Self-Correcting Broadcast in Distributed Hash Tables *

Ali Ghodsi[1],Luc Onana Alima[1], Sameh El-Ansary[2], Per Brand[2] and Seif Haridi[1]
[1]IMIT-Royal Institute of Technology, Kista, Sweden
[2]Swedish Institute of Computer Science, Kista, Sweden
{aligh, onana, seif}@it.kth.se, {sameh, perbrand}@sics.se

ABSTRACT
We present two broadcast algorithms that can be used on top of distributed hash tables (DHTs) to perform group communication and arbitrary queries. Unlike other P2P group communication mechanisms, which either embed extra information in the DHTs or use random overlay networks, our algorithms take advantage of the structured DHT overlay networks without maintaining additional information. The proposed algorithms do not send any redundant messages. Furthermore the two algorithms ensure 100% coverage of the nodes in the system even when routing information is outdated as a result of dynamism in the network. The first algorithm performs some correction of outdated routing table entries with a low cost of correction traffic. The second algorithm exploits the nature of the broadcasts to extensively update erroneous routing information at the cost of higher correction traffic. The algorithms are validated and evaluated in our stochastic distributed-algorithms simulator.

KEY WORDS
Distributed Algorithms, Distributed Hash Tables, Group Communication, Peer-to-Peer .

1 Introduction

The need for making effective use of the huge amount of computing resources attached to large scale networks, such as the Internet, has established a new field within the distributed computing area, namely, Peer-to-Peer (P2P) computing.

The current trend in this new field builds on the idea of distributed hash tables (DHT) that provide infrastructures for scalable P2P systems [11, 13, 1, 7]. The infrastructure is a logical network, called an *overlay network*, within which key/value pairs are stored. The main operation offered by DHT-based overlay networks is the lookup operation, that is finding a value associated with a given key. However, the lookup operation itself is not enough to perform arbitrary queries such as context dependent searches. Furthermore, it is difficult, in large DHT systems, to collect statistical information about the system, such as the overall system usage for billing purposes.

*This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-33234 PEPITO project and partially by the Vinnova PPC project in Sweden

In this paper we present two broadcast algorithms for the distributed k-ary system ($\mathcal{DKS}$) [1] that can be used to solve the above mentioned problems. The choice of $\mathcal{DKS}$ is motivated by two reasons. First, the $\mathcal{DKS}$ systems, in contrast to all other systems [9, 13, 5], avoid the use of periodic stabilization protocols for maintaining routing information. Instead, a novel technique called *correction-on-use* serves to correct outdated routing information on-the-fly. Network bandwidth is thus saved during periods when activity is low. Second, $\mathcal{DKS}$ provides the ability to tune the ratio between routing table size and maximum lookup length. E.g. a system can be configured with large routing tables and a low maximum lookup length, consequently, making broadcasts faster.

1.1 Contribution

The work in [3] paved the way for doing broadcasts on top of structured P2P networks such as the Chord system [11, 12]. However, the algorithm in [3] fails to cover all nodes when the routing information is inconsistent, which is the natural case in dynamic P2P networks as a consequence of nodes joining or leaving.

In this paper we present two broadcast algorithms that deal with routing table inconsistencies. The new broadcast algorithms guarantee 100% coverage even in the presence of frequent network changes and outdated routing information. Furthermore, unlike other similar attempts[8], nodes do not receive any redundant messages.

Furthermore, we extend the $\mathcal{DKS}$ philosophy of avoiding the use of periodic stabilization. The second broadcast algorithm exploits the nature of a broadcast to effectively correct outdated routing information at the cost of extra local computation and network traffic.

The proposed algorithms can be used to perform multicast. Each multicast group is then represented by an instance of $\mathcal{DKS}$ within which the proposed broadcast algorithms can be used to disseminate multicast messages.

