset to active mode. Because of this rule, the finger table is reduced to less than m - $logk$ fingers, where m is the number of bits in the key/node identifiers, and k is the maximum number of entries in the neighborship table. To maintain the whole routing table, heart-beat messages are also used as in the original Chord to watch if the entry is still valid.

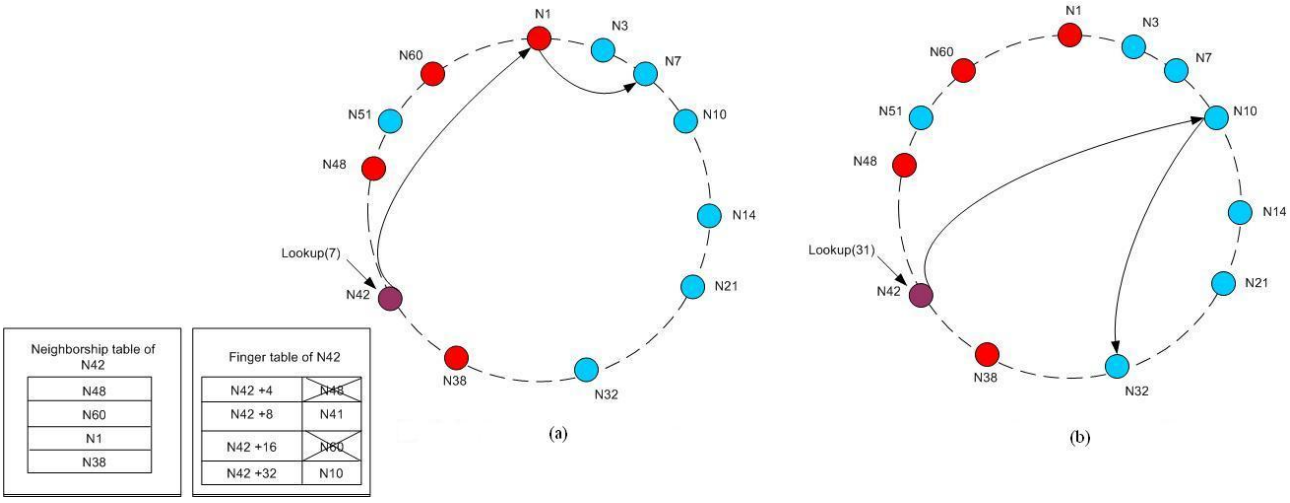

Figure 6. Routing in AChord. The lookup request is forwarded to the next hop based on (a) neighborship table or (b) finger table.

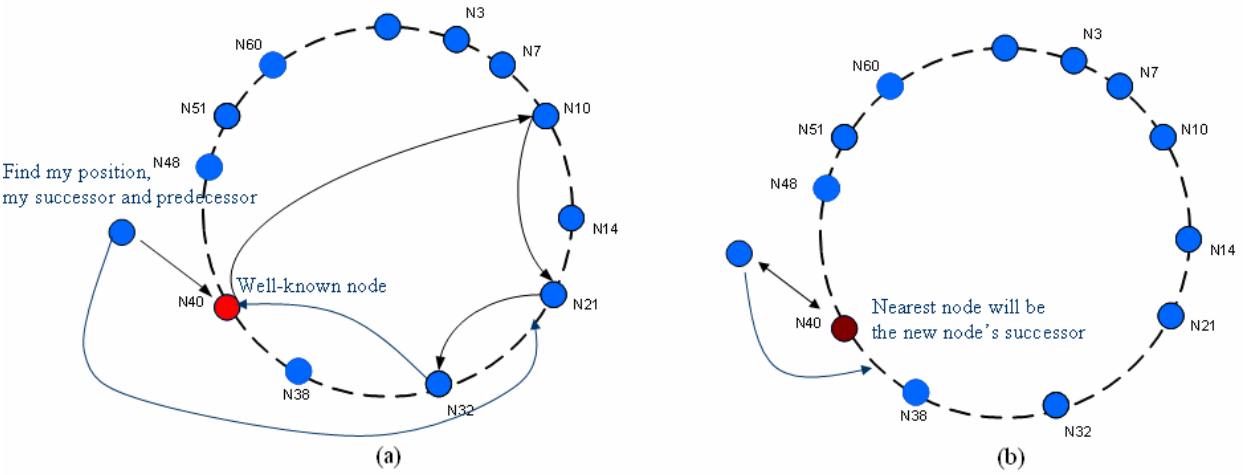

Figure 7. (a) The joining procedure of the original Chord has the problem of single points of failure. (b) AChord has a more lightweight joining procedure and distributes the load to all nodes.

Table 1. Comparison of current topology-aware versions of Chord

	Topology-awareness	Maintenance cost	Join overhead	Load-balancing	Routing efficiency
Chord	N/A	Low	High	No	Low
ChordPNS	Medium	High	High	No	Medium
PChord	Medium	High	High	No	Medium
Chord6	Low	Low	High	No	Low
AChord	High	Medium	Low	Yes	High

4. Evaluation

4.1. Comparative Analysis

Table 1 gives a general comparison of current approaches that aim to solve the topology-aware problem of Chord. Here, we discuss three major advantages of AChord over the others

- **Load-balancing:** All nodes in the overlay network can be contacted by a new joining node and thus avoiding the "single point of failure" problem caused by using some well-known nodes for bootstrapping.
- **Low joining overhead:** As shown in Fig. 7, a new node in AChord doesn't need a long procedure to find its position as well as its successor and predecessor, as in the original Chord. Simply, the nearest node will be chosen as its immediate successor. The only joining overhead in AChord is the simple ID computation that the new node needs to do, based on the nearest IDs of node and the predecessor.
- **Lightweight routing table:** Each node in AChord only needs to maintain a reduced finger table and a small fixed-sized neighborship table, thus having a lightweight maintenance cost.

