Message Routing Algorithm with Additional Node-Information for Capability-Aware Object Management in P2P Networks

Takuji Tachibana*

Abstract—**In large-scale heterogeneous P2P networks, it is indispensable to manage objects based on node's capabilities. In order to achieve this end, capability-aware object management based on Skip List has been proposed. However, in this method, the number of hops for message routing increases as the number of nodes becomes large. In this paper, for the capability-aware object management, we propose a new message routing algorithm s in order to decrease the number of hops. In our proposed algorithm, additional node-information is included in a transmitting message and this information is used so that a node does not receive the same message doubly. Each node can use our proposed algorithm simply, and its implementation is easy. We evaluate the performance of the proposed method by simulation. Numerical examples show that the proposed method can decrease the average number of hops in any cases. In addition, the maximum number of hops decreases significantly when the number of nodes is large.**

Keywords: P2P networks, message routing, Skip List, object management, additional information

1 Introduction

Peer-to-Peer (P2P) networks have been widely used over the Internet for many applications, for example, Internet telephony, distributed data storages, data streaming, and online games. In the future, it is supposed that such P2P applications are utilized by various nodes such as high-performance computer, mobile hand-held devices, and sensor nodes [1, 2] (see Fig. 1).

In such heterogeneous P2P networks, for managing objects by considering node's capabilities, [3] has proposed capability-aware object management based on SkipNet [4, 5]. This method utilizes two identifications called *TypeID* and *HashID*. TypeID is assigned to each node according to its own capabilities such as forwarding capability, data-storage capability, and mobility. On the other hand, HashID is assigned to each node for providing load balancing among similar types of nodes. Message routing for storing and searching objects is performed according to TypeID and HashID.

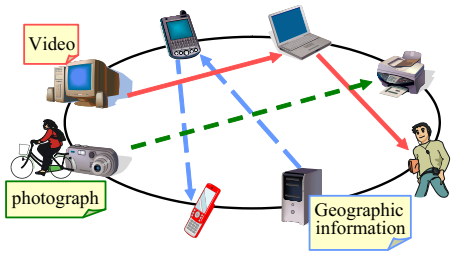

Figure 1: Node structure and message routing.

By using this method, an object can be stored in a node with the specified capabilities or searched from a node with the specified capabilities. The load balancing can also be provided among nodes with similar capabilities. However, when the message routing based on TypeID is performed, a message is likely to be routed to multiple nodes with the same TypeID, increasing the number of hops. In addition, when the message routing based on HashID is performed, a message is likely to be traversed on a node doubly, increasing the number of hops. As a result, the capability-aware object management can not provide sufficient performance in terms of the number of hops for message routing.

In [6], shortcut message routing has been proposed in order to decrease the number of hops for TypeID-based message routing. With this method, a message is always forwarded to a node with different TypeID at two hops. However, the number of hops for TypeID-based message routing is much smaller than that for HashID-based message routing. Therefore, the method in [6] is not so effective for decreasing the total number of hops.

In this paper, we propose a message routing algorithm so that the number of hops for HashID-based message routing decreases. In our proposed routing algorithm, additional node-information is included in a message and this information is used to avoid the redundant message forwarding. Each node can use our proposed algorithm simply, and its implementation is easy. We evaluate the performance of our proposed method by simulation. We

*Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630-0192, Japan. Tel/Fax: +81-743-72-5352/5359 Email: takuji-t@is.naist.jp

Table 1: Assignment of four-digit TypeID.

	0	1
w	$C(w) \geq 1$ Gbps	$C(w) < 1$ Gbps
x	$C(x) \geq 50$ Gbytes	$C(x) < 50$ Gbytes
y	$C(y) =$ low	$C(y) =$ high
z	$C(z) =$ high	$C(z) =$ low

compare the performance of the proposed method with that of the conventional method [3].

The rest of the paper is organized as follows. Section 2 introduces the capability-aware object management [3], and Section 3 explains the proposed message routing algorithm. Numerical examples are shown in Section 4 and finally, conclusions are presented in Section 5.