1.2 Related work

Our work can be classified as extending DHTs to support arbitrary-searches. From that perspective, the research in complex queries shares the same goal. In [4] the idea is to construct search indices that enable the performance of database-like queries. This approach differs from ours in that we do not add extra indexing to the DHT. The analysis of the cost of construction, maintenance, and performing

database-like join operations is not available at the time of writing of this paper.

Since broadcast is a special case of multicast, a multicast solution developed for a DHT such as [10, 8, 2] can provide broadcast functionality. Nevertheless, a multicast solution would require the additional maintenance of a multicast group which, in the case of broadcast, is a large group containing all the nodes in the network. For example [2] uses one rendez-vous node per group, that disseminates messages with the help of potential non-members called forwarders by using multicast trees. In [8], a bootstrap node stores information about a group, in which it is not necessarily a member. Additionally, there is a inherent redundancy of messages when the coordinate space is not perfectly partitioned. In our approach, these two drawbacks are avoided.

1.3 Outline

The remaining of this paper is organized as follows. In section 2 we give an overview of $\mathcal{DKS}$ systems. Section 3 provides informal and formal descriptions of the proposed algorithms. Section 4 is devoted to the validation and the evaluation of the two algorithms. Finally, section 5 concludes.

2 $\mathcal{DKS}$ overview

In the following sub-sections we present the $\mathcal{DKS}$ systems. We focus on its two main contributions, a generalization to tune the lookup length, and a correction-on-use technique used to avoid periodic stabilization protocols for maintaining routing information.

2.1 Structure of the $\mathcal{DKS}$

$\mathcal{DKS}$ systems are configured with the parameters, N, and $k \geq 2$, such that the lookup length is guaranteed to take at most $log_k(N)$ hops for a network of maximum size N. With k defined, the maximum number of nodes that can be simultaneously in a $\mathcal{DKS}$ network is chosen to be $N = k^L$ for some large L. Every node knows k and N, and can therefore compute L.

Once N has been defined, all nodes and keys in the system are deterministically mapped onto the identifier space, $\mathcal{I} = \{0, 1, .., N-1\}$, by using a globally known hash function, H. The identifier space is a circular space modulo N.

Each $key/value$ pair is physically stored at the first node encountered in the ring, moving in clockwise direction, starting at $H(key)$.

We shall use the notation $a \oplus b$ for $(a + b)$ modulo N for all $a, b \in \mathcal{I}$. The whole identifier space can be represented by an interval of the form $[x, x[$ or $]x, x]$ for an arbitrary $x \in \mathcal{I}$. For any $x \in \mathcal{I}$, we note that $[x, x] = \{x\}$ and $]x, x[= \mathcal{I} \backslash \{x\}$.

2.2 Routing tables

Each node, in addition to storing key/value pairs, maintains a routing table. The routing table consists of $log_k(N)$ *levels*. Let $\mathcal{L} = \{1, 2, .., log_k(N)\}$ be the set of levels.

At each level, $l \in \mathcal{L}$, a node n has a view of the identifier space defined as:

$$V_l = [n, n \oplus \frac{N}{k^{l-1}}[$$

This means that for level one, the view consists of the whole identifier space, and at any other level $l > 1$, one k:th of V_{l-1} is considered.

At any level $l \in \mathcal{L}$, the view is partitioned into k equally-sized intervals denoted I_i^l for $0 \leq i \leq k-1$. At a node n, I_i^l is defined as:

$$I_i^l = [n \oplus i\frac{N}{k^l}, n \oplus (i+1)\frac{N}{k^l}[, \;\; i \in \{0, 1, .., k-1\}, \; l \in \mathcal{L}$$

Each node, n, maintains a responsible node for every interval in its routing table. For any level, $l \in \mathcal{L}$ the responsible for interval I_0^l is always n itself. [1] For all other intervals $j \in \{1, 2, .., k-1\}$, the responsible for interval I_j^l is chosen to be the first node encountered, moving in clockwise direction, starting at the beginning of the interval. We shall use the function $R(I)$ to denote the id of the responsible node for interval I.