4.2. Simulation Results

We simulate Chord, ChordPNS and AChord in a discrete-event packet simulator, p2psim [7]. The simulated network consists of 1740 nodes with the pairwise latency matrix derived from measuring the internode latencies of 1740 DNS servers using the King method [8]. The simulation was run in the churn environment where nodes join and leave randomly. The node which had left then rejoined the dynamic network, so the equivalent total of nodes is much more than 1740.

To evaluate our design, AChord, we use two metrics. First, we use average Relative Delay Penalty (RDP): the ratio of the total latency a query travels through the overlay network to reach an object and the minimal latency to that object (the latency of the direct path between the requesting node and the node that contains the object), as a metric to measure the routing efficiency. Second, we use the average path length which is the number of nodes that must be visited to solve a query.

Table 2. Average RDP and average path length comparison.

Chord version	Average path length	Average RDP
Original Chord	5.2073	5.0112
ChordPNS (with 8 fingers' successors)	4.8211	2.3253
AChord (with 8 neighborship entries)	3.7296	1.8325

Table 2 shows the comparison of AChord to Chord and ChordPNS. Note that with 8 fingers' successors, each node needs to take care of approximately *m*8* other nodes for building its finger table, while for AChord with 8 neighborship entries, each node needs to take care of less than *m+8* nodes for the same purpose.

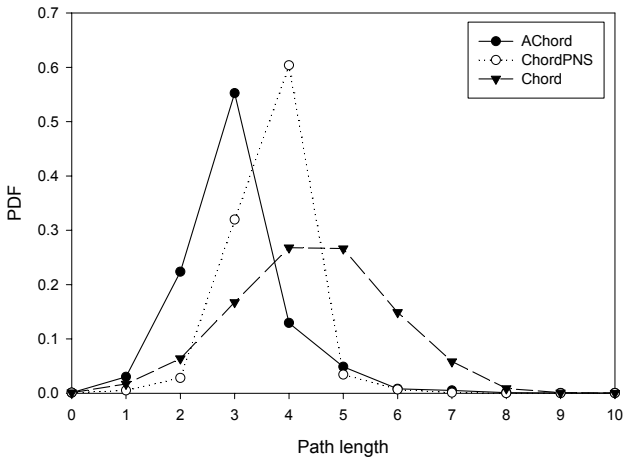

Fig. 8. PDF of the path length.

One of the most salient features of Chord is that it can achieve quite low path lengths, *O(logN)*. However, Fig. 8 shows that AChord can achieve a better performance with 28% lower path length.

4.3. Remaining Issues and Solutions

AChord achieves much better routing efficiency than Chord. However, since nodes' IDs are computed based on underlying network information, this method can have unequal probability of distributing nodes in identifier space and thus makes the number of data keys per node un-uniform. The data key space refers to the space of the node ID and its predecessor's ID. To deal with this possible problem, we apply two solutions. One is to utilize the load-balancing ability of anycast, and the other is to self-organize the system.

- **Load-balancing using anycast:** Anycast can be used to distribute load by directing a join query to a close but less contacted node in the overlay network. A less contacted node means that the maintaining data key space of this node may be larger than the node that has received more join requests. By applying load-balancing for admission control, a new joining node may not be in contact with its actual nearest node but in the long term, since its neighborship table is updated several times, this node can have the optimal routing table.
- **System self-organizing:** From a node's local view in the overlay network, it is aware of its own data key space and its neighbors' key space. Nodes can gather this information in the time they exchange periodic heart-beat messages. If a node receives a join request and its key space is small while some of its neighbor has larger key space, then this node will inform the new joining node about the best node to contact.

6. Conclusion and Future Work

We have proposed a new method to overcome one of the weaknesses of Chord, topology-awareness. The key idea is to use the property of anycast, where messages from outside targets to an anycast address will be delivered to the nearest node in this anycast group, and apply for the join procedure of all new nodes. Our solution is simple, efficient and can be applied to both Chord and other P2P overlay systems. This is also one of our future targets.

In this paper, we assume that underlying networks support an ideal anycast, i.e., the outside messages are always routed to the nearest node. However, in the real Internet, this is not always true, so in the future we will evaluate this case to understand its effects to our system.

7. Acknowledgement

This research is supported by Korea Research Foundation (KRF-2004-041-D00463).

8. References

[1] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. "Chord: A scalable peer-to-peer lookup service for internet," *IEEE/ACM Transactions on Networking,* Vol.11, No.1, February 2003.

[2] S. Weber and L. Cheng, "A survey of anycast in IPv6

network," *IEEE Communications Magazine.* vol. 42. no. 1. Jan. 2004.

[3] F. Hong, M. Li, J. Yu, and Y. Wang, "PChord: Improvement on chord to achieve better routing efficiency by exploiting proximity," in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops* (ICDCSW'05), June 2005.

[4] J. Xiong, Y. Zhang, P. Hong, and J. Li, "Chord6: IPv6 based topology-aware Chord," in *Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services* (ICAS/ICNS 2005), Aug 2005

[5] G. Agarwal, R. Shah, and J. Walrand, "Content distribution architecture using network layer anycast," in *Proceedings of The Second IEEE Workshop on Internet Applications,* July 2001.

[6] D. Katabi, "The use of IP-anycast for building efficient multicast trees," *Proc. IEEE Globecomm*, Costa Rica, December 1999.

[7] p2psim. http://pdos.csail.mit.edu/p2psim/.

[8] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "KING: Estimating latency between arbitrary Internet end hosts," *in Proceedings of SIGCOMM Internet Measurement Workshop,* Nov. 2002.

[9] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Survey and Tutorial*, 2004.