Modeling the Multicast Address Allocation Problem

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

Abstract—**To support IP multicast, domains must assign a unique multicast address to each application from a limited, globally-shared address space. We examine the performance of several classes of address allocation algorithms within the context of the MASC architecture. This study is the first of its kind to model the generalized multicast address allocation problem and consider non-contiguous allocation algorithms. We find that prefix-based allocation outperforms our non-contiguous algorithm, despite the apparent advantages of non-contiguous allocation. We also verify the benefit of using worst-fit for new allocations and outline several areas for future work.**

I. Introduction

Multicast address allocation is one of several obstacles that has slowed multicast deployment. The multicast infrastructure built using Deering's original IP multicast model – now referred to as Any Source Multicast (ASM) – requires that applications share a single, global address space. In this model, a multicast address identifies a logical group of members and any source may send data to this dynamic set of members at any time. No two applications may share the same multicast address at the same time, or else the group members may receive traffic they do not want.

The key problem for ASM multicast address allocation is to assign a unique address to each application from a limited, globally-shared address space. Because the address space is limited, addresses must be re-used both over time and across topologically-distinct multicast groups. Although the problem contains both aspects of sharing, for simplicity we refer to the problem of allocating and sharing of addresses over time as the *malloc problem* for the rest of this paper.

To address the malloc problem, Kumar et al. developed the MASC address allocation architecture, which dynamically allocates addresses along the provider-subscriber hierarchy [1]. In MASC, a domain claims a range of addresses from its parent, then allocates these addresses to hosts within its domain as well as to its child domains. In both Kumar's original research and in subsequent IETF specifications [2], MASC uses a contiguous, prefix-based allocation scheme with a worst-fit placement algorithm. Neither of these choices has been systematically studied to determine if they are the best for the MASC architecture, despite an intuition by Kumar et al. that non-contiguous schemes could provide better performance [1].

We believe the general malloc problem – and the MASC algorithms in particular – remain important to the multicast research and engineering communities, despite the recent surge of interest in alternative multicast routing architectures. The most viable of these architectures, Source-Specific Multicast (SSM) [3], [4], elegantly solves the multicast address allocation problem, but at the cost of restricting a multicast tree to a single source. Additional application-level services can be built to provide efficient multiple-source SSM delivery [5], but native support for ASM will always offer the best possible performance in terms of latency and network utilization. Likewise, pure application-layer approaches [6], [7], [8], [9], [10] offer simpler deployment, but have even worse performance than SSM for multiple-source sessions because no network layer support is utilized. Thus it is quite likely that ASM services will continue to be offered in the foreseeable future, operating alongside these other approaches.

Our goal in this paper is to systematically study the performance of a variety of allocation algorithms and fit methods to determine which are best suited for the MASC architecture. Previous work by Radoslavov et al. examined the performance of MASC's prefix-based allocation algorithm in isolation, demonstrating its capabilities in a variety of situations [11]. Our study broadens this work substantially by examining several classes of algorithms derived from our theoretical framework for the multicast address allocation problem [12]. The basis these algorithms is to use a more flexible representation for address blocks and hence provide a greater ability to recognize free blocks in a fragmented space. Our study also introduces a simple model for the generalized malloc problem, load functions that represent shifting demand between child domains, and a set of metrics that accurately capture an algorithm's ability to grow its current address holdings or migrate to new holdings. Finally, our study examines a variety of fit types to determine which are best suited to this dynamic system.

The initial results of our study indicate that prefix-based allocation actually performs quite well in comparison with non-contiguous algorithms. These results are surprising because it has been assumed – by the authors of MASC and by ourselves – that non-contiguous address schemes would outperform prefix-based allocation mechanisms. We also find that worst-fit for newly allocated blocks performs better than first-fit and best-fit. This result is remarkably consistent across algorithm classes, which validates MASC's general design independent of the allocation algorithm used.

In the following sections we provide background for the malloc problem, describe the algorithms we study, and present our model for the malloc problem. Then we present the results of our address allocation experiments and describe a range of additional issues to explore.

