Multicast in $\mathcal{DKS}(N, k, f)$ Overlay Networks [★]

Luc Onana Alima¹, Ali Ghodsi¹, Per Brand², and Seif Haridi¹

¹ IMIT-Royal Institute of Technology (KTH)
{onana,aligh,seif}@imit.kth.se,

² Swedish Institute of Computer Science (SICS)
perbrand@sics.se

Abstract

Recent developments in the area of peer-to-peer computing show that structured overlay networks implementing distributed hash tables scale well and can serve as infrastructures for Internet scale applications.

We are developing a family of infrastructures, $\mathcal{DKS}(N, k, f)$, for the construction of peer-to-peer applications. An instance of $\mathcal{DKS}(N, k, f)$ is an overlay network that implements a distributed hash table and which has a number of desirable properties: low cost of communication, scalability, logarithmic lookup length, fault-tolerance and strong guarantees of locating any data item that was inserted in the system.

In this paper, we show how multicast is achieved in $\mathcal{DKS}(N, k, f)$ overlay networks. The design presented here is attractive in three main respects. First, members of a multicast group self-organize in an instance of $\mathcal{DKS}(N, k, f)$ in a way that allows co-existence of groups of different sizes, degree of fault-tolerance, and maintenance cost, thereby, providing flexibility. Second, each member of a group can multicast, rather than having single source multicast. Third, within a group, dissemination of a multicast message is optimal under normal system operation in the sense that there are no redundant messages despite the presence of outdated routing information.

Key words: Peer-to-Peer Computing, Distributed Algorithms, Multicast, Distributed Hash Tables.

1 Introduction

The need for making effective use of the huge amount of computing resources attached to large scale networks such as the Internet is driving research in peer-to-peer (P2P) computing. Recent developments in this area show that structured overlay networks implementing distributed hash tables scale well and can serve as infrastructures for P2P applications. [1, 9, 8, 2].

In an effort to contribute to this new trend, we are designing and developing a family of infrastructures termed $\mathcal{DKS}(N, k, f)$ [1], for constructing P2P applications. Each instance of $\mathcal{DKS}(N, k, f)$ is an overlay network that implements a distributed hash table and which has a number of desirable properties including low cost of communication due to a novel technique for maintaining routing information, scalability, logarithmic lookup length, fault-tolerance and high guarantees of locating any data item that was inserted in the system.

[★] This work was funded by the European project IST-2001-32234, PEPITO and Vinnova PPC project in Sweden.

1.1 Motivations and contributions

Multicast is an important primitive for many distributed applications, such as content distribution and video conferencing. Traditional approaches for multicast work well in small scale systems. For large scale systems with extreme dynamism, such as P2P systems, the multicast problem must be addressed in a different fashion that enables creation of scalable, fault-tolerant groups in which multicast messages are delivered to all group members without any redundant messages. How to achieve this in the $\mathcal{DKS}(N, k, f)$ family is the main contribution of this paper.

Multicast in large scale and highly dynamic environments such as peer-to-peer environments has been addressed according to two main approaches. In the first approach exemplified by [3], the idea is to construct a multicast dissemination tree. However, this approach has at least two drawbacks. First, non-member nodes of a multicast group maintain information about the multicast dissemination tree. Furthermore, the number of messages sent for a given multicast is, in the worst case, in the order of the actual size of the underlying overlay network (i.e., Pastry). In the second approach exemplified by [6] a new CAN is created for the multicast group. But the dissemination of multicast messages does not ensure zero redundancy, because of the possibility of imperfect partitioning of the coordinate space.

The design presented in this paper follows the second approach as in [6], although it differs in some respects significantly. The main contributions of our design are: (i) Members of a multicast group self-organize in an instance of $\mathcal{DKS}(N, k, f)$ in a way that allows co-existence of groups of different sizes, degree of fault-tolerance, and maintenance cost, thereby, providing flexibility. (ii) As in [6], each member of a group can multicast, rather than having single source multicast like in [3]. (iii) In contrast to [6], within a group, dissemination of a multicast message is optimal under normal system operation in the sense that there are no redundant messages despite the presence of outdated routing information. This is achieved thanks to a “correcting broadcast” algorithm inspired by our previous work [4].

