PAPER

A Proximity-Based Self-Organizing Hierarchical Overlay Framework for Distributed Hash Tables

Kwangwook SHIN[†a)], ***Student Member***, **Seunghak LEE**[†], **Geunhwi LIM**[††], ***and*** **Hyunsoo YOON**[†], ***Nonmembers***

SUMMARY Several structured peer-to-peer networks have been created to solve the scalability problem of previous peer-to-peer systems such as Gnutella and Napster. These peer-to-peer networks which support distributed hash table functionality construct a sort of structured overlay network, which can cause a topology mismatch between the overlay and the underlying physical network. To solve this mismatch problem, we propose a topology-aware hierarchical overlay framework for DHTs. The hierarchical approach for the overlay is based on the concept that the underlying global Internet is also a hierarchical architecture, that is, a network of networks. This hierarchical approach for the overlay puts forth two benefits: finding data in a physically near place with a high probability, and smaller lookup time. Our hierarchical overlay framework is different from other hierarchical architecture systems in a sense that it provides a specific self-organizing grouping algorithm. Our additional optimization schemes complete the basic algorithm which constructs a hierarchical structure without any central control.

key words: *peer-to-peer, DHT, physical topology, hierarchical architecture, overlay network, efficient replication*

1. Introduction

Peer-to-peer networks can be characterized as self-organized dynamic server set (which also plays the role of the client). In the peer-to-peer networks, the node can join or leave the networks at any time and there is no control of the central coordinator. Several peer-to-peer lookup networks which support distributed hash table functionality have been created to solve the scalability problems of previous peer-to-peer systems such as Gnutella and Napster [1]–[3]. In the DHT (Distributed Hash Table) systems, one stores data in the node determined by the hash function and the other looks up the node storing data with the same hash function. Each node in the overlay network has the partial location information for logical routing. The partial location information is well distributed, so that one can be closer and closer to the destination on every logical hop. These peer-to-peer networks which support distributed hash table functionality construct a sort of structured overlay network, which can cause a topology mismatch between the overlay and the underlying physical network. The physical distance between the node looking for some data and the node storing this data may be long. In addition, the lookup time can also be large because the system does not consider the physical topology when it stores or replicates data.

Manuscript received September 7, 2006.
Manuscript revised January 5, 2007.
[†]The authors are with EECS, KAIST, Korea.
[††]The author is with Samsung Electronics Co., Korea.
a) E-mail: kwshin@nslab.kaist.ac.kr
DOI: 10.1093/ietcom/e90–b.7.1651

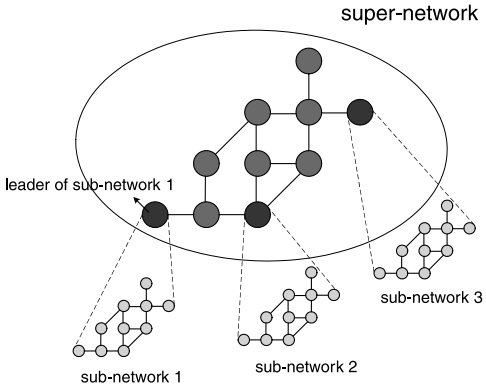


Fig. 1 The hierarchical structure of Grapes.

We propose a hierarchical overlay framework, which is named Grapes [4], for DHTs. Grapes constructs the hierarchical structure using physical topology information. Grapes has a two-layer hierarchical structure, sub-network, and super-network as shown in Fig. 1. In Grapes, nodes physically near each other construct the sub-network, and the leaders of each sub-network form the super-network. This hierarchical approach puts forth two benefits. First, a node can find data in its sub-network with a high probability due to the replicated data publication (data replication) in its sub-network. Grapes supports the efficient data replication considering the physical topology. Second, the hierarchical structure with the physical information makes the lookup time smaller. There are some other schemes to provide hierarchical concepts for peer-to-peer networks. Our hierarchical overlay framework is different from other hierarchical architecture systems in a sense that it provides a specific self-organizing grouping algorithm. To the best of our knowledge, Grapes is the only unique topology-aware hierarchical overlay network which supports the original peer-to-peer concept of self-organization.

The rest of this paper is structured as follows. We discuss the related work in Sect. 2. Sections 3 and 4 describe the basic design of Grapes and the optimization methods for the Grapes. Section 5 demonstrates the performance evaluation by simulation. Finally, we conclude the paper in Sect. 6.

2. Related Work

2.1 DHT (Distributed Hash Tables) Systems

CAN [5], Chord [6], Pastry [7] and Tapestry [8] are repre-

sentative DHT-based peer-to-peer lookup services. In CAN [5], the problem due to lack of physical distance information is compensated by two heuristic methods. First, each node measures its network delay to a set of landmarks and orders the landmarks in order of increasing RTT. Nodes with the same orderings are inserted into the same portion of the coordinate space, which makes logical neighbor physically close to each other [5], [9]. This heuristic is categorized into the Proximity Identifier Selection (PIS) by [10]. Second, the CAN node can forward a message to its neighbor with the maximum ratio of progress to RTT. This heuristic is categorized into the Proximity Route Selection (PRS) by [10]. Original Chord does not consider physical distance information, but enhanced scheme [11], [12] uses PRS. There are potentially several possible next hop neighbors that are closer to the message's key in the id space. PRS is to select, among the possible next hops, the one that is closest in the physical network or one that represents a good compromise between progress in the id space and proximity. In Tapestry [8] and Pastry [7], when a new node is inserted to the overlay network, it finds the bootstrap node close to it in physical distance by expanded ring search or out of band communication. Also, they can choose physically closer neighbors among the neighbor candidates when they construct the routing tables. This is categorized into Proximity Neighbor Selection (PNS) by [10].

However, the objective of all the above optimization methods (except Tapestry) considering physical topology is only to make the immediate neighbors in the overlay network close on the Internet. The physical distance between the node looking for some data and the node storing this data may still be long. It is because the above methods do not provide the framework for the efficient data replication with the physical information. Tapestry provides some kind of data replication scheme considering physical topology. When the Tapestry node publishes an object, the node stores its pointer in the root node of this object. The pointer is also stored in the nodes on the path from the publishing node to the root. When a node finds the object, it locates the object by routing a message to the root. Each node on the path checks whether it has a pointer for the object. If so, it redirects the message to the node of the pointer. If there are replicas of an object on various publishing node, the node locates the object in the logically closer node. In Tapestry, since the overlay is structured with PNS, the logical neighbor is somewhat physically close. This makes the possibility that the node finds the object in the physically closer node among the replicas higher. However, Tapestry does not consider hierarchical architecture overlay. To harmonize the virtual overlay with Internet physical topology, we think, the hierarchical approach is one of the best solution because global scale Internet is composed of various localized networks, which forms a hierarchical architecture.