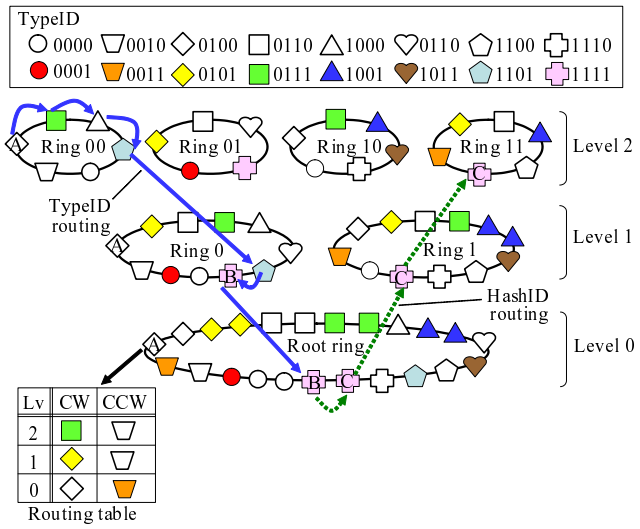

Figure 2: Node structure and message routing.

2 Capability-Aware Object Management

2.1 Overview

The capability-aware object management has been proposed in order to manage objects by considering node's capabilities in large-scale heterogeneous P2P networks [3]. This method is based on SkipNet, and it utilizes two identifications called *TypeID* and *HashID*.

TypeID is assigned to each node for specifying its capabilities such as forwarding capability and data-storage capability. For example, when four-digit TypeID ($wxyz$) is used, the first digit (w) denotes forwarding capability, the second digit (x) data-storage capability, the third digit (y) mobility, and the fourth digit (z) availability, and then each digit number is determined as shown in Table 1. On the other hand, HashID is assigned to each node by applying a collision-resistant hash function. Node's IP address or others are used as arguments of the hash function.

Figure 2 shows a node structure for the capability-aware object management in a case of four-digit TypeID. In this structure, there are one or more rings at each level, and rings at level i are obtained by splitting a ring at level $i-1$ into multiple disjoint sets. The number of levels is $H+1$ when the number of digits of HashID is H. Each node belongs to a ring at every level so that i-digits prefix of HashID is shared by other nodes. Because nodes are sorted by TypeID at each ring, nodes with the same TypeID, i.e., similar capabilities, are located in a ring sequentially.

Each node has a routing table which includes neighbor nodes at each level (see node A in Fig. 2). Figures 3 and 4 show two message routing algorithms. In Source node A sends a message which includes destination TypeID and HashID. At first, the message is routed based on TypeID as shown in Fig. 3 (solid lines in Fig. 2). When the message arrives at a node with the destination TypeID, the message routing based on TypeID terminates (node B in Fig. 2).

Just after the termination of the message routing based on TypeID, the message is routed based on HashID as shown in Fig. 4 (dotted lines in Fig. 2). This message routing is performed only among nodes with the destination TypeID. When the message is received by a node whose HashID is closest to the destination HashID, the message routing based on HashID terminates.

As an example of how the capability-aware object management is used, we can consider the following case. When a driver tries to listen a favorite song, the driver inputs its title and file type into the software. In this software, TypeID of the music file is determined from its file type, and HashID is determined with a hash function from its title. Then, message routing starts in the P2P network to find a destination node with the music file based on the determined TypeID and HashID. If the message reaches the destination node, the driver can download the object from this node.

2.2 Drawbacks

When a message is routed based on TypeID, the message traverses among nodes whose TypeIDs are between source node's TypeID and destination TypeID. Figure 5 shows an example of node structure in a case where a large number of nodes have the same TypeID. In this figure, six nodes from A to G have TypeID X, and routing tables of nodes C and E include only nodes with TypeID X. In this case, the message routing has to be performed several times by nodes with TypeID X before the message is forwarded to a node with different TypeID. This increases the number of hops for TypeID-based message

```
SendMsg(TypeID, HashID, msg) {
  if(LongestPrefix(TypeID, localNode.TypeID) == 0)
    msg.dir = RandomDirection();
  else if(TypeID < localNode.TypeID)
    msg.dir = false;  // CounterClockwise
  else
    msg.dir = true; // Clockwise

  msg.TypeID = TypeID;  msg.HashID = HashID;
  RouteByTypeID(msg);
}

RouteByTypeID(msg) {
  h = localNode.MaxRoutingTableHeight;
  while(h >= 0) {
    if(LiesBetween(msg.dir, localNode.TypeID,
       localNode.RoutingTable.[h][msg.dir].TypeID ,
       msg.TypeID ) == false )
    {
      h--; continue;
    }

    if(LiesBetween(msg.dir, localNode.TypeID,
       localNode.RoutingTable.[h][msg.dir].TypeID ,
       msg.TypeID ) == true )
    {
       NextCandidateNode = localNode.RoutingTable.[h][msg.dir];
       if(CheckIfAlreadyVisited(msg, NextCandidateNode))
       {
          h--; continue;
       }
       msg.AlreadyVisited(localNode);
       SendtoNode(NextCandidateNode, msg);  return;
    }
  }

  if( localNode.TypeID != msg.TypeID )
  {
     NegativeAck(msg); return;
  }

  msg.dir = true;
  RouteByHashID(msg);
}
```