In addition to storing a routing table, each node, n, maintains a *predecessor* pointer, that is the first node encountered, moving in counter-clockwise direction, starting at n.

An important property of a $\mathcal{DKS}$ system is that when a node n joins or leaves the system, only n's predecessor and successor are explicitly updated in a fault-free context. The rest of the nodes in the system will find out about n existence or departure by the correction-on-use technique described in section 2.4.

Figure 1 shows an example of a $\mathcal{DKS}$ network from one node's point of view. Note that in figure 1 we have mapped the modulo N circle onto a line from node 21's view.

2.3 Lookups

To initiate a search for a key identifier id at a node n the distributed lookup is performed as follows. If id is between n's predecessor and n, the key/value pair is stored at n itself and can be resolved locally at n.

Otherwise, n searches its routing table at level $l = 1$, for an interval I_i^l in V_l such that $id \in I_i^l$, for $0 \leq i \leq k-1$. The lookup request is thereafter forwarded to the responsible node for interval I_i^l with the parameters l and i piggybacked.

A node n' upon receipt of the forwarded request checks if the key identifier id is between its predecessor and itself. If so, then n' returns the value associated with id to n. Otherwise, it searches its routing table at level $l + 1$ for an interval that contains id. Then a lookup request is forwarded to the responsible for that interval. The current level and interval are again piggybacked in the forwarded request. This process repeats until the node storing the key

[1] The responsible node's identifier and network address is stored such that communication can be established with it.

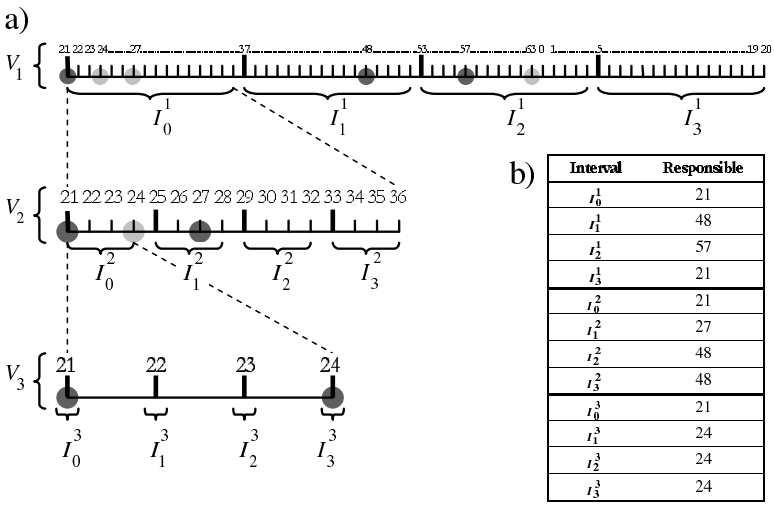

Figure 1: a) A $\mathcal{DKS}$ network with $k = 4$ and $N = 64$, with the nodes 21, 24, 27, 48, 57, and 63 present. The figure shows node 21's views, V_1, V_2 and V_3, and how each view is partitioned into $k = 4$ equally sized intervals. The dark nodes represent the responsible nodes from node 21's view. b) Node 21's routing table showing each interval and its responsible node.

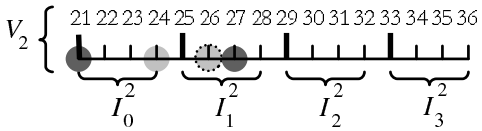

Figure 2: A node with identifier 26 joins the network depicted in figure 1. As node 21 is not the predecessor of node 26, it will not immediately be informed about node 26's existence. Hence it will continue to, erroneously, consider node 27 as responsible for I_1^2. If node 21 sends a lookup message to node 27, node 21 will find out about node 26's existence by correction-on-use. Alternatively, node 21 will become aware of node 26's existence if node 26 sends a lookup message to node 21.

id is found, in which case the value associated with id is recursively sent back to n.