2.2 Hierarchical Architecture Systems

Brocade [13] proposed a similar hierarchical architecture based on physical topology. Brocade constructs a secondary overlay to be layered on top of peer-to-peer lookup systems. The secondary overlay builds a location layer between the super-nodes. The nodes looking for some data which is stored in the long distant node, find a local super-node at first. Then the super-node determines the network domain of the destination and routes directly to that domain, which brings the shorter lookup time. A super-node is determined by the election algorithm between nodes or by the system administrator. Gateway routers or machines close to the routers are attractive candidates for super-nodes. Brocade does not provide any specific algorithm to determine whether the joining node is the super-node or the normal node. [13] mentioned that Brocade uses a kind of election algorithm to decide the super-node in an ISP, but no specific algorithm has been proposed. Neither does Brocade suggest any process to organize super-node's cover set with normal nodes in an ISP. Grapes suggests a specific self-organizing algorithm which constructs the hierarchical structure. In Grapes, any node can be a leader or sub-node depending upon the order of node insertion, and the nodes construct the hierarchical overlay network in an autonomous manner without any support of central coordinator. This kind of self-organization makes Grapes more suitable for pure peer-to-peer concepts.

L. Garces-Erce et al. [14] provides a general framework for a hierarchical DHT. The peers are organized into groups. A group may consist of the peers topologically close to each other. Each group is required to have one or more superpeers which play the role of gateways between the groups. The peer looking for some data sends a query message to one of its superpeers. Then the message is delivered to the group that stores the data by top-level lookup service. The superpeer of the group that stores the data, routes the query message to the peer that is responsible for the data with intra-group routing. In this system, the peer joining the network must know the identifier of the group that the peer belongs to. The identifier may be the name of the peer's ISP or university campus. The peer looks up the group id in the peer-to-peer network, and the destination gives the IP address of the superpeer which the peer will join. In order to get well-constructed hierarchy, we think, this system needs the help of a central coordinator. Without the help of a central coordinator, the peers having the same ISP name but physically not close enough to each other may join the same group (e.g. tier-1 ISP). Also, the group of the peers in a university campus can be too small. Too small groups may downgrade the system performance like data lookup time. Grapes tries to make well-constructed hierarchy without any centralized control.

3. Design of Grapes

3.1 Node Insertion

Any new node joining Grapes must know the contact point of at least one Grapes node, called the bootstrap node. When

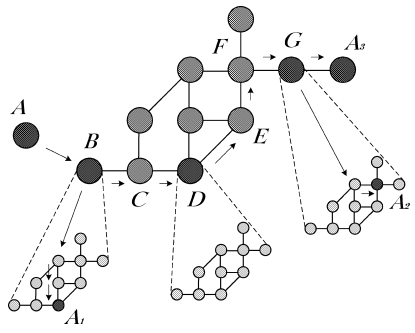


Fig. 2 Node insertion example in Grapes.

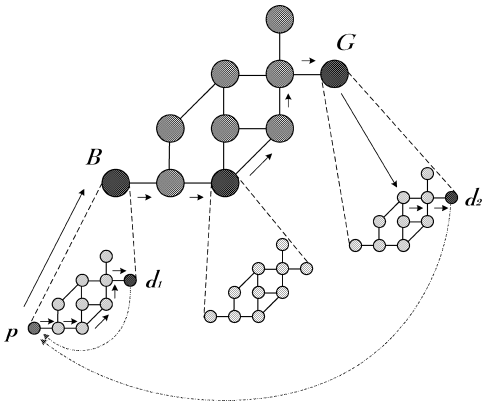


Fig. 3 Data publication example in Grapes.

a new node is inserted into Grapes, the new node checks its physical distance[†] to the bootstrap node. If the physical distance is shorter than the given threshold, the new node is inserted into the sub-network of the bootstrap node. If the bootstrap node is the leader of the sub-network, the bootstrap node will be the leader of the new node. Otherwise, the bootstrap node notifies the new node of its leader. If the physical distance between the new node and the bootstrap node is longer than the threshold, the new node is inserted into the super-network. The new node checks its physical distance to the leaders on the route[††] in the super-network. If it finds the leader to which the physical distance is shorter than the threshold, it is inserted into the sub-network of the leader. If there is no leader into which the new node is inserted on the route, the new node is inserted as a node (leader) into the super-network. In each layer of the overlay network, any peer-to-peer lookup routing algorithm such as CAN [5], Chord [6], Pastry [7], and Tapestry [8] can be used.

In Fig. 2, the new node A joins Grapes with the help of the bootstrap node B. If the distance between A and B is shorter than the threshold, A is located in B's sub-network (A_1). Otherwise, A is not inserted into B's sub-network, but to B's super-network. A checks the physical distance to the leaders on the route in the super-network. In the figure, A is not inserted into C (and D, E, F)'s sub-network because the distance between A and C (and D, E, F) is longer than the threshold. If the distance between A and G is shorter than the threshold, A is inserted into G's sub-network (A_2). If there is no appropriate leader on the route, A is inserted as a leader into the super-network (A_3).

3.2 Data Publication

When a node publishes data, it looks for the node managing the hashed key of the data in its sub-network first. The node inserts the index of data into the node which manages the key of the data, and then the node inserts the index into the super-network. The node, with the help of its leader, looks for the leader managing the key in the super-network. If the destination leader does not have its own sub-network, the index is inserted into the destination leader. If the destination leader has its sub-network, the node looks for the node managing the key in the sub-network. In this case, finally, the index is inserted into the destination node in the sub-network. The index of the data is a kind of metadata, one can get the list of the addresses of the data publication nodes from the index (The index can point to more than one data location when several nodes publish the same data). Data publication node periodically re-publishes the data, which refreshes the index timer of the data or inserts the index if the new destination node does not have the index. Every index has its own timer and when the timer expires the index is invalidated and removed. This periodic data publication makes Grapes fault-tolerant in a dynamic peer-to-peer environment.

In Fig. 3, the node p is publishing the data into the network. Firstly, the node p looks for the node managing the key of the data in its sub-network. In this example, the node d_1 is managing the key of the data. The node p inserts the index of data to the node d_1 (the index points to the publishing node p). And then the node p looks for the node managing the key of the data in the super-network through its leader B. The lookup request is routed from B to G, and G forwards the request to its sub-network. Finally the node p becomes aware of the node d_2, which is managing the key of the data in the super-network. The node p inserts the index to the node d_2 (this index also points to the publishing node p).

