2002-11-01

# A Theoretical Framework for the Multicast Address Allocation Problem

Daniel Zappala
daniel_zappala@byu.edu

Chris GauthierDickey

Virginia Lo

Timothy Singer



## Original Publication Citation

Virginia Lo, Daniel Zappala, Chris GauthierDickey, and Tim Singer, "A Theoretical Framework for the Multicast Address Allocation Problem", IEEE Globecom, Seventh Global Internet Symposium, November 22.

# A Theoretical Framework for the Multicast Address Allocation Problem

Virginia Lo, Daniel Zappala, Chris GauthierDickey, and Timothy Singer
Department of Computer Science, 1202 University of Oregon, Eugene OR 97403-1202
lo|zappala|chrisg|tsinger@cs.uoregon.edu

***Abstract*—The multicast address allocation problem requires Internet domains to allocate unique addresses to multicast applications from a globally-shared space. We develop a theoretical framework for multicast allocation algorithms that is influenced by subcube allocation in hypercube computer systems. Based on this framework we derive complexity results for the address allocation problem and describe several new allocation algorithms that use a hypercube model for address representation.**

## I. Introduction

The multicast address allocation problem is one of several key problems that has delayed deployment of native IP multicast throughout the Internet. While recent work in the areas of Source-Specific Multicast (SSM) and application-layer multicast protocols has side-stepped the *malloc problem*, neither has proposed fully satisfactory schemes to support any source multicast (ASM). The rapidly increasing use of the Internet for all manner of communications makes it imperative that we continue to seek support for minimum latency, maximally efficient multicast services.

In this paper, we show that the multicast address allocation problem is one instance of a well-known, general resource allocation problem in which a block of resources is allocated and de-allocated based on dynamic requests for sub-blocks of varying sizes. The ability to respond to requests under heavy loads is difficult because the resource space may become fragmented into many small non-aggregatable blocks.

The most well-known instance of this problem arises in memory management and disk space management in which contiguous bytes of memory (or physical blocks of disk space) are allocated and de-allocated over time. Other examples include distribution of zip codes and telephone numbers, and the processor allocation problem in hypercubes, tori, and meshes.

We focus our attention on the latter class of problems, specifically the subcube allocation problem in hypercubes. We show how results from subcube allocation – including its compact notation, complexity results, and algorithms – can be applied to the malloc problem to overcome the limitations of current schemes for address allocation.

We first reported the close relationship between the malloc problem and the subcube allocation problem in [1]. Here, we continue our development of a theoretical framework for the multicast address allocation problem and propose new algorithms that use a hypercube-based approach. A companion paper [2] builds on these theoretical results by studying the performance of these algorithms within the context of the MASC architecture [3].

In this paper, we develop a framework for the malloc problem by classifying address allocation algorithms according to their recognition capability: **prefix-based**, **contiguous**, and **non-contiguous**. While a prefix-based algorithm is recommended for MASC, contiguous and non-contiguous algorithms offer a more flexible representation for address blocks and hence provide a greater ability to recognize free blocks in a fragmented space.

Using this framework, we derive the first complexity results for the malloc problem and explain their implications for address allocation protocols. Until now, the malloc problem has not been studied formally. Our complexity results show that address allocation is a subtle and difficult problem, more so than heretofore understood by the networking community. Our results provide guidance towards approaches that are likely to reap practical benefits for multicast address allocation.

Finally, we develop several new polynomial time algorithms for address allocation that use a hypercube model for address aggregation. These algorithms are defined by their recognition capability and their fit model (first fit, ARBE fit, best fit, or worst fit). These new algorithms hold promise for use within the MASC architecture.

## II. Background and Terminology

We begin by describing the concept of an address expression and formally defining the malloc problem. We then show that there is a straightforward correspondence between the subcube allocation problem in hypercubes and the malloc problem.