II. Background

Handley and Jacobson developed the first address allocation mechanism as a part of their *session directory* tool [13]. This tool allows users to register multicast sessions and declare a scope for the session in terms of an administrative boundary or

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

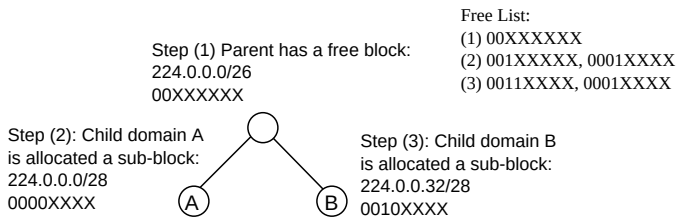

Fig. 1. MASC Allocation Example

a number of hops. Using a session scope allows addresses to be reused over space because two non-overlapping sessions can each use the same address. Reuse over time is enforced by requiring an application to return its addresses when it is done. Handley extensively studied the performance of the session directory address allocation mechanism and concluded that a hierarchical allocation architecture was needed to allocate addresses from a sufficiently large shared space [14].

Because of this work, a group of researchers at USC/ISI and Michigan developed the MASC architecture, which uses the provider-subscriber hierarchy already present in the Internet to dynamically allocate blocks of multicast addresses to domains [1]. A domain running MASC uses a claim-collide protocol to request blocks of addresses from its parent domain and resolve conflicts with any sibling domains trying to claim the same block. A separate set of protocols is used to allocate addresses from these blocks to hosts within the domain.

The MASC address allocation mechanism uses prefix-based expressions for address blocks and a worst-fit algorithm for new requests. Figure 1 illustrates these basic concepts using a simple two-level hierarchy. The parent domain has been allocated a range of 64 addresses given in dotted-decimal notation as 224.0.0.0/26. Ignoring the first 24 bits, we can represent this as 00XXXXXX, where the X's represent *don't care* bits that can be set to either 0 or 1. Because MASC uses prefix-based expressions, this means that all of the *don't care* bits must be in the rightmost positions. Similarly, child domain A has been allocated 16 addresses, represented as 224.0.0.0/28 or 0000XXXX. Given this situation, we can represent the free addresses in two blocks: 001XXXXX and 0001XXXX. When domain B requests 16 addresses, this request is filled using worst-fit; first, the largest free block (001XXXXX) is chosen, and then the first sub-block of the requested size is selected (0010XXXX). When there are multiple free blocks of the same size, one is chosen at random.

The evaluation of MASC performed by Radoslavov et al. [11] focuses on evaluating claim-collide as a viable architecture, rather than on evaluating the address allocation mechanism itself. They illustrate conditions under which address allocation latency is low, despite the possibility of colliding requests and network partitions. In addition, they show that overall resource utilization is a function of the number of levels of hierarchy and that the MASC architecture can adapt to changes in overall load. Their evaluation does not consider alternatives to prefix-based schemes nor alternatives to worst-fit as a fit type.

III. Algorithms

Our study of address allocation algorithms for the malloc problem is based on a theoretical foundation described in a companion paper [12]. In this paper we show how the malloc problem is closely related to the problem of subcube allocation in hypercube architectures and classify allocation algorithms into three types based on how they recognize blocks of addresses:

- Prefix-Based: Address blocks are represented by an expression where the *don't care* bits are in the rightmost positions.
- Contiguous: Address blocks are represented by an expression where the *don't care* bits are contiguous, with wraparound allowed.
- Non-Contiguous: Address blocks are represented by an expression where the *don't care* bits are in arbitrary positions.

For example, given a block of 2^4 addresses allocated from a space of 2^8 addresses, 0010XXXX denotes a prefix-based address expression, 01XXXX10 and XX0100XX represent contiguous allocations, and 0X0XX1X0 represents a non-contiguous allocation. Note that each class is contained in the next, with non-contiguous being the most general class.

A. Non-Contiguous Allocation

In this paper, we focus on the performance of non-contiguous algorithms to determine whether their ability to recognize more possible blocks translates into increased performance. When a domain needs additional addresses, an allocation mechanism has two basic choices. First, it can expand the domain's current block of addresses by *doubling* it. In effect, this means changing one of the block's instantiated bits to a *don't care* bit. If doubling is not possible (because some other domain holds a conflicting block), then the allocation mechanism can allocate a new block. If there is a limit on the number of blocks a domain can hold, then this may involve *migrating* a current block to a larger block that can satisfy current demand.