3.3 Data Retrieval

When a node retrieves data, it looks for the node managing the hashed key of the data in its sub-network first. If the sub-network does not have the index of data, the node looks up its super-network with the help of the leader. After the node retrieves the data from the target node to which the index points, it publishes the retrieved data. That is, the new index of data is inserted into both its sub-network and super-network. This data publication after the retrieval results in

[†]King [15], IDMaps [16], GNP [17] etc. suggested the latency estimation method between the arbitrary Internet end hosts. We assume that the physical distance can be obtained by any of the latency estimation methods.

[††]The route is determined by a peer-to-peer lookup routing algorithm e.g. CAN [5], Chord [6], Pastry [7], and Tapestry [8].

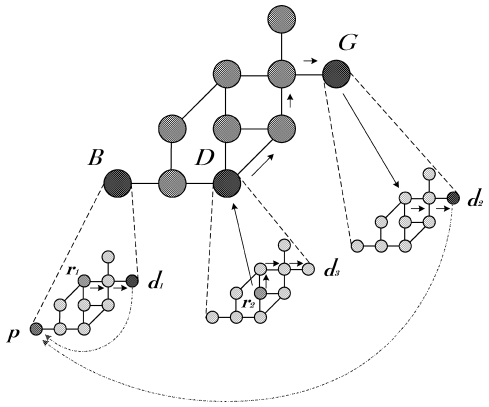


Fig. 4 Data retrieval example in Grapes.

the data replication. If the index points to more than one node, the retrieving node selects the closest one as the target node. This replication makes a node able to find the data in its sub-network with a high probability.

In Fig. 4, the nodes r_1 and r_2 are retrieving the data from the network. When the node r_1 retrieves the data, r_1 can find the index of data in its sub-network (from the node d_1). At the same time, the node r_1 gets the data from the publishing node p to which d_1 points. When the node r_2 retrieves the data, r_2 firstly looks up its sub-network. But the node d_3 does not have the index. And then the node r_2 looks for the node managing the key of the data in the super-network through its leader D. The lookup request is routed from D to G, and G forwards the request to its sub-network. Finally the node r_2 is aware that the node d_2 has the index. The node r_2 gets the data from the publishing node p to which d_2 points. After the nodes r_1 and r_2 retrieve the data from the target node p, they publish the data (r_1 inserts the index of data into d_1 and d_2; r_2 inserts the index into d_3 and d_2).

3.4 Neighbor Information

In comparison with peer-to-peer lookup services without Grapes framework, the neighbor information in each Grapes node increases only by one (its leader's information or its first sub-node's information). The sub-node maintains a list of its neighbor's information collected from the sub-network and it also maintains its leader's information. The leader maintains the neighbor's information gathered from the super-network and it also maintains the information about one of its sub-nodes. This one of the sub-nodes which joined the sub-network first of all is called the first sub-node.

3.5 Leader Replacement

Every node in the sub-network maintains its leader information. Therefore, a sub-node can notice that its leader is crashed or has left the network without any notification, in the process of node inserting or data publication and retrieval. The sub-node that detects the leader's crash or unexpected leaving, firstly checks the specific logical location in its sub-network whether there is any new leader candidate information. If the specific logical location has the new leader information, the sub-node replaces its leader with the new leader. Otherwise, the sub-node takes over the sub-network as a new leader. The specific logical location also maintains the neighbor information of the leader (the information of the leader's neighbors in the super-network and the information of one of its sub-nodes), which makes the new leader able to take over the old leader's logical position in the super-network[†]. After the new leader took over the sub-network of the old leader, one of the neighbor nodes of the new leader becomes the first sub-node of the sub-network. The new leader stores its information and its neighbor information collected from the super-network in a specific logical location (in its sub-network).

Each leader maintains the neighbor information of the first sub-node in case the first sub-node is crashed or has left the network without any notification. If the first sub-node failed, the leader selects one neighbor of the first sub-node as the new first sub-node.

4. Optimizations

In this section, we propose some additional optimization schemes complementary to the basic algorithm. The additional optimization schemes are necessary because Grapes constructs a hierarchical architecture without any central control. If there is a central coordinator or a system administrator, one can construct the optimal architecture based on the physical proximity because the central coordinator or the system administrator can acquire the global physical topology information. In Grapes, however, it is not easy to construct the optimal hierarchical architecture due to the characteristic of the self organization. In this section, we propose three optimization schemes; *Sub-network partition*, *Sub-network integration*, and *Sub-node migration*. Even if the three optimization schemes do not provide the optimal hierarchical architecture, they help Grapes to have a more efficient hierarchical structure.

4.1 Sub-Network Partition

Sub-network partition prevents the specific sub-network from maintaining too large number of nodes. If the size of the sub-network gets bigger, the average delay between sub-nodes increases, which downgrades the system performance (data lookup time and direct delay for data retrieval). *Sub-network partition* procedure is as follows.

1. The leader (super-node) floods *activate partition* message to the sub-network when it notices that the number

[†]With CAN or Chord, the new leader takes over the old leader's logical position in super-network without considering node ID, whereas with Tapestry or Pastry, we should consider node ID. To prevent node ID collision problem, a new leader newly joins the super-network and locates its logical position with its own node ID.

of its sub-nodes is above the maximum limit[†]. The *activate partition* message has the address of the leader and the new leader candidate node.
2. Receiving the *activate partition* message, each sub-node determines which node is its leader, the old leader or the new leader candidate, depending upon its physical distance between them. The sub-node chooses the closer node between the old leader and the new leader candidate as the leader.

4.2 Sub-Network Integration

Sub-network integration plans to organize better architecture by way of unifying two small sub-networks. *Sub-network integration* procedure is as follows.

1. If a node chances on the super-node to which the physical distance is below the threshold in the process of data publication or retrieval, the node sends *initiate integration* message to its leader[††]. *Initiate integration* message has the address of the new near super-node discovered.
2. If the number of sub-nodes is below the minimum limit[†††], and also if the distance between the leader and the new near super-node is below half the threshold, the leader checks whether the new near super-node can accept integration by sending *check integration* message.
3. The new near super-node checks how many sub-nodes exist in its sub-network. If the summation of the number of sub-nodes in two sub-networks is above the maximum limit, *sub-network integration* process halts with *reject integration* message from the new near super-node.
4. If the new near super-node can accept integration, the leader which has smaller number of sub-nodes between two sub-networks floods *activate integration* message to its own sub-network. *Activate integration* message has the address of the corresponding super-node.
5. All the nodes (including the leader and the sub-nodes) in the smaller sub-network are inserted to the larger sub-network with the help of the leader of the larger sub-network as the bootstrap node.