2.4 Correction-on-use

In a $\mathcal{DKS}$ network, routing information can become outdated as a result of joining or leaving nodes. Figure 2 shows how routing entries become outdated as a result of a join operation. The outdated routing entries are corrected only when they are used. As long as the ratio of lookups to joins, leaves, and failures is high, the routing information are eventually corrected. This is the essential assumption in $\mathcal{DKS}$, which is validated in [1].

Correction-on-use is based on two ideas. The first idea is to embed the level, l, and the interval, i, parameters with every lookup or insertion message. A node n receiving a lookup or insertion message from a node n' can then calculate the start of the interval, I_i^l at node n', for which n is responsible according to the node n'. If n's predecessor is in the interval $[n' \oplus i\frac{N}{k^l}, n[$, then node n notifies the node n' of the existence of n's predecessor. Node n' can then update its erroneous routing entry.

The second idea is that a message sent by a node p to another node n is an indication that p exists and is thus part of the $\mathcal{DKS}$ network. Hence, node n examines all of its intervals to determine if p should be responsible for any of the intervals, in which case routing information is updated.

3 The broadcast algorithms

3.1 Desired properties

The broadcast algorithms should have the following desirable properties:

- *Coverage*. All the nodes present in the system, at the time a broadcast operation starts, receive the broadcast message as long as they remain in the system.
- *Redundancy*. Any node that receives a broadcast message receives it once, disregarding messages sent trough erroneous pointers as they will trigger correction-on-use.
- *Correction of routing information*. The broadcast algorithms should contribute to the correction of outdated routing information.

3.2 Informal description

The basic principle of the two broadcast algorithms is as follows. A node starting the broadcast iterates through all levels in $\mathcal{L}$ starting at the first level. At each level, the node moves in counter-clockwise direction through all of its intervals, broadcasting a message to each responsible node. Each broadcast message, sent by a node n, carries with it the parameters *l*, *i* and *limit*. The message's purpose is twofold. First, it delivers the intended data to the receiving node. Second, it serves as a request to a receiving node

$R1_1$ :: **receive**(u, n, **BCASTREQUEST**($data$))
send($n : n :$ **BCAST**($data, 1, 0, n$)

$R2_1$ :: **receive**(n', n, **BCAST**($data, l, i, limit$))
if $n' \oplus i\frac{N}{k^l} \in]predecessor, n]$ **then**
%% Deliver the message to the application layer
for $\lambda := 1$ **to** $log_k(N)$ **do**
for $\tau := k-1$ **downto** 1 **do**
if $R(I_\tau^\lambda) \in]n, limit[$ **then**
send(n, R(I_τ^λ), **BCAST**($data, \lambda, \tau, limit$))
$limit := n \oplus \tau\frac{N}{k^\lambda}$
fi
od
od
else
send(n, n', **BADPOINTER**(**BCAST**($data, l, i, limit$), $predecessor$))
fi

$R3_1$ :: **receive**(n', n, **BADPOINTER**(**BCAST**($data, l, i, limit$), $candidate$)
for $\lambda := 1$ **to** $log_k(N)$ **do**
for $\tau := k-1$ **downto** 1 **do**
if $n \oplus \tau\frac{N}{k^\lambda} \in]n, candidate]$ **and** $R(I_\tau^\lambda) \in]candidate, n]$ **then**
$R(I_\tau^\lambda) = candidate$
fi
od
od
send(n, $candidate$, **BCAST**($data, l, i, limit$))

Figure 3: Algorithm 1

$R2_2$ :: **receive**(n',n,**BCAST**($data, l, i, limit$))
if $n' \oplus i\frac{N}{k^l} \in]predecessor, n]$ **then**
%% Deliver the message to the application layer
for $\lambda := 1$ **to** $log_k(N)$ **do**
for $\tau := k-1$ **downto** 1 **do**
if $R(I_\tau^\lambda) \in]n, limit[$ **then**
$(i', l') :=$FINDLOWEST($n, R(I_\tau^\lambda)$)
send(n, R(I_τ^λ), **BCAST**($data, i', l', limit$))
$limit := n \oplus i'\frac{N}{k^{l'}}$
fi
od
od
else
send(n,n',**BADPOINTER**(**BCAST**($data, l, i, limit$), $predecessor$))
fi