Intuitively, non-contiguous algorithms should have a performance advantage over prefix-based schemes with regards to both doubling and migration. For doubling, a non-contiguous algorithm can choose *any* of the instantiated bits in a domain's current block and convert it to a *don't care* bit as long as the space is available. A prefix-based algorithm, on the other hand, can only convert the right-most instantiated bit since it must always use a prefix expression for the domain's block. For migration, a non-contiguous algorithm can recognize all possible free blocks, whereas a prefix scheme can only see those blocks that can be represented by the prefix notation. As shown in [12], non-contiguous algorithms can potentially recognize $\binom{n}{k}$ more blocks of size k for an n-bit address space. Thus a non-contiguous algorithm should have an easier time finding a free block for migration.

B. General Allocation Algorithm

To study address allocation performance, we have developed a general algorithm that uses a free list to keep track of unallocated blocks of addresses. This general algorithm handles three basic cases:

• Allocation: Allocation is used when a domain is requesting an additional block or is migrating to a new block. In both cases, the domain requests a block of size k, and the free list is searched for a large enough free block according to a given fit type. With worst-fit, the largest free block is selected and a sub-block of size k is allocated from this block. For best-fit, the smallest free block that is at least size k is chosen. First-fit chooses the first free block that is at least size k.
• Doubling: Doubling is used when a domain has used all of the addresses in its current blocks and needs more. The algorithm will try to double by looking for a *buddy* of the current block in the free list. A buddy is a block with the same *don't care* bits and only one different instantiated bit. For prefix-based schemes, there can only be one buddy for a block, but for non-contiguous schemes there are as many buddies as there are instantiated bits.
• Release: Release occurs when a domain wants to return some addresses or when it migrates to a new block. When a block is released, the allocation algorithm will look for its buddy in the free list and, if it is found, combine them. Note that for non-contiguous allocations, a block can have as many buddies as instantiated bits. Once a block is combined with its buddy, this newer block may in turn have a buddy in the free list. This combination procedure is thus repeated until no more free buddies are found.

For any prefix-based scheme, this method for combining blocks is optimal since there is only one possible buddy for any block. For non-contiguous schemes, however, our algorithm may produce a sub-optimal free list. For our general algorithm, if m is the number of free blocks in the list and n is the number of bits in the address space, then the algorithm completes in $O(mn)$ time for each possible buddy. Thus with prefix-based allocations, the algorithm is $O(mn)$ while for non-contiguous allocations the algorithm is $O(mn^2)$.

C. The MaxQ Algorithm

We have developed a non-contiguous address allocation algorithm called MaxQ. This algorithm extends our general algorithm in two ways in order to make non-contiguous address allocation more efficient. First, MaxQ keeps track of the largest free block, even when non-contiguous addresses have been allocated. It does this by using the *consensus* operation from logic design [15] to consolidate blocks and maintain a polynomial-size free list. Second, MaxQ is able to find buddies for doubling that cross multiple blocks in the free list. More details on the MaxQ algorithm can be found in a companion paper [12].

IV. Modeling Address Allocation

We have developed a simplified model of the general malloc problem that enables us to isolate address allocation performance from the other aspects of MASC. We have also developed a model for load that stresses the allocation algorithm by shifting demand among child domains over time.

A. Allocation Model

Our address allocation model is based on the essential parts of the MASC architecture. A given domain may hold up to p blocks of addresses. When the domain needs more addresses, it may double any of its blocks as long as the utilization of the blocks is at least t, where t is the target occupancy. If the target occupancy cannot be met, or if none of the current blocks can be doubled, then the domain may add a new block. Once the domain reaches p blocks and it cannot double, then it tries to migrate one of its blocks to a new space where doubling can occur.

In order to isolate the performance of the allocation mechanisms, we do not model MASC's claim-collide mechanism nor address lifetimes. Instead, we use a simple request-reply mechanism, where a child domain requests addresses in blocks of size k from its parent domain. The parent domain runs a centralized address allocation mechanism, taking into account holdings for all its current children. Thus all doubling and migration is performed by the parent domain. In addition, migration occurs "instantaneously"; when migration occurs the child domain immediately releases its current block and accepts the new one it has been assigned.