4.3 Sub-Node Migration

Sub-node migration is the optimization scheme to make the average delay between sub-nodes in the sub-network shorter. *Sub-node migration* procedure is as follows.

1. If a node chances on the super-node to which the physical distance is below half the distance to its leader in the process of data publication or retrieval, the node sends *initiate migration* message to its leader.
2. If the number of sub-nodes is not below the minimum limit, the leader sends *activate migration* message to the sub-node. If the number of sub-nodes is below the minimum limit, *Sub-network integration* step 2 begins.
3. The node is inserted to the new sub-network with the help of the new near super-node as the bootstrap node.

Algorithm 1–7 present the pseudo code of the above three optimization schemes. Algorithm 1–3 and Algorithm 4–7 respectively describe the action of the sub-node and the super-node when the optimization process begins.

Algorithm 1
Action of the sub-node on discovering the new near super-node to which distance is below the threshold T

```
d_S ← the distance to the new near super-node
d_L ← the distance to its leader

if d_S > (1/2) d_L then
    sends initiate integration message to its leader and waits the reply
    from the leader
    if receiving activate integration message then
        bootstrap node ← the new near super-node
        re-inserts itself to the network
    end if
    if receiving no action message then
        RETURN
    end if
else
    {d_S ≤ (1/2) d_L}
    sends initiate migration message to its leader and waits the reply from
    the leader
    if receiving activate migration or activate integration message then
        bootstrap node ← the new near super-node
        re-inserts itself to the network
    end if
    if receiving no action message then
        RETURN
    end if
end if
```

Algorithm 2
Action of the sub-node on receiving *activate integration* message from its leader

```
{its leader receives check integration message from another super-node
S and its leader's sub-network size is smaller than S's}

bootstrap node ← S
re-inserts itself to the network
```

[†]The maximum limit can be represented by $f_1(N)$. N is the number of the nodes in the entire system and f_1 is the function which the system can select. On the assumption that all the nodes are linked by predecessor and successor on the identifiers' space, N can be estimated by $1/d(s, succ(s))$ [18]. $1/d(s, succ(s))$ is logical distance between the node s and the successor of s.

[††]If a node chances on the super-node to which the distance is also below half the distance to its leader, *Sub-node migration* process is initiated.

[†††]The minimum limit can be represented by $f_2(N)$. N is the number of the nodes in the entire system and f_2 is the function which the system can select.

Algorithm 3
Action of the sub-node on receiving *activate partition* message from its leader

{*activate partition* message has the address of its leader and the new leader candidate node}

$d_L \leftarrow$ the distance to its leader
$d_C \leftarrow$ the distance to the new leader candidate node

if $d_C < d_L$ **then**
  bootstrap node ← the new leader candidate node
  re-inserts itself to the network
else
  $\{d_C \geq d_L\}$
  bootstrap node ← its leader
  re-inserts itself to the network
end if

Algorithm 4
Action of the super-node on receiving *check integration* message from another super-node S

{*check integration* message has the number of nodes in another super-node S's sub-network}

$n_L \leftarrow$ the number of the nodes in its sub-network
$n_A \leftarrow$ the number of nodes in S's sub-network

if $(n_L + n_A) >$ maximum limit **then**
  replies *reject integration* message to S
else
  replies *accept integration* message to S
  if $n_L < n_A$ **then**
    floods *activate integration* message to its sub-network
  end if
end if

Algorithm 5
Action of the super-node on noticing the number of the nodes in its sub-network is above the maximum limit

floods *activate partition* message to its sub-network

5. Simulation

5.1 Methodology

We used Inet [16] topologies of 3,500 autonomous systems (AS) in our experiments. Among the 3,500 autonomous systems, we randomly chose 500 ASs. We assumed the number of nodes in each AS is 10,000, and the maximum percentage of nodes joining the system is 2% of the number of nodes in each AS, which makes the maximum number of joining nodes 100,000. We also assumed the delay between the nodes in the same AS to be 0. We selected the 2-dimensional CAN algorithm as the peer-to-peer lookup algorithm in both the sub-network and super-network (when we use Grapes).

Algorithm 6
Action of the super-node on receiving *initiate integration* message from one of its sub-nodes

{*initiate integration* message has the address of the new near super-node}

$n_L \leftarrow$ the number of the nodes in its sub-network
$d_{LS} \leftarrow$ the distance to the new near super-node
$T \leftarrow$ threshold

TO_INTEGRATION:
if $n_L \geq$ minimum limit or $d_{LS} \geq \frac{1}{2}T$ **then**
  replies *no action* message to its sub-node
else
  $\{n_L <$ minimum limit and $d_{LS} < \frac{1}{2}T\}$
  sends *check integration* message to the new near super-node and waits the reply from it
  if receiving *accept integration* message **then**
    {*accept integration* message has the number of the nodes in the new near super-node's sub-network}
    $n_S \leftarrow$ the number of the nodes in the new near super-node's sub-network
    if $n_L \leq n_S$ **then**
      floods *activate integration* message to its sub-network
      bootstrap node ← the new near super-node
      re-inserts itself to the network
    else
      $\{n_L > n_S\}$
      replies *no action* message to its sub-node
    end if
  else
    {receiving *reject integration* message}
    replies *no action* message to its sub-node
  end if
end if

Algorithm 7
Action of the super-node on receiving *initiate migration* message from one of its sub-nodes

{*initiate migration* message has the address of the new near super-node}

$n_L \leftarrow$ the number of the nodes in its sub-network
if $n_L \geq$ minimum limit **then**
  replies *active migration* message to its sub-node
else
  goto TO_INTEGRATION
end if

5.2 Effects of Optimizations

To show the effects of optimization schemes, we use the simulation parameters as presented in Table 1. In the simulation, *Sub-network integration* or *Sub-node migration* is initiated when each node chances on the super-node to which the physical distance is below the threshold in the process of 10 data lookup per a node. We decided the maximum limit to be the square root of the number of nodes joining the system and the minimum limit to be the half of the maximum limit. If 2-dimensional CAN is used as a lookup algorithm in both the super-network and the sub-network like our simulation environment, the summation of

Table 1 Simulation parameters (effects of optimizations).

Number of nodes	10,000
Maximum limit	100
Minimum limit	50
Threshold	25,50,75,100, 125,150(ms)
Data lookup for optimizations	10 per 1 node

the average number of hops in the super-network and the sub-network is minimal when the number of sub-networks is $\sqrt{N}$ and the size of each sub-network is $\sqrt{N}$ (where N is the number of nodes in the whole network). The simple analysis is as follows.