1.2 Roadmap

The rest of the paper is structured as follows. In Section 2, we give an overview of the preliminary design of the $\mathcal{DKS}(N, k, f)$ family. In section 3, we discuss how multicast groups are created, how nodes join a multicast group and present the algorithm used to transmit a multicast message to all members of a group without any redundancy. Thereafter we validate and evaluate our multicast scheme in section 4. Finally, Section 5 summarizes the paper.

2 Overview of $\mathcal{DKS}(N, k, f)$ design

In this section, we recall the key ideas behind the preliminary design of the $\mathcal{DKS}(N, k, f)$ family [1].

2.1 Underlying assumptions

For the design of the $\mathcal{DKS}(N, k, f)$, we model a distributed system as a set of *processes* linked together through a *communication network*. Processes communicate by message passing. The communication network is assumed to be *connected*, *asynchronous*,

reliable, and *FIFO*. By reliable, we mean that every message injected into the communication network eventually is delivered to its destination, provided that the latter remains connected to the communication network.

To set up a $\mathcal{DKS}(N, k, f)$ system, it is assumed that: k is an integer greater or equal 2. The maximum number of nodes that can be in the system is $N = k^L$ where L is assumed to be large enough to achieve very large distributed systems. f is the fault-tolerance parameter. Each peer knows the parameters N , k and f , thus can compute L .

2.2 The identifier space and notations

In designing $\mathcal{DKS}(N, k, f)$ systems, we assume, like in most P2P infrastructures [9, 8], that nodes of a $\mathcal{DKS}(N, k, f)$ and objects managed by these nodes are uniquely identified by identifiers taken from the same logical space.

In this paper, we assume that the identifier space, denoted $\mathcal{I} = \{0, 1, \dots, N - 1\}$ is organized as a ring. We use for $a, b \in \mathcal{I}$, $a \oplus b$ for $(a + b)$ modulo N .

Next in the paper, we shall need to determine whether a given identifier belongs, going in the clockwise direction, to a part (or interval) of the identifier space. For this reason, we use an appropriate boolean function $\hat{\in}$, which will serve that purpose. For simplicity of notation, we shall use infix notation for the function $\hat{\in}$.

2.3 Key/value pairs management

A $\mathcal{DKS}(N, k, f)$ overlay network implements a distributed hash table that serves for efficient storage and retrieval of key/value pairs. Let (t, v) be a key/value pair, where t is a key and v is the value associated with t . When inserted in a $\mathcal{DKS}(N, k, f)$ network, this pair is stored at the first node met moving on the identifier space starting from the hashed identifier of t , in the clockwise direction.

2.4 Levels and views

Any $\mathcal{DKS}(N, k, f)$ network is built in a manner that allows each lookup to be resolved by following a path of a virtual k -ary spanning tree of height $\log_k(N)$ ensuring logarithmic lookup path length. To achieve this, each node in a $\mathcal{DKS}(N, k, f)$ network has $\log_k(N)$ levels numbered from 1 to L , where $L = \log_k(N)$. From now on, we shall use $\mathcal{L}$ for $\{1, 2, \dots, L\}$.

When at level $l \in \mathcal{L}$, a node n has a *view* V^l of the identifier space. The view V^l consists of k equal parts, denoted I_i^l , $0 \leq i \leq k - 1$, and defined below level by level. Next, we use $\mathcal{K}$ for $\{0, 1, \dots, k - 1\}$.

At level 1: $V^1 = I_0^1 \cup I_1^1 \cup I_2^1 \cup \dots \cup I_{k-1}^1$, where $I_0^1 = [x_0^1, x_1^1[$, $I_1^1 = [x_1^1, x_2^1[$, $\dots$, $I_{k-1}^1 = [x_{k-1}^1, x_0^1[$, $x_i^1 = n \oplus i \frac{N}{k}$, for $0 \leq i \leq k - 1$.