### A. Address Expressions

An address expression is a compact notation for representing *block* or set of addresses. We use the standard *don't care* notation of hypercubes for expressions, e.g., the set of four addresses 0000, 0001, 0010, 0011 can be represented as the address expression 00XX, in which the X's represent *don't care* bits. This notation is similar to that of address masks, which are commonly used in Internet routing protocols.

We define the following taxonomy of address expressions, based on the allowable patterns of the *don't care* bits.

- **Prefix-Based**: Address expressions must have all all the *don't care* bits in the rightmost positions.
- **Contiguous**: Address expressions must haveconti guous *don't care* bits, with wraparound allowed.
- **Non-Contiguous**: Address expressions may have the *don't care* bits in arbitrary positions.

For example, given a block of $2^5$ addresses allocated from a $2^{10}$ bit address space, 00100XXXXX denotes a prefix-based address expression, 001XXXXX01 and XX00110XXX both denote contiguous address expressions, and X00XX10XX0 denotes a non-contiguous expression. Each class is contained in the next, with non-contiguous being the most general class.

This work was supported in part by the National Science Foundation under grants ANI-9977524 and NCR-9714680.

0-7803-7632-3/02/$17.00 ©2002 IEEE

### B. MASC and the Malloc Problem

We assume the widely-accepted model for interdomain multicast defined by Kumar et. al. [3] and the proposals of the IETF's MALLOC working group [4]. Under this model, domains use the Multicast Address-Set Claim (MASC) protocol to dynamically assign address blocks along the existing provider-subscriber hierarchy. A subdomain claims blocks of addresses from a parent domain in order to satisfy multicast address requests from internal applications as well as from its own child domains.

The heart of the MASC protocol lies in the scheme used for allocation and de-allocation of address blocks. This fundamental, yet difficult problem is what we refer to as the malloc problem, which can be defined as follows for a single hierarchy composed of a parent domain and $m$ child domains. The definition is easily extended to a multi-level hierarchy.

**The Malloc Problem**: A domain is given a contiguous set of $2^n$ multicast addresses, represented as binary numbers from 0 to $2^n - 1$. Initially, all addresses are available for allocation. Child domains $C_0$ through $C_m$ request blocks of addresses whose sizes are powers of 2. The challenge of the malloc problem is to allocate blocks of addresses to child domains under heavy demand, as the address space becomes fragmented over time. A good allocation algorithm should satisfy as many requests as possible, while attempting to minimize the number of blocks a child domain holds (to keep routing tables small) and the number of times a child must change addresses (to reduce routing table flux).

A child domain that requests additional addresses may be satisfied in three different ways:

- **expansion**: A child is given a new block in addition to its current blocks. Each new block increases the size of the domain's routing table.
- **doubling**: One of the child's blocks is combined with a free *buddy block*, which has the same address expression except for one different instantiated bit. By combining with a buddy, the new block can still be represented with a single address expression, namely one where the differing bit is changed to a *don't care* bit. Growth by doubling is desirable because it keeps routing table sizes stable and reduces the scope of routing table updates.
- **migration**: A child exchanges one or more of its blocks for a new block that is as large as all of the old blocks combined. Following a migration, the child then tries to grow by expansion; migration followed by expansion is used to keep the total number of blocks assigned to a child within some bound. This helps reduce the size of the domain's routing table at the expense of some routing table flux.

In the MASC architecture, the allocation algorithm uses prefix-based expressions and allocates new blocks using a worst-fit placement mechanism called ARBE. Worst-fit placement generally leaves free space adjacent to each newly allocated block, which can be used in the future for doubling. When a child needs more addresses, it first checks whether it has free addresses available in one of its current blocks. Otherwise it tries to expand to an additional block or double one of its existing blocks. If this fails then it tries to migrate all its holdings to a new block. An adaptive mechanism triggers request or release of blocks based on low and high utilization thresholds. Later in this paper, we examine ARBE more closely, in the context of our taxonomy of allocation algorithms, and discuss its advantages and disadvantages.