- y: the summation of the average number of hops in the super-network and the sub-network[†]
- x: the number of sub-networks
- N: the number of nodes in the whole network

$$y = \frac{1}{2}\sqrt{N} + \frac{1}{2}\sqrt{\frac{N}{x}} \tag{1}$$

$$y' = \frac{1}{4}x^{-\frac{1}{2}} - \frac{1}{4}\sqrt{N}x^{-\frac{3}{2}} \tag{2}$$

When $y' = 0$, $x = \sqrt{N}$. Therefore, if $x = \sqrt{N}$, y has the minimum value.

Based on the simple analysis above, we choose $\sqrt{N}$ as a heuristic basis to decide the maximum limit and the minimum limit. We set the maximum limit and the minimum limit to $\sqrt{N}$ and $\frac{1}{2}\sqrt{N}$ respectively in the simulation.

Figure 5 and Fig. 6 show the effects of three optimization schemes; *Sub-network partition*, *Sub-network integration* and *Sub-node migration*. Figure 5 presents the number of leaders in the network over a range of thresholds. The heuristic basis number of leaders in our simulation environment is 100, the square root of the number of nodes that joined the system. The number of leaders in the optimized Grapes is closer to the basis number than that in the basic Grapes. Figure 6 shows the average delay of each sub-node to its leader over a range of thresholds. The average delay in the optimized Grapes is about 30–70% of that in the basic Grapes over a range of thresholds. Figure 5 and Fig. 6 also show the performance difference over the various thresholds. If the threshold is smaller, the average delay between the sub-nodes and the leader is shorter but the number of leaders is much larger than the basis number. If the threshold is larger, the number of leaders is closer to the basis number but the diameter of the sub-network is longer. The optimal threshold is determined by considering the trade-off between the number of leaders and the diameter of the sub-network. Also, the optimal threshold is concerned with the physical internet topology below and how many nodes joined the system in each AS among the total number of nodes in the system. Therefore, it is almost impossible to find the optimal value of the threshold. We are researching on some heuristic, distributed and dynamic methods of setting the reasonable value of the threshold. In the following

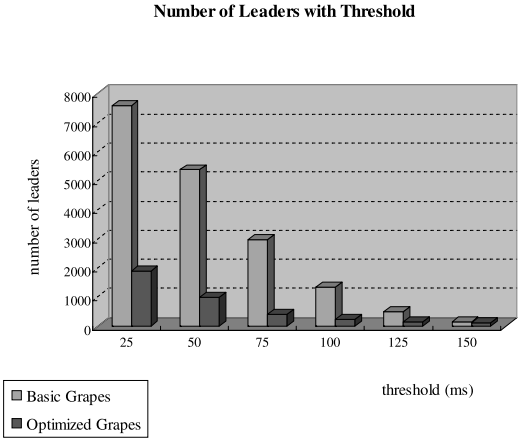


Fig. 5 The effects of optimizations (1).

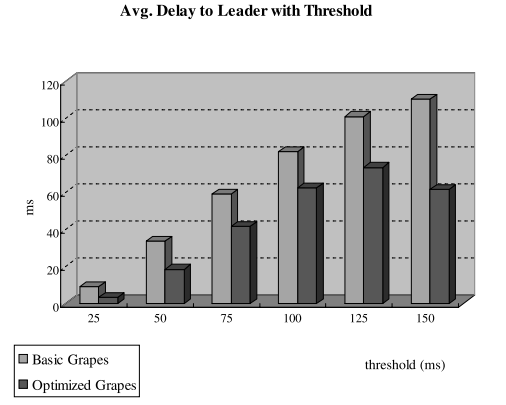


Fig. 6 The effects of optimizations (2).

Table 2 Simulation parameters (performance comparison).

Threshold	75(ms)
Kinds of data	1,000
Number of nodes	10,000;40,000; 70,000;100,000
Number of data access per node	5
Data access pattern	Uniform Exponential

simulations, the threshold is set to the specific value, 75 ms.

5.3 Performance Comparison

To compare the performance, we use the simulation parameters as presented in Table 2. We assumed that there were 1,000 kinds of data and each node looked up the data 5 times. The uniform data access pattern means the data access pattern follows the uniform distribution. The probabilities of accessing each piece of data are identical. And the exponential data access means the data access pattern follows the exponential distribution. The probability of accessing a few popular data is high and the probability of

[†]The average routing path length in CAN is $\frac{d}{4}n^{\frac{1}{d}}$[5]. d means d-dimensional CAN and n equals the number of joining nodes.

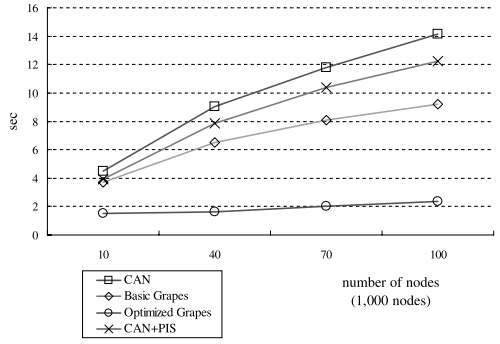


Fig. 7 Data publication time.

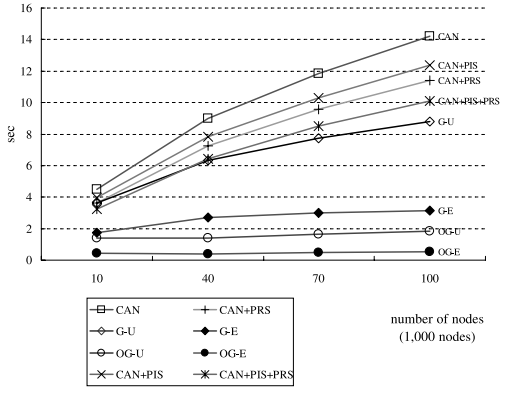


Fig. 8 Data lookup time.