At level $2 \leq l \leq L$: $V^l = I_0^l \cup I_1^l \cup I_2^l \cup \dots \cup I_{k-1}^l$, where $I_0^l = [x_0^l, x_1^l[$, $I_1^l = [x_1^l, x_2^l[$, $\dots$, $I_{k-1}^l = [x_{k-1}^l, x_0^{l-1}[$, $x_i^l = n \oplus i \frac{N}{k^l}$, for $0 \leq i \leq k - 1$.

2.5 Responsibilities

Let n be an arbitrary $\mathcal{DKS}(N, k, f)$ node. Let l , $1 \leq l \leq L$, and let $V^l = \cup_{0 \leq j \leq k-1} I_j^l$ be the view that node n has at level l . With respect to node n , for each I_j^l , $0 \leq j \leq k-1$, there is a node $R(I_j^l)$ that node n considers *responsible* for I_j^l . Intuitively, the responsible for I_j^l represents the node that n will contact, for example, when trying to resolve a key identifier that belongs to I_j^l .

We denote by $S(x)$, for an identifier x , the successor³ of x .

For an arbitrary node n and an arbitrary level $l \in \mathcal{L}$, the responsible for I_i^l , $i \in \mathcal{K}$ is $S(x_i^l)$. Each node is itself responsible for I_0^l for any $1 \leq l \leq \log_k(N)$.

2.6 Routing information

To ensure logarithmic ($\log_k(N)$) lookup length, each node n of a $\mathcal{DKS}(N, k, f)$ system maintains a routing table RT , which is of type $RT: \mathcal{L} \rightarrow (\mathcal{K} \rightarrow \mathcal{I})$. Hence, $(RT(l))(i)$ denotes the responsible for the interval I_i^l . Sometimes, for the sake of clarity, we write RT_n , to emphasize that the routing table under consideration is that of node n .

In addition to the above routing table, each node n maintains a pointer denoted p , to its predecessor on the ring. The predecessor of a node n is the first node met when moving in the counterclockwise direction starting from n . So, in total, each node needs only $(k-1)\log_k(N) + 1$ for the purpose of the lookup.

2.7 $\mathcal{DKS}(N, k, f)$ networks construction and maintenance

A $\mathcal{DKS}(N, k, f)$ is built as described in [1]. The core idea is that the join and leave operations are handled by local atomic actions that ensure a consistent interleaving of concurrent joins, leaves and other operations, such as multicast and broadcast. Typically, these local atomic operations involve very few nodes.

Given that concurrent joins at different parts of the circle can take place by the same time, and that when a node joins it obtains an approximate routing table, it is possible that routing entries can be out of date.

Handling such out of date information is the distinguishing property of the $\mathcal{DKS}(N, k, f)$ infrastructures. Indeed, rather than using separate stabilization mechanism to be run periodically, the $\mathcal{DKS}(N, k, f)$ design adopts another approach in which erroneous or out-of-date routing entries are detected and corrected on-the-fly. We intuitively present the technique used in the next subsection.

We refer the reader to [1] for the handling of failures.

2.8 Correction-on-use

The correction-on-use technique builds on two simple observations.

Observation 1: By piggybacking the level and the interval information in lookup or insert messages, a remote node n' can determine upon receipt of a message from n , whether the routing entry used by n to send the received message was correct.

³ The successor of an identifier t is the first node encountered, moving in the clockwise direction, starting at t .

Observation 2: Upon receipt of a message from a remote node n' , a node n becomes aware of the existence of n' , and can therefore determine whether it should consider n' as responsible some of its routing entries.

To exploit the first observation, each node n maintains a *back list*⁴, which consists of at most the first f nodes encountered in the counter-clockwise direction, starting at node n . We recall by the definition of responsibilities, that at a node n , for any l ($1 \leq l \leq \log_k(N)$), and any $i \in \mathcal{K}$, $(RT_n(l))(i) = S(n \oplus i \frac{N}{k^l})$. This invariant will hold in any configuration of the system where each node has correct routing information. Therefore, if a node n sends the level (i.e., l) and the interval information (i.e., i) while sending message to a node n' , then node n' upon receipt of the message can determine whether the entry used by n was correct. With respect to node n' , the entry used by n to send a message carrying l and i , is correct *only if* node n' is the successor of $(n \oplus i \frac{N}{k^l})$. So, node n' , using its back list, can determine whether a node in its back list is a better approximate successor for $n \oplus i \frac{N}{k^l}$.