Subroutine :: FINDLOWEST(n', r)
for $\lambda := 1$ **to** $log_k(N)$ **do**
for $\tau := k-1$ **downto** 1 **do**
if R(I_τ^λ) = r **then**
$(l', i') := (\lambda, \tau)$
fi
od
od
return (l', i')

Figure 4: Algorithm 2. The rules $R1_2$ are $R3_2$ are the same as rules $R1_1$ and $R3_1$ in figure 3.

to cover all nodes in the interval $]n \oplus i * \frac{N}{k^l}, limit[$. Each node, receiving the broadcast message, repeats the mentioned process, but makes certain not to broadcast to a node beyond the $limit$ given to it.

To illustrate the principle of the proposed algorithms, a fully populated $\mathcal{DKS}$ network with $N = 16$ and $k = 4$ is considered. A broadcast initiated at node 0 proceeds level by level. Beginning at level one, node 0 sends a broadcast message to node 12 giving it responsibility to cover the interval $]12, 0[$. Thereafter it repeats the same procedure for I_2^1 giving node 8 responsibility for the interval $]8, 12[$. After sending a broadcast to interval I_1^1 the algorithm moves to level two, repeating the process for the intervals I_3^2, I_2^2, and I_1^2. Each of the responsible nodes receiving the message from node 0 will repeat a similar process except they will not go beyond the limits assigned to them. For example node 12 will not send, at level one, to its intervals I_3^1, I_2^1, I_1^1 as they are beyond the given limit 0. Instead, it will move to level two, sending a broadcast to the nodes responsible for intervals I_3^2, I_2^2, and I_1^2.

3.3 Formal description

In both algorithms we assume a distributed system modeled by a set of nodes communicating by message passing through a communication network that is: (i) Connected, (ii) Asynchronous, (iii) Reliable, and (iv) providing FIFO communication.

A distributed algorithm running on a node of the system is described using rules of the form:

$$R :: \frac{\textbf{receive}(\textit{Sender, Receiver,} \text{MESSAGE}(arg_1, .., arg_n))}{\text{Action}}$$

The rule R describes the event of receiving a message MESSAGE at the *Receiver* node and the action taken to handle that event. A *Sender* of a message executes the statement **send**(*Sender, Receiver,* MESSAGE($arg_1, .., arg_n$)) to send a message to *Receiver*.

The first algorithm The first broadcast algorithm is given by figure 3. Rule $R1_1$ describes the reaction of a $\mathcal{DKS}$ node upon receipt of a BCASTREQUEST($data$) from the application layer. Rule $R1_1$ triggers rule $R2_1$ with the parameters $l = 1$, $k = 0$, and $limit$ set to the initiating node's id, giving the initiating node responsibility to cover all nodes in the system.

When a broadcast is initiated, the algorithm proceeds level by level. At each level, the node iterates all intervals from $k-1$ down to 1 and sends a message to the responsible node for each of the intervals. To avoid sending duplicate messages to nodes responsible for several intervals, a message is only sent when the id of the responsible node is not beyond the end of the interval checked for.

Due to outdated routing table entries some intervals might not seem to have any nodes even though they are populated. The responsibility of covering those intervals is delegated to the next interval in the iteration. This is done by not changing the $limit$ parameter when an interval seem to be unpopulated.