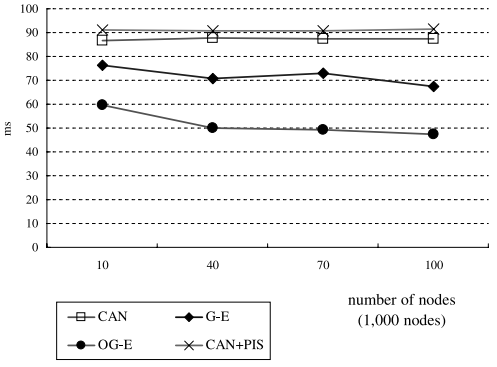


Fig. 9 Direct delay for data retrieval.

accessing most of the unpopular data is low. Figure 7 shows data publication time when an arbitrary node inserts data index into bare bones CAN[†], CAN with PIS, Grapes, and optimized Grapes. Grapes and optimized Grapes use bare bones CAN as the peer-to-peer lookup algorithm in both the sub-network and super-network. In Grapes, the data index is inserted into both the sub-network and the super-network. During the index insertion into the super-network, if the destination leader has its own sub-network, the index is inserted into the sub-network of the leader. In the maximum case, the data publication needs two sub-network path routings and one super-network path routing. Because the sub-network is composed of the nodes near each other, the routing delay for each hop in the sub-network is very small. In the super-network, the average routing delay for each hop is the same as the original CAN. However, the less number of hops (in proportion to the number of leaders) makes the total routing delay smaller. The data publication time depends on the index insertion path routing delay in the overlay network. The hierarchical structure with the physical information (the physically near sub-network and the super-network with the small number of routing hops) makes Grapes have the smaller path routing delay than that of the original CAN. The figure presents the data publication time while the number of nodes is increasing. In spite of inserting data index twice, into the sub-network and super-network, the data publication time in Grapes is shorter than that of the original CAN and even that of CAN with PIS. The data publication time in optimized Grapes is the shortest of all.

Figure 8 shows the data lookup time when an arbitrary node retrieves data from bare bones CAN, CAN with PIS, CAN with PRS, CAN with PIS and PRS, Grapes with bare bones CAN and optimized Grapes with bare bones CAN. The data lookup time depends on the path routing delay and the effects of data replication. Even if the node storing data is located in another sub-network, the total routing delay in Grapes is smaller than that of CAN without Grapes framework. Also, if the data is located in the same sub-network, the path routing delay is much shorter. The figure presents the data lookup time while the number of nodes is increasing. It shows that optimized Grapes with exponential data access (OG-E) is the most scalable scheme with regard to the network size. The data lookup time in optimized Grapes with exponential data access is one thirtieth of that of bare bones CAN when the number of nodes is 100,000. Also we can find that Grapes with uniform data access (it has the longest data lookup time among Grapes schemes) is more scalable than CAN with PIS and PRS.

Figure 9 shows the delay between the data retrieving node and the data storing node in the physical network. We refer this delay as direct delay. The direct delay depends on the location of the data. Data replication in Grapes makes the node enable to find the data in its own sub-network with a high probability. Therefore, the direct delay for data retrieval in Grapes is smaller than that in CAN. The figure presents the direct delay between two ends while the number of nodes is increasing. The direct delay in both Grapes and optimized Grapes is decreasing as the number of nodes is increasing. The reason is as follows. While the number of nodes is increasing in Grapes, the number of nodes in the sub-network is also increasing, which makes the probability of retrieval from the sub-network high (The number of autonomous systems is fixed to 3,500 in the simulation). Also, we can find that the delay in CAN with PIS is longer than that in bare bones CAN. CAN with PIS makes logical neighbor physically closer. While the nodes that are physically close to each other are clustered in a logical space, the data is distributed in a random way. It makes the data stor-

[†]Bare bones CAN is CAN with no optimization scheme [5].

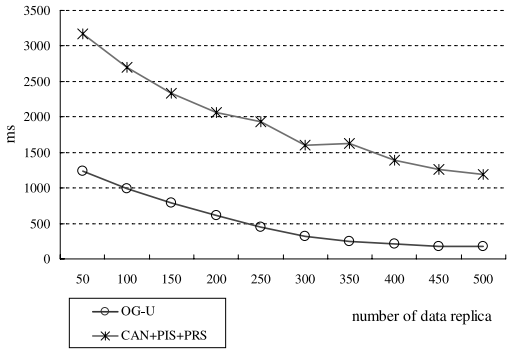


Fig. 10 Data lookup time with replication.

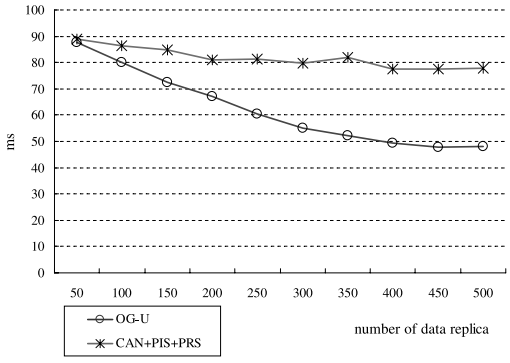


Fig. 11 Direct delay for data retrieval with replication.

ing node physically farther from the data lookup node. So the average delay between the two ends in CAN with PIS is a little bit longer than that in bare bones CAN.

Figure 10 and Fig. 11 present the data lookup time and the direct delay for data retrieval with regard to the size of data replication. To show the effects of the data replication, we modified CAN. Modified CAN stores R replica at random nodes in the overlay network. When a node retrieves data, the lookup process checks for each node on the route, whether or not the node has the replica of the data. If there is a replica on the route, the node retrieves the data from the first midway node storing the replica. We configured CAN with PIS and used PRS routing in this simulation. PRS is to select, among the possible next hops, the one that is closest in the physical network. PRS routing makes data lookup time shorter than the routing in selecting the random one among the possible next hops. CAN with PIS makes logical neighbor physically close to each other. If we retrieve data from the first midway node which stores a replica on the DHT route, we can expect the data lookup time and the direct delay for data retrieval be shorter as the number of replicas increases. R replica in Grapes is to replicate the data in R sub-network which is randomly selected. The maximum number of replicas in Grapes is the number of leaders (or sub-networks). When the Grapes node retrieves data, it looks for the node managing the hashed key of the data in its sub-network first. If the sub-network does not have the data, the node looks up its super-network with the help of the leader. After the node retrieves the data from the target node, it replicates the data in its own sub-network. In this simulation, we turned off this replication operation in order to prevent the total number of replicas from exceeding the simulation parameter value. In this simulation, the number of hosts is set to 10,000. Both optimized Grapes and CAN use uniform data access pattern which accesses all the data with the same frequency. The number of leaders in optimized Grapes is about 420, in this simulation. The maximum number of replicas is limited by the number of leaders in Grapes. Therefore, when the number of replicas is above 400 (450 and 500), the effects of the replication are restricted by the number of leaders.

Table 3 Simulation parameters (communication overhead).