When a node n' detects that there is a better approximate successor for $n \oplus i \frac{N}{k^l}$, node n' sends a BADPOINTER message to node n . This message serves to inform node n that the entry it used to contact n' is outdated. In addition, the BADPOINTER message carries the address of a node C , that node n' believes is the best candidate for correcting the routing entry used by n . The best candidate C is selected from the *back list* of n' .

When node n receives such a BADPOINTER message, it updates its routing table and repeats the operation (e.g., lookup request, key/value pair insertion request, broadcast) that led it to contact n' , but now the message is sent to C .

The exploitation of the second observation is immediate. Indeed, when a node n receives a message from a remote node n' , node n checks its routing table to determine whether it should have n' in its routing table. If this is the case, node n updates its routing table accordingly.

2.9 Lookup in a $\mathcal{DKS}(N, k, f)$

The protocol for resolving keys in a $\mathcal{DKS}(N, k, l)$ overlay network is an interval routing protocol that serves not only for resolving keys, but also for correcting outdated routing entries. As we have already sketched the idea of detection and correction of outdated routing entries, in this section, we only present briefly how keys are resolved.

When a node n receives a lookup request for key identifier t , from its user, node n checks if t is between its predecessor, p , and itself. If this is the case, node n does a local lookup to find the value associated to t . The result is returned to the user. Otherwise, node n triggers a forwarding process that goes level by level, and that consists in routing lookup messages toward the successor of t . Each lookup message carries necessary information (level and interval) for detection and correction of outdated routing entries. When the node n' , successor of t is reached, node n' performs a local lookup to retrieve the value associated to t . The result is forwarded backward on the reverse path to the origin of the lookup.

⁴ The *back list* component is one of the enhancements we have made on the preliminary design.

Inserting key/value pairs in the system is similar to the lookup. In addition, messages induced by key/value pairs insertion are also used for detection and correction of routing entries.

3 Multicast in $\mathcal{DKS}(N, k, f)$

In this section, we show how groups are created and managed in $\mathcal{DKS}(N, k, f)$ overlay networks. We first give the underlying principle in Subsection 3.1. Then in Subsection 3.2 we explain the creation of groups. Subsection 3.3 gives an explanation of how nodes join a given group. Finally in Subsection 3.5, we present a “correcting broadcast” that serves to forward multicast messages in an optimal way within a group.

3.1 The principle

The multicast in $\mathcal{DKS}(N, k, f)$ is inspired from [7]. However, our design differs from the one given in [7] in many aspects. First, in [7], the identifier space is a d -dimensional space while in $\mathcal{DKS}(N, k, f)$, the identifier space is a ring. Second, CAN relies on active correction for maintaining routing information while $\mathcal{DKS}(N, k, f)$ uses correction-on-use. Third, the broadcast algorithm used for multicast message forwarding in CAN is not optimal, as it does not eliminate redundant messages under normal system operation.

Our design borrows from [7], the idea of creating new instances of $\mathcal{DKS}(N, k, f)$ overlay networks for multicast groups. The idea is as follows. Let O be an instance of $\mathcal{DKS}(N, k, f)$. A node in O can decide to create a multicast group identified by g . Members of g self-organize in an instance G_g of $\mathcal{DKS}(N, k, f)$. The instance G_g is created and maintained as the underlying overlay network O . Information about group membership is distributed over the members of the group, as each group is represented by a specific instance of a $\mathcal{DKS}$.

To multicast a message within a group g from a member node n , the multicast message is handed out to n . Then node n simply starts a “correcting broadcast” process that serves to forward the message only to members of g . The challenging part is to design the “correcting broadcast” for forwarding multicast messages while ensuring that under normal system operation, there are no redundant messages. The broadcast must be “correcting”, because in a $\mathcal{DKS}(N, k, f)$ overlay network representing a group, members can have erroneous routing information due to the way nodes are inserted in the group, and also because of leave or failure of group members.

