DAOMAP: A Distributed IPv6 Multicast Address Allocating Protocol

Marco Slaviero Martin S Olivier
ICSA Research Group
Department of Computer Science
University Of Pretoria
{mslaviero,molivier}@cs.up.ac.za

***Abstract*— Multicast address assignment is a limiting factor in multicast deployment, and previous research has shown that dynamic prefix-based addressing outperforms other models. In this paper we expound on the requirements for a scalable IPv6 multicast address allocation protocol, compute thresholds for security policies, and detail our protocol DAOMAP, which fulfills our listed requirements.**

I. Introduction

Multicasting is the point-to-multipoint transmission of packets, as opposed to unicast which uses a point-to-point mechanism. In order for disparate applications to multicast data worldwide, each application requires a globally unique multicast address (or multicast group) for listeners to receive data. The allocation of these addresses is the multicast address allocation (or *malloc*) problem [1]. In a study commissioned by 6net [1], the authors state that the primary concern of the malloc problem is **collision avoidance** [2]; i.e. ensuring two addresses selected for disparate applications are different. We might add that where collisions do occur, some mechanism is needed to detect such collisions. A further issue is the efficient allocation of the address space, in such a way that addresses are reusable over time.

The multicast address allocation problem has been studied in numerous papers, and has been the subject of several technical documents and standards. The importance of the problem is cast into sharp relief with the statement that ubiquitous global multicast is not possible without scalable address allocation. Why global multicast? In the age of broadband connectivity, the Internet becomes a viable and attractive means for content delivery, to give just one example.

Diot and his colleagues [3] list a number of possible reasons for the slow deployment of IP multicast, of which address allocation is demonstrated to be an important issue. Current solutions for address allocation are not optimal as they place unnecessary burdens on administrators. These include deployment of allocation servers or static assignment through manual registration. Although multicast has not yet lived up to its promise on a global scale, ongoing work in the multicast arena continually refines and enhances its capabilities.

Most of the work concerning the malloc problem has dealt specifically with IPv4. The MBONE, which provided multicast services over the largely unicast Internet, was an IPv4 network and deficiencies in the IPv4 address structure had to be worked around. An example of this is the lack of scope information in a standard IPv4 multicast address, leading to complete reliance on the time-to-live (TTL) field in the IPv4 header. (It is possible to limit IPv6 multicast to an administrative 'zone', based on the fields in the address.) The limitations present in IPv4 have prompted developments that, while ingenious, suffer from fundamental problems.

We believe a fresh start is needed. IPv6, as the successor to IPv4, introduces radical changes to the address structure. The address space is larger by 28 orders of magnitude and addresses have meaning and structure. In this paper we explore ways to exploit the features of the IPv6 address to provide a new solution to the malloc problem.

The rest of the paper is structured as follows: Section II provides the requisite background knowledge on current and previous practices. We solidify the requirements for an address allocation scheme in Section IV and present our protocol in Section V. Related work is described in Section VII and we conclude in Section VIII.

II. Background

A. Configuration Methods

Multicasting, like any other data transmission method, requires at least two resources to exchange data between hosts: addresses and links. The links provide a means for data to travel and addresses enable the data to be received by the correct party. Generally links are a hardware feature and are static in nature, and so we do not consider them further. Addresses on the other hand can be volatile; they are often altered, both manually and automatically. *Dynamic address* is another term for an automatically assigned address.

Under IPv6, a number of options are available to the entity assigned the task of configuring unicast addresses automatically. These can be broadly split in *stateful* and *stateless* categories. Stateful configuration makes use of a server which stores configuration information, whereas stateless configuration is different in that no configuration server is required.

The stateful configuration design has partially resolved the malloc problem; however it suffers from the administrative burden that is general to stateful configuration. The best current practice for Internet-wide multicast address allocation

[1] 6net is a temporary IPv6 project amongst European researchers

is as follows [4]: inter-domain allocation is achieved running MASC [5], [6] between domains, the allocated block is further split up for intra-domain allocation to the multicast address allocation servers (MAAS), and finally applications can request addresses from a MAAS with the MADCAP [7] mechanism. The scheme is laborious because address blocks need to be registered and manually injected into MASC. Currently no protocol exists for allocating to intra-domain MAAS and MADCAP is a cumbersome protocol with eight different message types and a client/server model.