Figure 3: Routing algorithm based on TypeID.

```
RouteByHashID(msg) {
  if( msg.HashID == localNode.HashID  ||
                          msg.FinalDestination == true ) {  … (A)
     DeliverMessage(msg);
     return;
  }

  if(msg.StartNode != null &&  localNode == msg.startNode ) {  … (B)
     msg.FinalDestination = true;
     SendtoNode(msg.bestNode);
     return;
  }

  h = CommonPrefixLen(msg.HashID, localNode.HashID);

  if( h > msg.ringLvl) {
     msg.ringLvl = h;
     msg.startNode = msg.bestNode = localNode;
  }
  if( abs(localNode.HashID - msg.HashID) < abs(msg.bestNode.HashID -
       msg.HashID) ){
     msg.bestNode = localNode;
  }

  if( localNode.RoutingTable[h][msg.dir].TypeID == msg.TypeID ){
     SendtoNode(msg, localNode.RoutingTable[h][msg.dir]);
  }
  else if( msg.dir == true ){
     msg.FinalDestination = false;
     msg.startNode = null;
     msg.dir = false;
     SendtoNode(msg, localNode);
  }
  else if( msg.dir == false ) {  … (C)
     msg.FinalDestination(true);
     SendtoNode(msg, msg.bestNode);
  }
}
```

Figure 4: Routing algorithm based on HashID.

routing.

In addition, the message routing based on HashID is performed as shown in Fig. 6. Each node checks whether TypeID of its neighbor node is the same as its own TypeID, and the node forwards the message to the neighbor node if the neighbor node has the same TypeID (see (1) of Fig. 6). At this time, when HashID of the neighbor node is the most closest to the destination HashID, the information about the neighbor node is stored in the message as the best node. When the neighbor node of a node has a different TypeID, the current node reverses the routing direction and continues the message routing (see (2) of Fig. 6). The HashID-based message routing terminates when the node with the destination HashID is found (see (A) of Fig. 4), when the message routing for both directions finishes (see (C) of Fig. 4 and (3) of Fig. 6(a)), or when a node receive the message again in the initial direction if ring structure is constructed (see (B) of Fig. 4). In this message routing, some nodes may receive the message twice. When the number of nodes with destination TypeID is large, the number of hops for HashID-based message routing increases.

[6] has proposed shortcut message routing which can decrease the number of hops for TypeID-based message routing. With this method, a message can be forwarded to a node with different TypeID at two hops. However, the number of hops for TypeID-based message routing is much smaller than that for HashID-based message routing. Therefore, this method cannot decrease the total number of hops significantly.

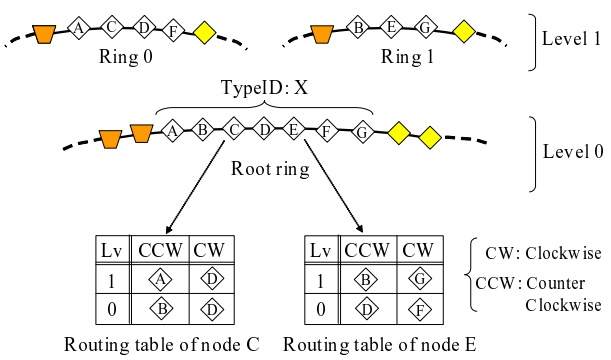

Figure 5: Node structure and routing tables in a case where a large number of nodes have the same TypeID.

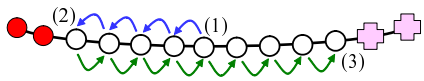

Figure 6: Node structure for message routing based on HashID.

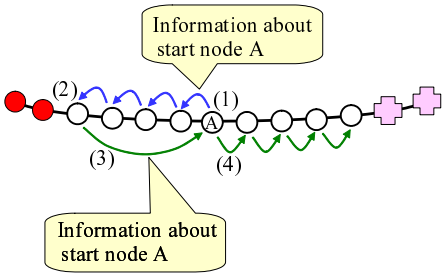

Figure 7: Hash-based message routing with the proposed method.