3.2 Group creation

Let O be an instance of $\mathcal{DKS}(N, k, f)$. We write N_O , k_O and f_O the values of the parameters N , k and f for the creation of the overlay network O . Let n be a node of O and assume that node n wants to create a new group identified by g . Then, node n first determines the characteristics of the group g . These include:

- k_g : the arity parameter within the group. This parameter serves for the construction of the topology of the overlay representing the group g .

- N_g : a power of k_g that denotes the maximum number of members that the group g can have. This number serves to determine the ring onto which group members are mapped.
- f_g : the fault-tolerance parameter within the group g . This parameter serves also to determine the size of the back list.
- H_g : the hash function used to map nodes in O onto the ring of maximum size N_g .

Then, once these parameters are set up, node n inserts itself as the first node of the group. The characteristics of the group, g , and the address of node n are made available such that other nodes of O interested in g can join the group g . The joining of a group deserves some explanations that we give in the next subsection.

3.3 Joining a group

A node n_j that wants to join the group g must know the characteristics of g and at least one node that belongs to g .

Assume node n_j knows the characteristics of a group g and some node n , member of g . Then, using H_g , node n_j computes its potential position, a number in $\{0, \dots, N_g - 1\}$, in the overlay network associated to group g . Then node, n_j starts a join process similar to the one used to insert nodes in O , except that now, because N_g can be small compared to N_O , we must resolve collisions during insertion. In the current design, we adopt the “first come first in” policy. As a consequence, it is possible for some nodes to continuously be rejected due to collisions when they want to join a group g . Addressing this issue is the subject of a future work.

Once a node n_j is inserted in a group g , node n_j makes available the characteristics of g and the addresses of some nodes it knows are in g . This way, the group information diffuses.

3.4 Illustration

The principle of the multicast group in $\mathcal{DKS}(N, k, f)$ overlay networks is illustrated in Figure 1. In this figure, panel *a*) shows a $\mathcal{DKS}(N = 32, k = 2, f = 3)$ instance in which only nodes with identifiers 4, 11, 15, 19, 25 and 28 are present. Let us call this instance O . Panel *b*) shows a $\mathcal{DKS}(N = 16, k = 4, f = 2)$ representing a multicast group with identifier $g = 4$, in which nodes 28, 25, 19 and 15 of O are members. Note that because of the hashing function for the multicast group $g = 4$, denoted H_4 , each of the nodes 28, 25, 19 and 15 receives a new identifier relatively to this multicast group. For example, node 28 receives 1 as its identifier in the multicast group $g = 4$. Panel *c*) shows a $\mathcal{DKS}(N = 8, k = 2, f = 1)$ representing a multicast group with identifier $g = 2$ in which nodes 4, 11 and 15 of O are members. Note that each node of O that is member of the multicast group $g = 2$ receives a new identifier relatively to this group.

3.5 Correcting broadcast

The main difficult part of multicasting to groups in $\mathcal{DKS}(N, k, f)$ is to achieve optimal forwarding of messages. To overcome this difficulty, we build on our previous work on

Fig. 1. Two groups based on a $DKS(32, 2, 3)$.

broadcasts [4], which we adapt to fit our need for multicast. The proposed algorithm allows multicast messages to be delivered to all members of a group without any redundant messages. In addition, it allows members of a group to correct their routing information on-the-fly, leading to efficient use of the bandwidth. From this on-the-fly correction of erroneous routing information comes the name “correcting broadcast”.

We shall proceed in two steps. First, we give the principle of the correcting broadcast in Subsection 3.5. Next, we present a formal description of the algorithm in Subsection 3.5.

The principle Each node n maintains a mapping GT_n that takes a group identifier, g , then returns N_g , k_g , f_g and the routing table, RT_g , of n for the overlay network associated to g . For practical reasons, we assume that each node can be in at most $MaxGroup$ groups.

Let u be the application layer at a node n , member of a group g . When u wants to multicast an information x , u sends a $BCASTREQUEST(x, g)$ message to n .

Upon receipt of $BCASTREQUEST(x, g)$, node n starts a broadcast process using the routing table associated to group g . In this broadcast process, node n conceptually moves in the counter clockwise direction to deliver a $BCAST(x, g, l, i, Limit)$ to each node n' considers responsible by n . The parameters of the BCAST message are: x , the information to be multicast. l and i serve for the correction-on-use. Actually, l and i represent the interval for which node n considers n' responsible for. The

Limit parameter, serves the receiving node n' for determining which part of the ring associated to g , node n' should cover during the broadcast process.