In their description of the state of the malloc problem, Zappala et al. [1] report that *"MASC (and the rest of the hierarchical allocation structure) [have] never been deployed"*. This lead us to explore a solution to the malloc problem which uses stateless configuration, where the administrative burden is much lighter.

Separate from stateful/stateless arguments is the notion of a *session directory*. Once an application has somehow been assigned a multicast address (by stateful or stateless means), it needs to be able to advertise its presence so that potential listeners can be informed and this is accomplished by an announcement in the session directory. Session announcements are especially important for dynamic addresses, where the address an application is going to use is not known amongst its listeners prior to the session announcement.

In the MBONE the session directory also served as the address allocator; if an address was in the directory it was obviously in use and so could not be used by another application. Handley [8] showed that a directory which combined the functions of session announcement and address allocation does not scale well. He concluded that they should be split, and that an address hierarchy needs to be introduced.

In the first comprehensive analysis of the malloc problem, Zappala, Lo and GauthierDickey pointed out that the malloc problem is simply another incarnation of a well known resource allocation problem, where varying block sizes of resources are allocated and deallocated to differing parties over time. They apply techniques learned in the field of hypercube-computing processor allocation and conclude that amongst three types of addressing, *prefix*-based allocation performs at least as well as any of the other two.

Handley's and Zappala's results mesh well; a prefix-based allocation approach naturally forms an address hierarchy.

B. IPv6

We referred earlier to the added semantics in an IPv6 address, and RFC 3306 [9] defines a multicast address which is based on the unicast prefix assigned to a network. This is important because it lays the foundation for hosts to generate their own globally unique multicast addresses. Under IPv4, multicast users had three choices if they wished to acquire a globally unique multicast address. (1) Manually register a well known address with IANA, (2) register as a user of the `233/8` address space, and IANA would assign them a 24-bit prefix or (3) configure a MASC system. Using IPv6 all this becomes unnecessary.

The jump which makes this possible is the embedding of the unicast prefix in the multicast address. Each organisation which is globally routable on the IPv6 Internet has been assigned a unique network prefix in a hierarchical manner. Since this prefix is unique, it is possible for a node to generate a multicast address utilising the unique prefix. The newly formed multicast address is now unique to an organisation; what remains is to verify that it is unique *within* the organisation, i.e. not currently in use by another node. Organisations are also free to further subnet their alloted address space, which on large networks would be prudent.

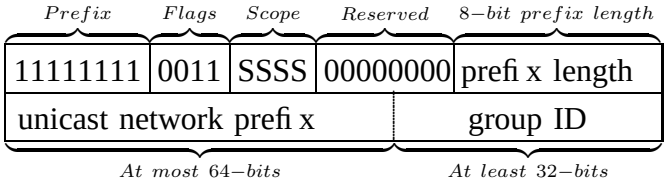

Fig. 1. Unicast-prefix based multicast address

TABLE I
PROBABILITY OF ADDRESS COLLISIONS WITH VARYING ADDRESS SPACE AND NUMBER OF GROUPS

	Group ID bit size			
$\#groups$	**32**	**40**	**48**	**64**
100	0.000001	4.50e-09	1.75e-11	2.68e-16
1000	0.000116	4.54e-07	1.77e-09	2.70e-14
10000	0.011573	0.000045	1.77e-07	2.71e-12
100000	0.687812	0.004537	0.000018	2.71e-10

Figure 1 depicts the format of a Unicast-Prefix Based (UPB) IPv6 multicast address [9]. The 4 **S** bits indicate the scope of the multicast address, which ranges from interface-local to global [10]. The **prefix length** field denotes how long the unicast network prefix field is, which is at most 64-bits. Clearly the unicast prefix is embedded in the **unicast network prefix** field, and the remainder of the address is used for **group ID** generation. The group ID will be some identifier for a particular multicast group in an organisation. It is this field which must be unique within the organisation. Combined with a unique global unicast prefix, we can form globally unique multicast addresses.

The complexity of the problem has now been reduced from detecting collisions on a global scale to generating a unique group ID within an organisation of at least 32-bits (and thus also detecting collisions), and the focus of our efforts lies in this direction.