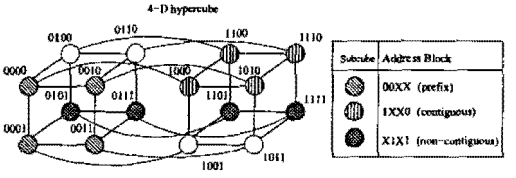



Fig. 1. The correspondence between address allocation and subcube allocation

### C. Subcube Allocation and the Malloc Problem

The hypercube is an elegant recursive mathematical structure that served as the underlying communication network of the Intel iPSC and N-Cube parallel processors back in the late 1980s and early 1990s. In a hypercube, the $2^n$ processors are each labeled with an $n$-bit address; processors whose labels differ in exactly one bit position are connected.

A *subcube* is a subset of the nodes and edges of a hypercube that themselves form a smaller hypercube. In a hypercube machine, parallel applications request subcubes, hold them for the runtime of the application, and then release the subcubes back to the operating system scheduler. The algorithm used by the scheduler to handle the requests and releases of the subcubes is the *subcube allocation algorithm* and has been the target of intensive research for many years [5], [6], [7], [8]

A key observation is the fact that a subcube is equivalent to a block of addresses described by a single address expression. Thus, as shown in Figure 1, a given subcube — or its equivalent block of addresses – can be described using prefix-based, contiguous, or non-contiguous address expressions. This equivalence means that subcube recognition techniques can be applied to the problem of multicast address allocation. However,a number of key differences and practical constraints associated with the malloc problem require that results from hypercube theory be applied to the address allocation problem with great care.

## III. Complexity of Address Allocation

Any practical allocation scheme must be able to double and migrate efficiently. In this paper, we seek algorithms that yield optimal solutions in polynomial time and space. Where this is not possible we sacrifice optimality in order to maintain a polynomial solution.

Below, we summarize complexity results for the three classes of address allocation schemes: prefix-based, contiguous, and non-contiguous.

### A. Doubling Complexity

In any prefix-based allocation scheme, there is only one choice for doubling, i.e., doubling can occur only by converting the rightmost instantiated bit to a *don't care* bit. For example, if child domain $C1$ holds address block 000XX, it can only double into the block 00XXX.

Recognition for $n$ bit address space, $k$ bit subcube/block

| Subcube Allocation Scheme | Total blocks recognized: General formula | Example: $n = 8, k = 3$ |
|---|---|---|
| Buddy (prefix) | $2^{n-k}$ | 32 |
| Gray (non-contig) | $2^{n-k+1}$ | 64 |
| Dbl Gray (non-contig) | not given | 128 |
| Partners (non-contig) | $(n-k+1) \times 2^{n-k}$ | 192 |
| Cyclic (contiguous) | $n \times 2^{n-k}$ | 256 |
| Full (non-contig) | $\binom{n}{k} \times 2^{n-k}$ | 1792 |

TABLE I

RECOGNITION CAPABILITY OF ALLOCATION SCHEMES

In any contiguous allocation scheme, there are two choices for doubling, i.e. by converting either the leftmost or rightmost instantiated bit to a *don't care* bit. For example, if $C1$ holds block 0XX00, it can double into either block XXX00 or block 0XXX0.

The complexity of doubling for prefix and contiguous allocation is $O(C)$, where $C$ is the number of child domains. The algorithm simply generates the address expression for the candidate buddy block and then tests whether that block is available by checking for intersection with the other children's blocks via bitwise comparison of address expressions.

In any non-contiguous allocation scheme there are $n - k$ choices for doubling, where $n$ is the total number of bits in the full address space and $k$ is the number of *don't care* bits in the current address expression. Doubling occurs by converting any one of the instantiated bits to a *don't care*.

The complexity of doubling for non-contiguous allocation is $O(C * n)$ since it may have to examine all $n - k$ choices for doubling, testing each for intersection with the other children's blocks.

### B. Migration Complexity

The ability of an allocation scheme to migrate to a new block in a highly fragmented address space is a function of its ability to recognize blocks of the desired size in the free address space.