When a node n receives a $\text{BCAST}(x, g, l, i, L)$ from a node n' , node n first adjusts (if necessary) its routing table for the group g to reflect the presence of n' in the overlay network associated to g ⁵. Second, node n checks whether it is actually the node that n' should consider responsible for the interval represented by l and i . If this is the case, node n delivers the information x to its application layer. Then, node n conceptually moves in the counter clockwise direction to deliver a $\text{BCAST}(x, g, l, i, \text{Limit}_x)$ to each node x that n considers responsible within the interval $]n, L[$. The Limit_x sent to a node x is computed such as to guarantee that the intervals covered by two arbitrary nodes are disjoint. This is the key to avoid redundant messages.

If node n finds out that it shouldn't be considered responsible by node n' , then node n sends an $\text{BADPOINTER}(\text{BCAST}(x, g, l, i, L), C)$ message to n' . This message serves as a notification to n' that the information it has about interval represented by l and i is erroneous. In addition, the parameter C in the message $\text{BADPOINTER}(\text{BCAST}(x, g, l, i, L), C)$ denotes a node that n knows and believes is the successor of $n' \oplus i \frac{N}{k^t}$.

When a node n' receives a $\text{BADPOINTER}(\text{BCAST}(x, g, l, i, L), C)$ from another node n , node n' adjusts its routing information to reflect the presence of node C , thereafter it re-sends the message $\text{BCAST}(x, g, l, i, L)$ to the new responsible for its interval i at the level l .

The formal description This subsection gives the formal description of the “correcting broadcast” under the assumption on the communication network made in Subsection 2.1. The algorithm is specified as a set of rules of the form

$$\frac{\text{Event}}{\text{ACTION}}$$

where **EVENT** is the reception of a message and **ACTION** is a sequence of statements that is executed atomically upon receipt of **EVENT**. For a node n to send a message $\text{MSG}(\text{param})$ to another node n' , node n executes the statement **send**($n : n' : \text{MSG}(\text{param})$).

We assume that each node n has a mapping GT that takes a group identifier g and returns a 4-tuple (RT_g, N_g, k_g, f_g) where RT_g is the routing table of n for the multicast group g , N_g is the maximum size of this group, k_g is the search arity within this group and f_g is the fault-tolerance degree in this group. We call GT the group table of the node. Note that in addition to GT , each node has the routing table for the underlying $\mathcal{DKS}(N, k, f)$ instance.

4 Simulation Results

To validate our multicasting scheme we have simulated the algorithm in our stochastic discrete event simulator developed by our team using the Mozart[5] programming system. We focused on the following metrics:

⁵ This action is performed for correction-on-use.

Algorithm 3.1 Correcting Broadcast

Message types:

BCASTREQUEST(x, g)
BCAST(x, g, l, i, L)
BADPOINTER(BCAST(x, g, l, i, L), C)

Variables:

GT

Rules:

$R_1 :: \text{receive}(u : n : \text{BCASTREQUEST}(x, g))$

 $(RT_g, N_g, k_g, f_g) := GT(g)$
 $Limit := n$
for $l = 1$ **to** $\log_{k_g}(N_g)$ **do**
 for $i = k_g - 1$ **downto** 1 **do**
 if $(RT_g(l))(i) \in]n, Limit[$ **then**
 send $(n : (RT_g(l))(i) : \text{BCAST}(x, g, l, i, Limit))$
 $Limit := n \oplus i \frac{N_g}{(k_g)^l}$
 fi
 od
od
□