Improving the correction of the routing information In order to improve the correction of outdated routing information, we extend Algorithm 1 with *self-correction*. The idea consists of extending the responsibility assigned to a

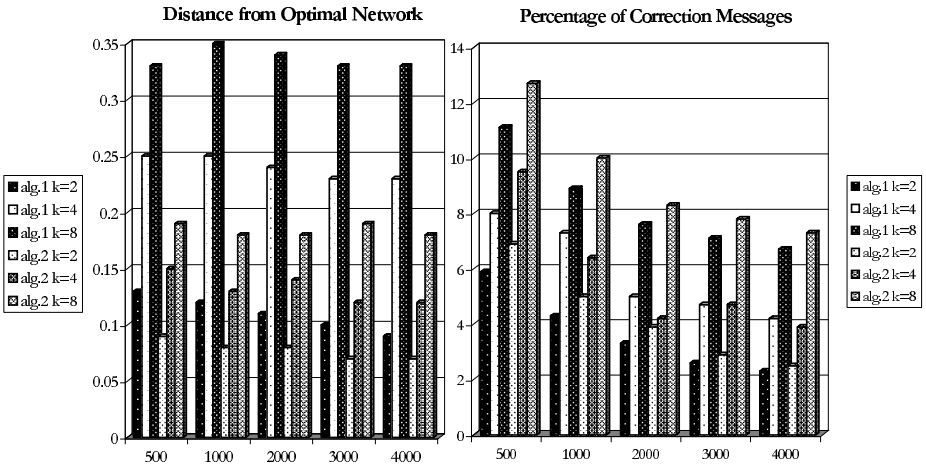

Figure 5: Experiment 1: a) Shows the distance from the optimal network b) Shows the percentage of correction messages

node n', by a node n, to cover other preceeding intervals that n' is responsible for according to n. Hence, if other nodes exist in n''s preceeding intervals, which n is not aware of, n' will trigger correction-on-use and the routing information will be corrected at n. The subroutine FINDLOWEST is used for this purpose.

The second broadcast algorithm is the same as the first algorithm, except that rule $R2_1$ is replaced by rule $R2_2$ as shown in figure 4.

4 Simulation Results

In this section we show preliminary simulation results for the broadcast algorithms. We use the following four metrics for evaluation. *Coverage*, *Redundancy*, *Correction Cost* and *Distance from Optimal Network*.

The *Coverage* and the *Redundancy* metrics are calculated by taking a snapshot of all the nodes present in the overlay network at the initiation time of each broadcast. The simulator then maintains a counter for each node receiving the broadcast message. The coverage is calculated by counting the percentage of nodes in the snapshot that received the broadcast message by the end of the simulation. The redundancy is computed by counting the number of covered nodes that received the message more than once. *Correction Cost* is defined as the percentage of messages used for correction of routing entries out of the total number of messages generated by a broadcast. *Distance from Optimal Network* is the ratio of the number of erroneous routing entries in all nodes to the total number of routing entries in the system. When this ratio is equal to 0 the routing information is said to be optimal.

The experiments were conducted on a stochastic discrete distributed-algorithms simulator developed by our team and using the Mozart [6] programming platform. In this paper we present the results of two experiments. The purpose of the first experiment was to test the system in a dynamic setting and evaluate the performance of our algorithms using the mentioned metrics. The second experiment focused on the convergence towards a minimal distance from the optimal network.

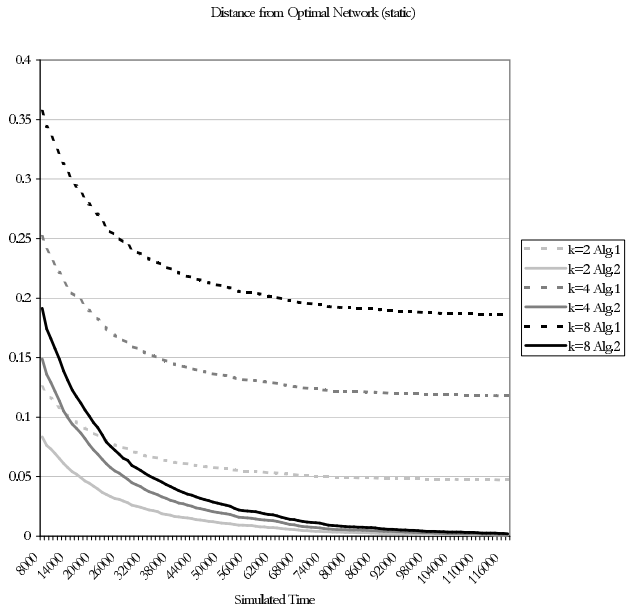

Figure 6: Experiment 2: Shows the convergence to a maximally optimal network while performing broadcasts with algorithm 1, 2.