Table I shows the *recognition capacity* for a spectrum of subcube allocation schemes all of which can be invoked for the malloc problem. The table gives the general formula for the total number of subcubes/blocks of size $2^k$ that can be recognized in a hypercube/address space of size $2^n$. It is clear that relaxing constraints on the format of the address expression from prefix-based to non-contiguous vastly improves the potential recognition capacity. This potential may not necessarily lead to better migration performance, due to fragmentation. Nevertheless, the increased recognition capability provides strong motivation to explore contiguous and non-contiguous algorithms.

Prefix-based allocation was proved to be polynomial time in [6]. Under prefix schemes, blocks are allocated and deallocated in a rigid pattern using a free list organized by block size.

We have developed the first known polynomial time algorithm for contiguous allocation. Earlier work with hypercubes under this model focused on parallel algorithms which use an exponential number of processors [9]. Our algorithm, which we call Cyclic, exploits the fact that there are only $n$ classes of cyclic blocks, categorized by the position of the rightmost *don't care* bit. It uses techniques for logic design that are are exponential time for logic circuits [10], but polynomial time for cyclic address allocation. In the next section we give an overview of Cyclic; the algorithm is fairly complex and described more thoroughly in [11].

### B.1 Non-Contiguous Allocation

Non-contiguous allocation is not as straightforward because subtly different statements of the problem have been proposed with different complexity results. We first give the complexity results, then discuss their implications for address allocation protocols. In the following, a *feasible* set of requests is one in which the sum of the all the requested blocks does not exceed the full address space.

**Problem 1Sin gle-Request Address Allocation.** *Given child domains $C_1$ through $C_m$ which have already been successfully allocated (disjoint) blocks $B_1$ through $B_m$, respectively, does there exist a free block of size $2^k$, $k <= n$?*

**Theorem 1Sin gle-Request Address Allocation is NP-hard.** We prove this by reduction from SAT. We establish a direct correspondence between clauses and subcubes, showing that a set of clauses is satisfied *iff* there is a free subcube of dimension $k$ after the subcubes corresponding to those clauses are allocated to the child domains. The full proof can be found in [11].

**Problem 2Un ordered-Requests Address Allocation.** *Given a feasible unordered set of requests for blocks of sizes $s_1$ through $s_m$, is there an allocation that satisfies this set of requests regardless of the order in which they are issued?*

**Theorem 2Un ordered-Requests Address Allocation is NP-hard.** This is an instance of a more general problem involving offline subcube allocation that was proved NP-hard by Dutt and Hayes [5].

**Problem 3Or dered-Requests Address Allocation.** *Given an ordered sequence of requests for blocks of sizes $s_1$ through $s_m$ is there an allocation that assigns a block to each request if a free block exists at the time of the request?*

Ordered-Requests Address Allocation is an open problem. We conjecture that it is solvable in polynomial time and outline an algorithm for this problem. The reason this problem may admit a polynomial time solution, while the others do not, lies in the fact that with ordered requests we know which of the past requests have been satisfied and which blocks have been allocated to each child. In other words, past history and current state are known at the time of each given request. Problem 2 requires that an algorithm be able to satisfy all $n!$ possible request sequences, while Problem 1 requires that the algorithm be able to reconstruct the sequence of requests that led to the current situation.

### B.2 Implications for the MASC Architecture

Theorem 1 implies that it is not sufficient to determine the current allocation state and then satisfy a given request. For example, a child domain that needs a new block of addresses may want

to query its siblings to find out what blocks they hold, or a parent domain may simply track its allocations. In both cases, it is not possible for the child or the parent to find a free block of the desired size in polynomial time.

Theorem 2 states that there is no polynomial time algorithm that can satisfy a feasible set of unordered requests. However, we note that in a realistic setting requests for blocks may occur in a fixed, ordered sequence; hence it is not necessary to optimize over all possible orderings.