3 Message Routing Algorithm with Additional Information

In this paper, we propose a new message routing algorithm for the HashID-based message routing. In our proposed method, additional node-information is used for HashID-based message routing. Here, the additional information specifies a node from which HashID-based message routing starts at a level (see node A in Fig. 7). Every time the level of HashID-based message routing changes, this node-information is also updated by a new start node.

The additional node-information is used in the following (see Fig. 7).

(1) When a node receives a message and the ring-level increases, the node updates information about start node in the message.

(2) The node forwards a message to a neighbor node along the initial direction according to the conventional Hash-based message routing.

(3) A border node receives the message, the node checks the information of the start node in the message.

(4) The border node forwards the message to the start node directly.

(5) The start node forwards to the message to another neighboring node along the opposite direction.

Figure 8 shows the detailed routing algorithm for HashID message routing. Here, the message transmission from a border node to the start node is highlighted in boldface type. As shown in this figure, it is easy to implement the shortcut message routing for HashID.

```
RouteByHashIDWithShortcut(msg) {
  if( msg.HashID == localNode.HashID ||
                                msg.FinalDestination == true ) {
    DeliverMessage(msg);
    return;
  }

  if(msg.StartNode != null &&  localNode == msg.startNode
      && msg.dir == true  ) {
    msg.FinalDestination = true;
     SendtoNode(msg.bestNode);
    return;
  }

  h = CommonPrefixLen(msg.HashID, localNode.HashID);

  if( h > msg.ringLvl) {
    msg.ringLvl = h;
    msg.startNode = msg.bestNode = localNode;
  }
  if( abs(localNode.HashID - msg.HashID) < abs(msg.bestNode.HashID -
      msg.HashID) ){
    msg.bestNode = localNode;
  }

  if( localNode.RoutingTable[h][msg.dir].TypeID == msg.TypeID ){
    SendtoNode(msg, localNode.RoutingTable[h][msg.dir]);
  }
  else if( msg.dir == true ){
    msg.FinalDestination = false;
    msg.dir = false;
    SendtoNode(msg, msg.startNode);
  }
  else if( msg.dir == false ) {
    msg.FinalDestination(true);
    SendtoNode(msg, msg.bestNode);
  }
}
```

Figure 8: Proposed HashID-based routing algorithm.

4 Numerical Examples

In this section, we evaluate the performance of the proposed method by simulation. We assume that the number of P2P nodes is N and the number of objects which should be managed is $M = \lfloor N/2 \rfloor$.

In this P2P network, a four-digit TypeID is assigned to each node and each object. For the simplicity, in the following, we denote four-digit TypeID with decimal number format, for example, TypeID 0101 is denoted as TypeID 5. We assume that TypeID i $(0 \leq i \leq 15)$ is assigned to a node (an object) with probability γ_i (σ_i). On the other hand, HashID is denoted as 128 bits binary string, and it is assigned to a node (an object) with a hash function.

Under this situation, we evaluate the performance of the proposed shortcut message routing for large-scale heterogeneous P2P networks. We have executed 50 simulations for the same parameters and have computed the average number of hops for the proposed method. In addition, we have derived the maximum number of hops for the proposed method among 50 $\times M$ message routings. For the performance comparison, we also evaluate the performance of the conventional capability-aware object management shown in Fig. 2.

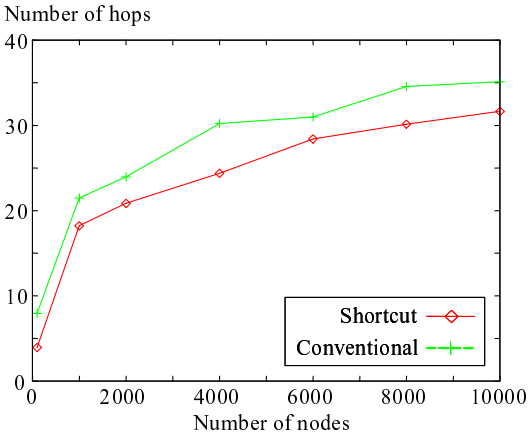

Figure 9: Average number of hops for HashID-based message routing.