Our initial set of experiments use a single-level hierarchy, with one parent domain and a set of children. We set k, the block size, to 256. In addition, we use $p = 2$ and $t = 75\%$, as specified in the original work on MASC [1]. We are currently experimenting with multiple levels of hierarchy and different values for p and t.

B. Load Model

Modeling load for an address allocation algorithm is difficult since we cannot predict the demand for multicast should it be deployed on a wide scale. However, we can make two simple observations. First, it is unlikely that domains will have a fixed demand over time, or else static allocation of addresses would be acceptable. Second, it is unreasonable to have demand for multicast that is larger than the total number of available addresses. Such a situation would make the ASM architecture untenable since large numbers of people would be turned away.

Based on this reasoning, we have designed a model where demand for addresses shifts between child domains. This differs from the model used by Radoslavov et al., where each of the child domains has a homogeneous demand and only the overall load fluctuates significantly. For our model, the demand for each child domain is given as a function that varies between a minimum and maximum value using a staircase function. Specifically, the demand starts at the minimum value, ascends to the maximum value using a staircase function, stays at the maximum value for some time, then descends back to the minimum value and stays there for some time. This function repeats periodically for the length of the simulation. To model shifting demand between domains, we shift each domain's load function in time by a constant value.

In our experiments, domains use a minimum of 0 blocks, a maximum of 256 blocks, and request 256 addresses at a time (this is the size of a block and the step of the staircase function).

V. EXPERIMENTS

Our primary measure of the effectiveness of an address allocation algorithm is the outcome of a request for additional addresses. For a given request, there are four possible *outcomes*:
1. The request is filled by a new block or one of the domain's current blocks. The latter happens when one of the blocks has unused addresses. For example, a domain with 512 addresses that asks for 256 more will double its block to hold 1024 addresses. Initially this domain will only use 768 addresses and has room for an additional 256 to satisfy the next request.
2. The request is filled by doubling one of the domain's current blocks. This happens when all of the addresses in the current blocks are used and a buddy of one of the blocks is free.
3. The request is filled by migrating the domain to a new block. This happens when doubling cannot occur, either if no buddies are free or doubling would reduce utilization below the target occupancy. If a large enough free block can be found, the domain will migrate to it.
4. The request fails. This occurs if none of the above actions can be taken.

In some cases, we want to examine only those requests that must be satisfied either by doubling or migration. We call these *growth requests* since they cannot be filled by a domain's current blocks and hence its holdings need to grow. For a growth request, doubling is preferred to migration because migration causes routing table entries to change; frequent routing table change may lead to instability for the multicast routing protocol.

Our experiments evaluate all three classes of algorithms – prefix, contiguous, and non-contiguous – but we report here only on prefix and non-contiguous. Generally, the same results apply to contiguous algorithms as for non-contiguous algorithms, though performance in some cases is a little better.

Our primary set of experiments uses a single parent with 25 child domains, since it is at this point that some requests start to fail for even the best algorithms. With 25 children, the load is 78%, meaning the sum of all requests for all children (whether granted or not) account for 78% of all available addresses. We have also run experiments at a number of different loads by varying the number of child domains. A subsequent section shows how address allocation algorithms react to increasing load and discusses utilization.

A. Allocation Outcome for 25 Children

Surprisingly, our initial results indicate that prefix-based allocation outperforms the MaxQ non-contiguous algorithm. Figure 2 shows the outcome over time for worst-fit prefix-based allocation, and Figure 3 shows the outcome for worst-fit MaxQ. Under this load, about 10% of the growth requests fail, close to 35% are filled by doubling, and about 55% by migrating. With this same load, worst-fit MaxQ has a failure rate of about 55%, with doubling at 15% and migration at 25%.