Problem 3 is a more natural statement of the malloc problem, and we believe this can lead to a polynomial time algorithm for non-contiguous address allocation. The algorithm we present in the following section is framed in terms of a request-reply protocol, but the same results should apply to any protocol that maintains ordering for requests.

## IV. Multicast Address Allocation Algorithms

We present several address allocation algorithms within the context of our theoretical framework, including those from the networking community and those we have adapted or developed that are hypercube-based. Because doubling is a straightforward operation for all algorithms, we focus on migration. Recall that with MASC a child domain tries to migrate when it is unable to double one of its current blocks.

Migration algorithms can be characterized by their recognition capacity (prefix, guous, non-contiguous) and by their *fit type* (first fit, last fit, best fit, and worst fit). Due to space limitations, we give only a high level description of our algorithms; details can be found in [11].

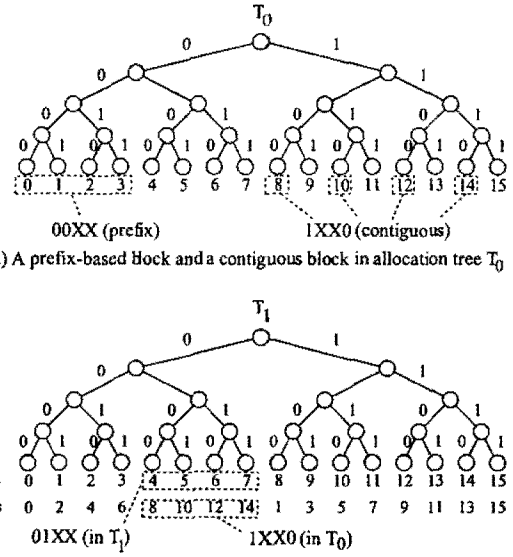
Our discussion uses a simple example throughout: a single-level domain hierarchy, an address space of $2^4$ addresses, and the following sequence of requests for addresses (given as block sizes): 2, 1, 1, 2.

### A. Prefix-based algorithms: Prefix-FF and Prefix-ARBE

Prefix-based algorithms can be best understood through the use of an allocation tree in which the leaf nodes are labeled left to right with the binary addresses 0 through $2^n - 1$. Left edges are labeled with 0 and right edges labeled with 1. See Figure 2(a).

It is easy to see that the binary sequence on the path from the root to any leaf node is precisely the label of that leaf node. Any interior node in the tree corresponds to a block of addresses contained in the subtree rooted at that node. The expression for this block is the binary sequence on the path from the root to that interior node, followed by *don't cares*.

Prefix-FF allocates addresses using first-fit; it is identical to the Buddy Subcube Algorithm [6]. Prefix-ARBE allocates blocks using a worst-fit, reverse-bit ordering [12]. Figures 3(a) and (b) show how Prefix-FF and Prefix-ARBE would handle the above sequence of addresses. With Prefix-FF, the requests are all packed into the low numbered addresses. As a result, no child block can double into its buddy block, but migration requests for 2, 4, or 8 addresses can be accommodated. Under Prefix-ARBE, the four initial requests are spaced out so that all children can double. However, no migration requests of size 4 or 8 can be satisfied.




(a) A prefix-based block and a contiguous block in allocation tree $T_0$

(b) The contiguous block 1XX0 in $T_0$ above has been transformed into a prefix block 01XX in $T_1$

Fig. 2. Allocation trees for prefix and contiguous allocation

(a) First Fit: [0 1] [2 3] [4 5] 6 7 8 9 10 11 12 13 14 15

(b) ARBE: [0 1] 2 3 [4] 5 6 7 [8] 9 10 11 [12 13] 14 15

(b) Fragmented ARBE: [0 1] 2 3 [4] 5 6 7 [8] 9 [10] 11 [12 13] 14 [15]

Fig. 3. Allocation for requests 2, 1, 1, 2 under Prefix-FF and ARBE fits

### B. Contiguous algorithm: Cyclic

Assuming the initial allocations shown in Figure 3(a) and (b), contiguous allocation improves over prefix-based allocation: all of the children can double and migration requests of sizes 2,4, and 8 can be satisfied under either FF or ARBE fits.