Threshold	75(ms)
Number of nodes(N)	10,000;40,000; 70,000;100,000
Maximum limit	$\sqrt{N}$
Minimum limit	$\frac{1}{2}\sqrt{N}$
Data lookup for optimizations	10 per 1 node

Figure 10 shows the data lookup time while the number of replicas increases. The data lookup time in Grapes is about one third of that in CAN when the number of replicas is 50 and about one ninth of that in CAN when the number of replicas is 450. Figure 11 shows the direct delay for data retrieval between the request node and the data storing node while the number of replicas increases. When the number of replicas is 50, the average direct delay in both Grapes and CAN is about 90 ms. However, the difference in the delay between Grapes and CAN increases with regard to the size of data replication. The maximum difference is when the number of replicas is 450. In Grapes, both the data lookup time and the direct delay with the larger data replication above 450 mean nothing. (We limited the maximum number of replicas to the number of sub-networks.) However, those in CAN decrease as the number of replicas increases over 450. Although the figures show the results below 500 data replicas, but we found a fact, through simulation, that for Grapes data lookup time and direct delay is visible for around 450 replicas whereas for CAN a similar data lookup time and direct delay is visible for around 3,000 and 3,500 replicas respectively. The 3,000 and 3,500 data replicas are about one third of the number of total hosts in the network, which is considerably a huge data replication.

5.4 Communication Overhead

While the performance of Grapes is superior to that of the original CAN, Grapes need additional communication overhead cost. To analyze the communication overhead of Grapes, we use the simulation parameters as presented in Table 3. The threshold used for structuring the hierarchy of Grapes is set to 75 ms. We took a look at the changes in the communication overhead while the number of nodes is increasing. The maximum limit used for the optimization schemes is set to the square root of the number of nodes

Table 4 Communication overhead per a node.

Node	RTT	ActPart	PartRoute	InitInteg	ChkInteg	ActInteg	IntRoute	InitMig	ActMig	MigRoute
10,000	16.428	0.079	0.554	23.475	0.268	0.479	3.167	0.307	0.067	0.332
40,000	25.383	0.044	0.435	30.428	0.242	0.866	7.690	0.312	0.132	1.143
70,000	29.692	0.040	0.450	32.458	0.225	1.017	10.256	0.267	0.123	1.227
100,000	32.652	0.035	0.424	33.571	0.210	1.066	11.550	0.249	0.111	1.245

and the minimum limit is set to the half of the maximum limit. Optimization process, especially *Sub-network integration* and *Sub-node migration*, is initiated when each node happens to come in contact with a super-node where physical distance is below the threshold in the process of 10 data lookup per node. In this part of simulation, we did not consider the node's leave or failure. Additional communication overhead of the optimized Grapes is as follows. Table 4 shows how many messages each node transmits while the number of nodes is increasing.

The new node inserted to Grapes checks its physical distance to the leaders on the route in the super-network. This needs the RTT check message, which the new node sends to the leaders and the response message that the leader sends back. The frequency of the RTT check message is equal to that of the response message. RTT column in Table 4 shows the frequency of the RTT check message and the response message per node while the number of nodes is increasing. When the number of the sub-nodes is above the maximum limit, the leader floods *activate partition* message to its sub-network. Each sub-node receiving the *activate partition* message checks its physical distance to the leader candidate. The address of the new leader candidate is included in the *activate partition* message. The sub-node chooses the closer node (between the old leader and the new leader candidate) as the leader, which partitions the sub-network. When all the sub-nodes check the physical distance, the RTT check message and the response message is transmitted. The frequency of the *activate partition* message is equal to that of the RTT check message and the response message. ActPart column in Table 4 shows the frequency of the *activate partition* message, the RTT check message and the response message. When the sub-network is partitioned, its sub-nodes are inserted either to old leader's sub-network or to new leader candidate's sub-network. This additional insertion of sub-nodes produces a routing message overhead on the sub-network. PartRoute column in Table 4 shows the frequency of the additional sub-network routing message.

If a sub-node happens to come in contact with a super-node to which the physical distance is below the threshold in the process of data publication or retrieval, the node sends an *initiate integration* message to its leader. If a sub-node happens to come in contact with a super-node to which the physical distance is below half the distance to its leader, the node sends an *initiate migration* message to its leader instead of *initiate integration* message. InitInteg and InitMig column in Table 4 show the frequency of *initiate integration* message and *initiate migration* message respectively. If the number of sub-nodes is small enough for an efficient integration (below the minimum limit), and also if the distance between the leader and the new near super-node is considerably near (below half the threshold), the leader receiving *initiate integration* message checks whether the new near super-node can accept integration by sending *check integration* message. The new near super-node checks how many sub-nodes exist in its sub-network. If the summation of the number of sub-nodes in two sub-networks is below the maximum limit, the new near super-node sends *accept integration* message. Otherwise, the new near super-node sends *reject integration* message. The frequency of *check integration* message is equal to that of the response message (*accept/reject integration* message). ChkInteg column in Table 4 shows the frequency of *check integration* message and the response message. If the new near super-node can accept integration, the leader which has smaller number of sub-nodes between two sub-networks floods *activate integration* message to its own sub-network (ActInteg column in Table 4). The sub-nodes receiving *activate integration* message are inserted to the correspondent leader's sub-network. This additional insertion of sub-nodes produces routing message overhead on the sub-network (IntRoute column in Table 4). If the number of sub-nodes is small enough (below the minimum limit), the leader receiving *initiate migration* message tries to do *Sub-network integration* process. Otherwise, the leader sends *activate migration* message to the sub-node (ActMig column in Table 4). The sub-node receiving *activate migration* message is inserted to the correspondent leader's sub-network. This additional insertion of the sub-node produces routing message overhead on the sub-network (MigRoute column in Table 4).

Table 5 Total communication overhead.

Number of nodes	Total message frequency per a node
10,000	62.010
40,000	92.388
70,000	105.752
100,000	114.045

Table 4 shows top three messages (RTT, InitInteg, and IntRoute) produce most of communication overhead (about 96–97% of total communication overhead in Table 5). The frequency of these three messages per node is increasing as the number of nodes is increasing. The degree of the increment, however, is decreasing. Table 5 presents the total message frequency per node while the number of nodes is increasing. The total message frequency includes all kinds of overhead messages in Table 4 (especially the frequency of RTT which is added twice, ActPart three times, and ChkIteg twice). The frequency of the total messages per node is increasing while the number of nodes is increasing. The degree of the increment, however, is decreasing.

If we assume the size of all the messages to be 50 bytes (IP header −20 bytes, TCP header −20 bytes, and additional information such as message identification tag and node address −10 bytes), total communication overhead is about 5,700 bytes per node even when the number of nodes is 100,000. The simulation time where each node executes 10 data lookups is 1,000 seconds. The communication overhead each node produces is about 5.7 bytes per second (that is 45.6 bps) when the number of nodes is 100,000.