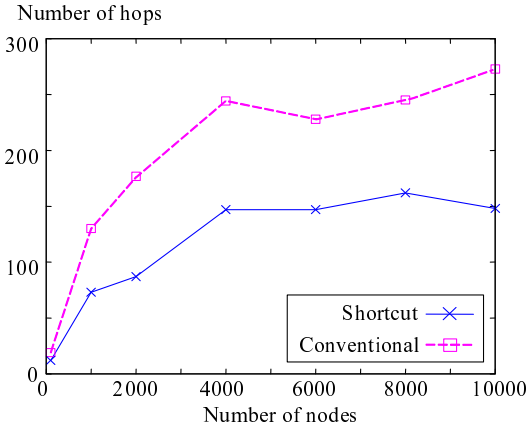

Figure 10: Maximum number of hops for HashID-based message routing.

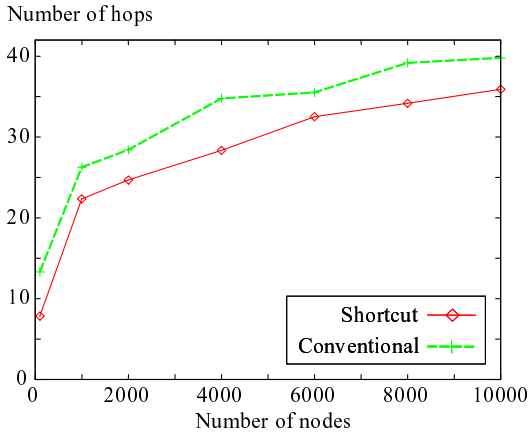

Figure 11: Average number of hops for the proposed and conventional methods.

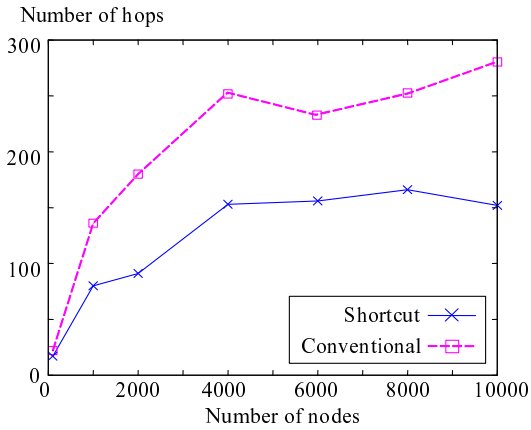

Figure 12: Maximum number of hops for the proposed and conventional methods.

4.1 Impact of Number of Nodes

In this subsection, we investigate how the number of hops changes by using the proposed method as the number of hops increases. Here, TypeID i is assigned to a node and an object with probabilities γ_i=0.0625 and σ_i=0.0625, respectively.

Figure 9 shows the average number of hops for HashID-based message routing. From Fig. 9, we find that the average numbers of hops for both methods increases the number of nodes N increases, as expected. However, the average number of hops for the proposed method is smaller than that for the conventional method, regardless of the number of nodes. This result shows that the proposed method can decrease the average number of hops for HashID-based message routing. The difference between the numbers of hops for both methods does not change so much. Therefore, the effectiveness of the proposed method does not become small.

Figure 10 also shows the maximum number of hops for HashID-based message routing, respectively. From this figure, we can also find that the maximum number of hops for the proposed method is smaller than that for the conventional method, regardless of the number of nodes. In especial, the proposed method can decrease the maximum number of hops for HashID-based message routing by about 45 % when the number of nodes is 10,000.

The performances for the total number of hops are shown in Figs. 11 and 12. In Figs. 9 to 12, we can observe that results for total number of hops are similar to those for HashID-based message routing. This denotes that HashID-based message routing is dominant. From Figs. 11 and 12, we can conclude that the proposed method is effective to decrease the number of hops and the effectiveness increases as the number of nodes becomes large.

4.2 Impact of TypeID Assignment

In this subsection, we investigate the impact of TypeID assignment on the performance of the proposed method. We consider two cases for TypeID assignment; case 1 and

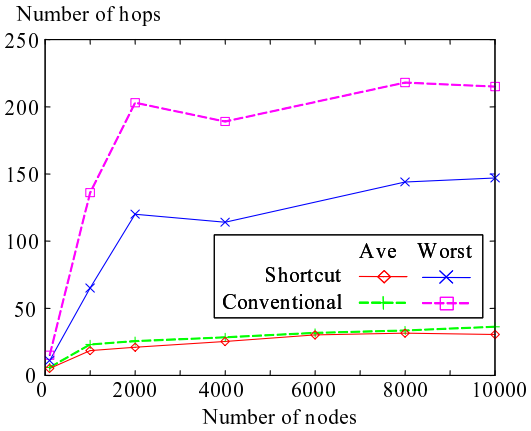

Figure 13: Average and maximum numbers of hops for HashID-based message routing in case 1.