We have developed a polynomial time algorithm for contiguous address allocation called Cyclic. The key features of the algorithm are (1) it inspects only $n$ allocation trees, (2) it simplifies the task of finding a $k$-cube into that of finding a single free node in a truncated allocation tree, and (3) it uses binary search and the consensus operation from logic design to locate a single free node and thus a free $k$-cube.

Cyclic inspects $n$ allocation trees, one corresponding to each of the possible bit positions occupied by the rightmost *don't care* bit. Figure 2(b) shows the representation of block 00XX in $T_0$ and block 1XX0 as they appear in trees $T_0$ and $T_1$.

Within a given allocation tree $T_i$, Cyclic transforms the task of searching for a free $k - cube$ into the task of finding a single free node in two steps. First, the child holdings are represented as prefix-based holdings in the current allocation tree $T_i$ via wraparound right-shift of $i$ bits. Then, the last $k$ bits are truncated from each child's holding.

Once a tree is transformed, then Cyclic does a binary search of the tree to find a free node. If the search is successful, it yields a free node in tree $T_i$ that can be translated back to the address expression for the corresponding free $k - cube$. If the search is

not successful, there is no free node in the tree and the operation must be repeated on the next allocation tree.

To determine whether there is a free node in a given subtree, Cyclic uses the *consensus operation* [10]. Consensus is a binary operation that finds the common block of addresses for two adjacent blocks. If consensus is applied to two buddies, the result is the combination of the buddies into a larger block. Cyclic starts with a list of the child holdings and repeatedly applies consensus to all pairs of adjacent blocks. Any blocks that are covered by a larger block are removed from the list. This procedure is repeated until no buddies remain. At most, a block can be combined with its buddy $n$ times since each buddy changes an instantiated bit to a *don't care*. Thus, we are guaranteed that the algorithm terminates after $n$ iterations. These repeated invocations of the consensus operation will yield the whole subtree *iff* the subtree is covered by the children. This indicates a failure to find a free node in the subtree.

The complexity of Cyclic is $O(C * n^3)$.

### C. Non-contiguous algorithm: MaxQ

The advantages of non-contiguous allocation can be seen from the highly fragmented situation in Figure 3(c). Cyclic can only migrate to new blocks of size 2, while a non-contiguous algorithm can migrate to blocks of sizes 2 and 4. For example, a free non-contiguous block is 0X1X.

We have developed a non-contiguous address allocation algorithm for the Ordered-Requests problem called MaxQ. MaxQ uses the consensus operation to maintain a free list that contains a maximal free subcube. This free list is a weaker type of free list than that proposed by [5] which is a maximal free list that is greater than all other maximal free lists. Our free list only attempts to find one of all the maximal free blocks of addresses, of which there may be several, and then the rest of the list contains a sub-optimal list of free address blocks. For example, if the free list contained the free addresses 000, 001, 110, 100, the algorithm in [5] would be guaranteed to find 00X and 1X0 as the maximal free list. While MaxQ might find this list, it could also find the free list of 001,X0 0,and 110.

Using the consensus operation, MaxQ compares the elements of the free list to each other and find the consensus between all pairs. Any new consensus which covers a pair of addresses is kept and the covered pairs are removed. We apply the consensus operation to all pairs in the free list repeatedly until there are no new consensus blocks. As with Cyclic, we are guaranteed that this will execute at most $n$ iterations. Once we have a list of maximal free blocks given from the pairs in the original list, we can choose any of the largest blocks and then keep the results of this block with the subtractions of the other blocks in a new list and maintain a polynomial sized free list.

Using a free list allows us to ensure that if a migration needs a block of size $k$, then a simple traversal through the list in search of a k-sized block will reveal if one exists. Since we know our free list will contain a maximal free block, then if there is not a $k$-sized block in the list, there is not a maximal block of that size in the address space. Proving MaxQ is polynomial time consists of proving that the free list will always remain polynomial in size.