6. Conclusion

In this paper, we proposed a proximity-based self-organizing hierarchical overlay framework for DHTs, called Grapes. In Grapes, a node can find data in its sub-network with a high probability due to the data replication in its sub-network. In DHT systems, the replication is important to make up for the reliability problem due to the dynamic characteristic of peer-to-peer network. Grapes supports an efficient data replication considering the physical topology. Grapes also makes the lookup time smaller than that of the flat one due to the hierarchical structure using the physical information. There are some other schemes to provide hierarchical concepts for peer-to-peer networks. Grapes is different from those in a sense that it provides the specific grouping algorithm without any centralized control. Grapes is the only unique topology-aware hierarchical overlay network supporting the original peer-to-peer concept of self-organization. Due to its self-organization characteristic, Grapes is not easy to make an optimal hierarchical structure. To solve this problem, we proposed three optimization schemes; *Sub-network partition*, *Sub-network integration*, and *Sub-node migration*. The optimization schemes do not provide an optimal architecture, but they help Grapes to have a more efficient hierarchical structure.

Acknowledgments

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc) and University IT Research Center Project.

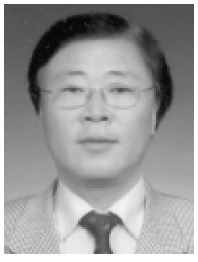
References

[1] Gnutella, http://www.gnutella.com
[2] Napster, http://www.napster.com/
[3] S. Ratnasamy, S. Shenker, and I. Stoica, "Routing algorithms for DHTs: Some open questions," Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, March 2002. http://www.cs.rice.edu/Conferences/IPTPS02
[4] K. Shin, S. Lee, G. Lim, J. Ma, and H. Yoon, "Grapes: Topology-based hierarchical virtual network for peer-to-peer lookup services," Proc. International Workshop on Ad Hoc Network, pp.159–166, Vancouver, B.C., Canada, Aug. 2002.
[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," Proc. ACM SIGCOMM, pp.161–172, San Diego, CA, Aug. 2001.
[6] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrkshnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," Proc. ACM SIGCOMM, pp.149–160, San Diego, CA, Aug. 2001.
[7] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," Proc. 18th IFIP/ACM Int. Conf. Distributed Systems Platforms (Middleware 2001), pp.329–350, Nov. 2001.
[8] B.Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Comput. Sci. Div., Univ. California, Berkeley, Tech. Rep., UCB/CSD-01-1141, 2001.
[9] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically aware overlay construction and server selection," Proc. IEEE INFOCOM Conference, pp.1190–1199, New York, NY, June 2002.
[10] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," Proc. ACM SIGCOMM, pp.381–394, Karlsruhe, Germany, Aug. 2003.
[11] F. Dabek, A cooperative file system, Master's Thesis, Massachusetts Inst. Technol., Cambridge, 2001.
[12] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," IEEE/ACM Trans. Netw., vol.11, no.1, pp.17–32, Feb. 2003.
[13] B.Y. Zhao, Y. Duan, L. Huang, A.D. Joseph, and J.D. Kubiatowicz, "Brocade: Landmark routing on overlay networks," Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, March 2002. http://www.cs.rice.edu/Conferences/IPTPS02
[14] L. Garces-Erce, E. Biersack, P. Felber, K.W. Ross, and G. Urvoy-Keller, "Hierarchical peer-to-peer systems," Proc. ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par), LNCS 2790, pp.1230–1239, Klagenfurt, Austria, 2003.
[15] K.P. Gummadi, S. Saroiu, and S.D. Gribble, "King: Estimating latency between arbitrary Internet end hosts," Proc. SIGCOMM IMW 2002, pp.5–18, Nov. 2002.
[16] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMAPS: A global Internet host distance estimation service," IEEE/ACM Trans. Netw., vol.9, no.5, pp.525–540, Oct. 2001.
[17] E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," Proc. IEEE INFOCOM Conference, pp.170–179, New York, NY, June 2002.
[18] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," Proc. ACM Conf. on Principles of Distributed Computing (PODC), pp.183–192, Monterey, CA, July 2002.
[19] J. Winick and S. Jamin, "Inet-3.0: Internet topology generator," Technical Report CSE-TR-456-02, University of Michigan, EECS Dept., 2002.
[20] Z. Xu, M. Mahalingam, and M. Karlsson, "Turning heterogeneity into an advantage in overlay routing," Proc. IEEE INFOCOM Conference, pp.1499–1509, San Francisco, CA, April 2003.
[21] M. Castro, P. Druschel, Y.C. Hu, and A. Rowstron, "Exploiting network proximity in distributed hash tables," Proc. International Workshop on Future Directions in Distributed Computing (FuDiCo), pp.52–55, June 2002.
[22] M. Castro, P. Druschel, Y.C. Hu, and A. Rowstron, "Topology-aware routing in structured peer-to-peer overlay networks," Technical Report MSR-TR-2002-82, Microsoft Research, 2002.
[23] S. Ratnasamy, A scalable content-addressable network, Doctor's Dissertation, University of California at Berkeley, 2002.

Kwangwook Shin received the B.S. degree in computer science from the Sogang University, Korea, in 1997, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 1999. He is currently working toward the Ph.D. degree at the Division of Computer Science, Department of Electrical Engineering and Computer Science, KAIST. His research interests include mobile and wireless communications, mobile ad hoc networks, and peer-to-peer networks.

Seunghak Lee received the B.S. degree and the M.S. degree in computer science from the KAIST in 2000 and 2003 respectively. He is currently working toward the Ph.D. degree at the Division of Computer Science, Department of Electrical Engineering and Computer Science, KAIST. His research interests include wireless sensor networks, mobile ad hoc networks, and peer-to-peer networks.

Geunhwi Lim received the B.S. degree, the M.S. degree, and the Ph.D. degree in computer science from the KAIST in 1996, 1998, and 2003 respectively. He has been working for the Global Standard and Research team in Samsung Electronics since 2003. His research interests include wireless communications, mobile ad hoc networks, and peer-to-peer networks.



Hyunsoo Yoon received the B.S. degree in electronics engineering from the Seoul National University, Korea, in 1979, the M.S. degree in computer science from the KAIST, in 1981, and the Ph.D. degree in computer and information science from the Ohio State University, Columbus, Ohio, in 1988. From 1988 to 1989, with the AT&T Bell Labs. as a Member of Technical Staff. Since 1989 he has been a Faculty Member of the Division of Computer Science at the KAIST. His research interests include parallel computer architecture, mobile communication, ad hoc networks, and information security.