Our experiments also indicate that the fit type for a given algorithm affects how requests are filled. Using best-fit for prefix allocation, the failure rate reduces to about 5%, but doubling is less than 25% and migration is above 70%. Thus, compared

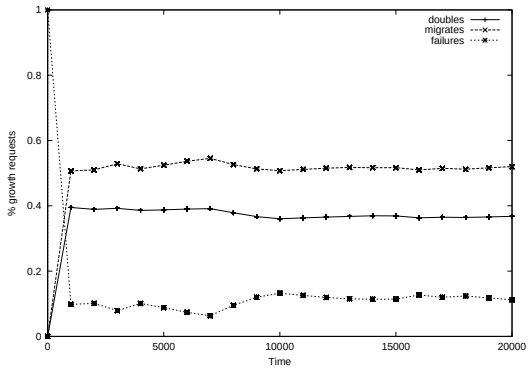

Fig. 2. Worst-Fit Prefix Allocation: Outcome of Growth Requests

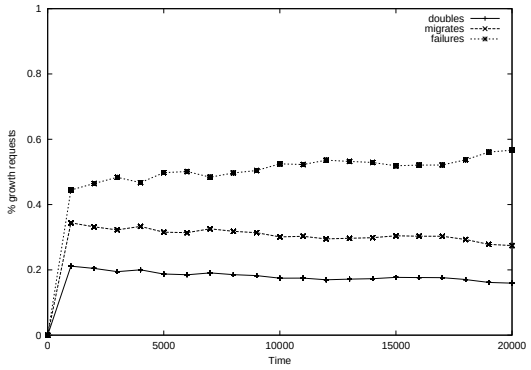

Fig. 3. Worst-Fit MaxQ Allocation: Outcome of Growth Requests

to worst-fit, best-fit is able to satisfy slightly more growth requests at the expense of much more frequent migration. First-fit performs about the same as best-fit.

The algorithms using worst-fit are likewise sensitive to how the largest block is chosen. When worst-fit finds several largest blocks of the same size, it must decide which block to select for a given allocation request. For the worst-fit prefix results shown above, we use the first block given by numerical ordering. When we instead choose the largest block randomly, as suggested by MASC, the failure rate rises to almost 25%. Performance is even worse for selection using reverse-bit ordering. Table I shows the complete results for each of these orderings.

To understand the significance of these findings with respect to overall performance, we examine the outcome of all requests instead of just the growth requests. Table II summarizes these results. For both prefix and MaxQ allocation, most requests are filled "internally", that is by unused addresses within a domain's current block. This is because one successful doubling will result in several later requests being filled internally. While in both cases growth requests account for only a small number of all requests, they do have a significant impact on performance. Prefix fails only about 1% of the time while for MaxQ

Worst-Fit Ordering	Failure Rate	Doubling Rate	Migration Rate
Numerical	10%	35%	55%
Reverse-Bit	25%	25%	50%
Random	25%	30%	45%

Table I. Worst-Fit Ordering for Prefix: Outcome of Growth Requests

Algorithm	Failure Rate	Internal Rate	Doubling Rate	Migration Rate
Prefix: Best-Fit	1%	93%	1%	5%
Prefix: First-Fit	1%	93%	1%	5%
Prefix: Worst-Fit, Numerical	1%	92%	4%	3%
Prefix: Worst-Fit, Reverse-Bit	3%	90%	4%	3%
Prefix: Worst-Fit, Random	3%	90%	4%	3%
MaxQ: Best-Fit	6%	87%	5%	2%
MaxQ: First-Fit	6%	87%	5%	2%
MaxQ: Worst-Fit, Numerical	10%	83%	4%	3%
MaxQ: Worst-Fit, Random	10%	83%	4%	3%

Table II. Allocation Algorithms : Outcome of All Requests

failure over all requests is close to 10%. Note that we omit the cases where a domain obtains a new block because this happens rarely.

Overall, these results indicate that prefix allocation performs better than non-contiguous allocation, that fit-type influences how growth requests are filled, and that the differences between worst-fit orderings are relatively minor.

B. Load and Utilization

Our results indicate that address allocation algorithms are very sensitive to load. We measure load as the percentage of the total address space that is requested by child domains, and we increase load by adding additional children. With 20 children the load is 60%, with 25 children the load is 78%, and with 30 children the load is 95%.

Figure 4 illustrates that the failure rate increases sharply for Prefix Worst-Fit once the load goes above 80%. The failure rate for MaxQ increases earlier than this, likely closer to 70% if we extrapolate a similar curve. Each point on this graph represents the average failure rate after the system has stabilized (the period 10,000 to 20,000 on previous graphs).