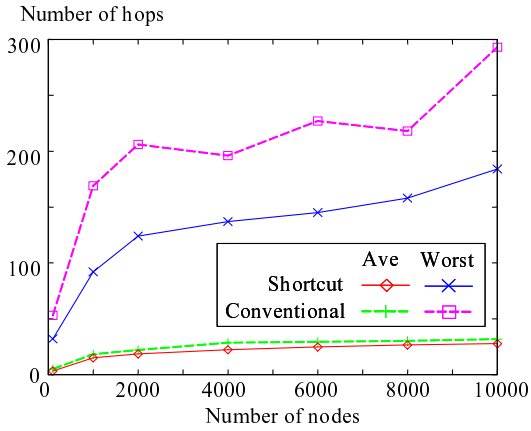

Figure 14: Average and maximum numbers of hops for HashID-based message routing in case 2.

case 2. In case 1, $\gamma_0 = 0.5$, $\gamma_1 = \gamma_2 = 0.1$, $\gamma_3 = \gamma_4 = 0.05$, $\gamma_5 = \cdots = \gamma_{14} = 0.02$, $\gamma_{15} = 0.0$, and $\sigma_0 = \cdots = \sigma_{15} = 0.0625$. In case 2, on the other hand, $\gamma_0 = \cdots = \gamma_{15} = 0.0625$, $\sigma_0 = \cdots = \sigma_9 = 0.02$, $\sigma_{10} = \sigma_{11} = 0.05$, $\sigma_{12} = \sigma_{13} = 0.1$, $\sigma_{14} = 0.2$, and $\sigma_{15} = 0.3$.

Figure 13 shows the average and maximum numbers of hops for HashID-based message routing against the number of nodes N for case 1. From this figure, we find that the proposed method can decrease the average and maximum numbers of hops for HashID-based message routings. The proposed method can decrease the average number of hops for HashID-based message routing decreases by 10 % and the maximum number of hops decreases by 40 %.

Figure 14 also shows the average and maximum numbers of hops for HashID-based message routing for case 2. From this figure, we can also observe that the proposed method can decrease significantly the number of hops for HashID-based message routing. As a result, the proposed method is effective when there are a large number of nodes with different capabilities.

5 Conclusions

In this paper, for the capability-aware object management based on SkipNet, we proposed a message routing algorithm in order to decrease the number of hops. In our proposed routing algorithm, additional node-information is used so that a node does not receive the same message twice. The proposed method can be implemented and used simply. We evaluate the number of hops for the proposed method by simulation. From simulation results, we found that our proposed method can decrease the average number of hops for HashID-based message routings. As the number of nodes increases, the effectiveness of the proposed becomes large. In addition, the proposed method can decrease the maximum number of hops significantly. We also found that our proposed method is effective in large-scale heterogeneous P2P networks, where a large number of nodes with different capabilities have participated and a large variety of objects are managed. From the above results, it is expected that the proposed method is a promising method in large-scale P2P networks for ubiquitous computing environments.

References

[1] Hu, J., Li, M., Yu, H., Zheng, W., "Tourist: A Self-Adaptive Structured Overlay in Heterogeneous P2P Networks," *Technical Report THTR-CS-HPC-2006-001*, 2006.

[2] Xu, Z., Mahalingam, M., Karlsson, M., "Turning Heterogeneity into an Advantage in Overlay Routing," *IEEE INFOCOM 2003*, 4/03.

[3] Tomimoto, T., Tachibana, T., Sugimoto, K., "Capability-Aware ID Assignment and Message Routing based on SkipList in Large-Scale Heterogeneous P2P Networks," *IEEE Globecom 2007*, 11/07.

[4] Pugh, W., "Skip Lists: A Probabilistic Alternative to Balanced Trees," *Workshop on Algorithms and Data Structures*, 7/03.

[5] Harvey, N., Jones, M., Saroiu, S., Theimer, M., Wolman, A., "SkipNet: A Scalable Overlay Network with Practical Locality Properties," *4th USENIX Symposium on Internet Technologies and Systems*, 3/03.

[6] Tomimoto, T., Tachibana, T., Sugimoto, K., "Capability-aware ID assignment and message routing based on SkipList in large-scale heterogeneous P2P Networks," *NAEC 2007*, 10/07.