$R_2 :: \text{receive}(n' : n : \text{BCAST}(x, g, l, i, L))$

 $(RT_g, N_g, k_g, f_g) := GT(g)$
Adjust RT_g with n'
if $n \oplus i \frac{N_g}{(k_g)^l} \in]p, n]$ **then**
 deliver x to application layer
 $Limit := L$
 for $\tau = 1$ **to** $\log_{k_g}(N_g)$ **do**
 for $j = k_g - 1$ **downto** 1 **do**
 if $(RT_g(\tau))(j) \in]n, Limit[$ **then**
 send $(n : (RT_g(\tau))(j) : \text{BCAST}(x, g, \tau, j, Limit))$
 $Limit := n \oplus j \frac{N_g}{(k_g)^\tau}$
 fi
 od
 od
else
 $C :=$ best approximate successor from back list
 send $(n : n' : \text{BADPOINTER}(\text{BCAST}(x, g, l, i, L)), C)$
fi
□

$R_3 :: \text{receive}(n : n' : \text{BADPOINTER}(\text{BCAST}(x, g, l, i, L)), C)$

Let RT_g be the routing table of n for g
Adjust RT_g with C
send $(n : C : \text{BCAST}(x, g, l, i, L))$

Fig. 2. Average Lookup Length

Fig. 3. Number of BADPOINTER messages

Redundancy, the percentage of redundant messages. *Coverage*, the percentage of those nodes that received each multicast message. *Average Lookup Length*, the average number of hops a multicast message takes from the source to each destination. *Badpointer Traffic*, the number of BADPOINTER messages induced by the correction-on-use.

The results from all the experiments showed that the multicast messages were received by every node in the group. Furthermore, no redundant messages were sent in any of these experiments.

Figures 2 and 3 compare two multicast groups, $\mathcal{DKS}(N = 8^3, k = 8, f = 5)$ and $\mathcal{DKS}(N = 2^9, k = 2, f = 5)$. Each group is initiated with 100 joining members, and thereafter 900 broadcasts are interleaved with 100 additional joins over time.

Figure 2 shows that the $\mathcal{DKS}(N, k, f)$ system can be instantiated with a larger value for k to obtain a lower average lookup length. However, as Figure 3 shows, this will come at the cost of higher number of BADPOINTER messages.

5 Concluding remarks and future work

We shown how multicast is achieved in $\mathcal{DKS}(N, k, f)$ overlay networks. The main idea lies on the fact that for each new group, a specific instance of $\mathcal{DKS}(N, k, f)$ is created and maintained exactly as for the underlying overlay network. Very large groups can be created at the cost of large amount of data for multicast group. Interestingly, each group is created with specific characteristics, such that smaller and bigger groups can co-exist. For example, powerful nodes can be in many or larger groups while less powerful node restrict themselves to smaller groups.

With a given group, each multicast message is transmitted in an optimal way under normal operation, i.e. there is no redundant message. Because of the dynamism (join, leave and failure), members of a group can have outdated routing information relatively to groups, the proposed “correcting broadcast” fixes outdated routing information on-the-fly, thus avoiding unnecessary bandwidth consumption. The proposed multicast scheme has been validated by simulations. We performed preliminary experiments that confirmed our expectations: no redundant messages are sent, and any multicast message sent to a group is received by every member. The simulations showed that the correcting broadcast effectively updates incorrect routing informa-

tion. Moreover, each group can be tuned to adjust the trade-off between the efficiency and the cost of multicasting.

Acknowledgments

We would like to thank Mr. Thomas Sjöland for correcting some typos in the preliminary version of this paper.

References

1. Luc Onana Alima, Sameh El-Ansary, Per Brand, and Seif Haridi. DKS(N, k, f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In *3rd International Symposium on Cluster Computing and the Grid - CCGRID2003*, Tokyo, Japan, May 2003.
2. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. Technical Report UMIACS-TR 2002-53 and CS-TR 4373, 2002.
3. 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.
4. A. Ghodsi, L. O. Alima, S. El-Ansary, P. Brand, and S. Haridi. Self-Correcting Broadcast in Distributed Hash Tables. In *15th IASTED International Conference, Parallel and Distributed Computing and Systems*, Marina del Rey, CA, USA, November 2003.
5. Mozart Consortium. <http://www.mozart-oz.org>, 2003.
6. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. Technical Report TR-00-010, Berkeley, CA, 2000.
7. Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level Multicast using Content-Addressable Networks. In *Third International Workshop on Networked Group Communication (NGC '01)*, 2001.
8. Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218, 2001. citeseer.nj.nec.com/rowstron01pastry.html.
9. 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.