This result indicates that for MASC we can expect at most 80% of the addresses to be allocated at each level of the hierarchy. At this load, the child domains average between 90 to 95% utilization, meaning they use over 90% of the addresses they request. These numbers indicate that with a 28-bit address space (allowing some bits to be used for SSM or other architectures) and a 3-level hierarchy we can expect to allocate close to 40% of the addresses, supporting over 100 million sessions. This is significantly better than the $O(sqrt(n))$ allocation limit of the session directory tool. This limit was derived by Handley in 1997 and was the primary motivation for developing MASC. Our numbers are comparable to those obtained by Radoslavov et al. [11], indicating our simplified model captures the essential aspects of the MASC architecture.

VI. Future Work

Our initial results from this study of the malloc problem indicate that non-contiguous allocation does not perform as well as prefix-based allocation. Non-contiguous schemes should be able to recognize more free blocks and thus allow for greater utilization of the address space. Our intuition is that each time a non-contiguous scheme allocates an address in one dimension it fragments the space in multiple overlapping dimensions.

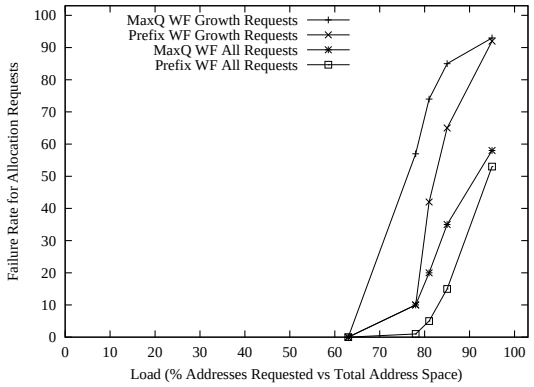

Fig. 4. Failure versus Load for Prefix and MaxQ

Studying multi-dimension problems is difficult; we are focusing on large-scale visual representations of the address space.

We are also in the process of exploring other variables in our general model of the malloc problem. These include varying the target occupancy, varying the number of prefixes a domain can hold at one time, and using different types of migration. Finally, we are exploring additional load functions beyond our staircase function.

References

[1] 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.

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

[3] D.R. Cheriton and H.W. Holbrook, "EXPRESS Multicast: Making Multicast Economically Viable," in *ACM SIGCOMM*, August 1999.

[4] H. Holbrook and B. Cain, "Source-Specific Multicast for IP," Internet Draft: work in progress, November 2000.

[5] Daniel Zappala and Aaron Fabbri, "Using SSM Proxies to Provide Efficient Multiple-Source Multicast Delivery," in *IEEE Globecom, Global Internet Symposium*, November 2001.

[6] Yang hua Chu, Sanjay G. Rao, and Hui Zhang, "A Case For End System Multicast," in *ACM Sigmetrics*, June 2000.

[7] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "Almi: An application level multicast infrastructure," in *3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.

[8] J. Liebeherr and M. Nahas, "Application-layer multicast with delaunay triangulations," in *IEEE Globecom, Global Internet Symposium*, November 2001.

[9] Yatin Chawathe, Steven McCanne, and Eric A. Brewer, "RMX: Reliable Multicast for Heterogeneous Networks," in *IEEE INFOCOM*, 2000.

[10] Paul Francis, "Yoid: Extending the Internet Multicast Architecture," Unrefereed report, 2000, Available at http://www.aciri.org/yoid.

[11] Ramesh Govindan Pavlin Ivanov Radoslavov, Deborah Estrin, "A Claim-Collide Mechanism for Robust Distributed Resource Allocation," Tech. Rep. USC-CS-99-711, Computer Science Department, University of Southern California, 1999.

[12] Virginia Lo, Daniel Zappala, and Chris GauthierDickey, "A theoretical framework for multicast address allocation," *submitted to IEEE Globecom 2002, Global Internet Symposium*, 2002.

[13] Mark Handley and Van Jacobson, "sdr," Now maintained at http://www-mice.cs.ucl.ac.uk/multimedia/software/sdr.

[14] M. Handley, "Session Directories and Scalable Internet Multicast Address Allocation," in *ACM SIGCOMM*, August 1998.

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