Note that a non-contiguous model for address expressions called *kampai* was introduced in [13] for unicast routing. However, the *kampai* algorithm was restricted to growth through doubling only.

### D. Conclusion

In this paper we have established a theoretical framework for the multicast address allocation problem by showing its close resemblance to the subcube allocation problem in hypercubes. We developed a classification scheme for address expressions into prefix-based, contiguous, and non-contiguous, based on constraints on the location of *don't care* bits. We then proved complexity results for each class, showing prefix and contiguous allocation to be polynomial time, and showing two non-contiguous allocation problems (Single Request and Unordered Requests) to be NP-hard. We presented Cyclic, the first polynomial time algorithm for contiguous allocation. Finally, we conjectured that Non-Contiguous Ordered Requests is polynomial time and presented the MaxQ algorithm for this problem. Throughout, we focused on the implications of our results and on finding practical algorithms for the malloc problem.

Based on their recognition capability, contiguous and non-contiguous algorithms appear to hold great promise for multicast address allocation. In our companion paper [2], we investigate the performance of these algorithms within a general model of the malloc problem.

## REFERENCES


[1] M. Livingston, V. Lo, K. Windisch, and D. Zappala, "Cyclic Block Allocation: A New Scheme for Hierarchical Multicast Address Allocation," in *First International Workshop on Networked Group Communication*, L. Rizzo and S. Fdida, Eds., November 1999.

[2] D. Zappala, C. GauthierDickey, and V. Lo, "Modeling the Multicast Address Allocation Problem," in *IEEE Globecom 2002, Global Internet Symposium*, November 2002.

[3] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D.Estrin, and M. Handley, "The MASC/BGMP Architecture for Inter-domain Multicast Routing," in *ACM SIGCOMM*, August 1998.

[4] P. Radoslavov, D. Estrin, R. Govindan, M. Handley, S. Kumar, and D. Thaler, "The Multicast Address-Set Claim (MASC) Protocol," RFC 2909, September 2000.

[5] S. Dutt and J. P. Hayes, "Subcube Allocation in Hypercube Computers," *IEEE Transactions on Computers*, vol. 40, no. 3, March 1991.

[6] M. Chen and K. G. Shin, "Process Allocation in an N-Cube Multiprocessor Using Gray Code," *IEEE Transactions on Computers*, vol. 36, no. 12, December 1987.

[7] A. AlDhelaan and B. Bose, "A New Strategy for Processor Allocation in an n-Cube Multiprocessor," in *Proceedings of the International Phoenix Conference on Computers and Communication*, March 1989.

[8] V. M. Lo, W. Liu, B. Nitzberg, and K. Windisch, "Noncontiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers," *IEEE Transactions on Parallel and Distributed Systems*,J uly 1997.

[9] M. Livingston and Q. F. Stout, "Fault Tolerance of the Cyclic Buddy Subcube Location Scheme in Hypercubes," in *Proceedings of the 6th Distributed Memory Computing Conference (DMCC6)*, 1991.

[10] M. R. Dagenais, V. K. Agarwal, and N. C. Rumin, "McBOOLE: A New Procedure for Exact Logic Minimization," *IEEE Transactions on Computer-Aided Design*,v ol. CAD-5, no. 1, January 1986.

[11] V. Lo, D. Zappala, C. GauthierDickey, and T. Singer, "A Theoretical Framework for Multicast Address Allocation," Tech. Rep. UO-TR-2002-01, University of Oregon, 2002.

[12] P. I. Radoslavov, D. Estrin, and R. Govindan, "A Claim-Collide Mechanism for Robust Distributed Resource Allocation," Tech. Rep. USC-CS-99-711, Computer Science, University of Southern California, 1999.

[13] P. Tsuchiya, "Efficient and Flexible Hierarchical Address Allocation," in *INET92*, June 1992.