Experiment 1. A *DKS* network of size $N = 2^{12}$ was created. The population of nodes in the system was considered a variable P that took values from $\{500, 1000, 2000, 3000, 4000\}$. For each value of P, we proceeded in two steps. First, we initialized the system with 10% of P. Second, 90% of P nodes joined while P broadcasts were initiated. The experiment was repeated for the values of $k = 2, 4, 8$. That is, with a high probability, each node initiated one broadcast while the overlay network was growing.

Experiment 2. A *DKS* network of size $N = 2^{12}$ was created. The system was initialized with 1500 nodes. Thereafter an arbitrary number of broadcasts were initiated. The experiment was repeated for the values of $k = 2, 4, 8$.

Results. In all our experiments, the *Coverage* and *Redundancy* were 100% and 0 respectively as expected from the design.

Distance from Optimal Network. Two observations can be made from Figure 5 a). First, for all values of k, Algorithm 2 corrects routing information more effectively than Algorithm 1. Second, the final distance from the optimal network is mainly affected by the search arity, k, and not the population size. From Figure 6 we can see that algorithm 2, in contrast to algorithm 1, effectively converges to the optimal network for all search arities.

Correction Cost. As shown in Figure 5 b), the correction cost is in general higher for Algorithm 2. This was expected as the correction requires some additional overhead.

5 Conclusion

In this paper we presented two algorithms for broadcasting on structured peer-to-peer networks. Our work was motivated by two reasons. First, the need to extend distributed hash tables to perform arbitrary queries and retrieval of global statistical information about the DHTs. Second, to provide robust algorithms that can be used for multicasting within groups in the context of $\mathcal{DKS}$ overlay networks. Each group is formed by creating a specific $\mathcal{DKS}$ instance for it.

The proposed algorithms use the $\mathcal{DKS}$ philosophy of avoiding periodic stabilization to maintain routing information. The second algorithm extends the philosophy by heavily correcting incorrect routing information.

In addition, the broadcast algorithms provide full coverage even if nodes have erroneous routing information. Furthermore, each broadcast message is received once even if new nodes join while the broadcasts are taking place.

The proposed algorithms have been validated and evaluated in a dynamic network through simulations and the obtained results confirm our expectations. More precisely, Algorithm 1 gives less correction overhead and larger distance from the optimal network compared to Algorithm 2.

References

[1] L. O. Alima, S. El-Ansary, P. Brand, and S. Haridi. DKS(N, k, f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In *The 3rd International workshop on Global and Peer-To-Peer Computing on large scale distributed systems - CCGRID2003*, Tokyo, Japan, May 2003.

[2] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications*, 2002.

[3] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient Broadcast in Structured P2P Netwoks. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, February 2003.

[4] M. Harren, J. M. Hellerstein, R. Huebsch, B. T Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *The 1st Interational Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[5] P. Maymounkov and D. Mazires. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *The 1st Interational Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[6] Mozart Consortium. http://www.mozart-oz.org, 2003.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. Technical Report TR-00-010, Berkeley, CA, 2000.

[8] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level Multicast using Content-Addressable Networks. In *Third International Workshop on Networked Group Communication (NGC '01)*, 2001.

[9] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218, 2001.

[10] I. Stoica, D. Adkins, S. Ratnasamy, S. Shenker, S. Surana, and S. Zhuang. Internet Indirection Infrastructure. In *The 1st Interational Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[11] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM 2001*, pages 149–160, San Deigo, CA, August 2001.

[12] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. Technical Report TR-819, MIT, January 2002.

[13] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. U. C. Berkeley Technical Report UCB//CSD-01-1141, April 2000.