III. GROUP ID

We begin with Table I, which depicts the probability of collisions one could expect when varying the number of allocated addresses and the address space or group ID size, assuming addresses are picked at random. We have calculated the probabilities for 32, 40, 48 and 64-bit group ID sizes. These sizes are reliant on the UPB address from Figure 1; the Unicast-prefix is at most 64-bits and the group ID is at least 32-bits. The intermediate values map to possible prefix sizes an organisation could be assigned. The number of listeners are one hundred, one thousand, ten thousand and one hundred thousand.

The table is of interest because it demonstrates how unlikely an address collision is. The only situation, according to Table I,

	Group ID bit size			
	32	**40**	**48**	**64**
$\sqrt{n}$	65536	1048576	16777216	4294967296
Addresses per node before collision				
100	655	10485	167772	42949672
1000	65	1048	16777	4294967
10000	6	104	1677	429496
100000	0	10	167	42949

TABLE II

NUMBER OF ADDRESSES PER NODE BEFORE COLLISIONS ARE PROBABLE

which has a high probability of address collisions is the intersection of a 32-bit group ID and 100000 listeners. This extreme case is highly unlikely to manifest itself, because this requires a Unicast-prefix of 64-bits. Current service providers assign 64-bit prefixes to individuals, so this situation would require a single node to request 100000 different addresses. Furthermore, the node would be competing with itself for addresses, because its prefix is unique to that node. Here optimisations are possible because the internal competition for addresses does not have to leak onto the network.

Since we assume that allocation is completely random, Handley [8] states that we can expect a collision once approximately $\sqrt{n}$ addresses are allocated, where n is the size of the address space. Table III lists values for 32, 40, 48 and 64-bit group ID sizes above which one expects an address collision, followed by the average number of addresses per group ID size that each node can reserve before an address collision becomes likely.

The significance of these values becomes clear later, however hold in mind that these figures are a lower bound for number of addresses assigned before a collision becomes likely, because we assume random allocation. In reality we make use of methods for predicting future addresses and avoiding collisions which means we can improve the odds of requesting unallocated addresses quite drastically.

IV. REQUIREMENTS

In developing a protocol for address allocation, we first need to enumerate our requirements. Below we specify eight factors which we believe are necessary in designing a malloc protocol, followed by a detailed discussion of each.

One, we are aiming for a **dynamic** allocation scheme. Two, the allocation scheme must be **distributed**. Three, our protocol must be easy to implement and, more importantly, **integrable** in current networks. Four, addresses must have a **limited lifetime**. Five, the allocation should have some level of **security**. Six, **fair-use** of the address space should be policed. Seven, the protocol must be **robust** in the face of failure. Eight, the protocol must take steps to **limit address collisions**.

While static malloc is useful for well-known services such as the Network Time Protocol, user applications in general will request, use and release addresses on an ad-hoc basis, and the nature of the applications are such that static malloc is not feasible. A dynamic protocol is clearly the better option.

We want our protocol to steer clear of central decision-making and state storage so that (a) malloc servers are not required and (b) a single point-of-failure is not present. Distributing the allocation across participating nodes creates a resilient allocation mechanism.

To satisfy the integration constraint, only those nodes wishing to participate should be forced to run the protocol. In addition, no changes will be needed on routers and other network equipment. This ties in closely with the distributed requirement, as administrators are not burdened with deployment of extra servers and software upgrades for routers.

Even though a large pool of addresses might be available, it is inevitable that eventually the space will become full if applications do not release their currently held addresses. Once the lifetime of an address expires, it is added back into the pool of unallocated addresses. Nodes are free to renew the lease on allocated addresses.

Either authenticated packets or encrypted packets, or both, would satisfy the security condition. If the objective is to prevent unauthorised nodes from acquiring addresses, authentication will limit allocation to those nodes which share a secret key. Encryption, while protecting the contents of packets, does not provide much more than authentication in the way of functionality. An attacker can simply monitor the traffic originating from a node to determine which groups the node has been allocated, although allocated but unused groups would stay hidden.

It is untenable that a single application can grab the entire address space for itself. If a rogue application can allocate all possible addresses, then other requesters will suffer from denial of service. This is difficult to satisfy in a distributed environment, due to the lack of a distinct arbitrator.

Robustness should be a cornerstone for any network protocol, we include it for completeness. In the case of error conditions, such as node failure or address collision, the protocol should still be able to allocate legitimate addresses.

We saw how generation of group IDs cannot simply take place in a random fashion. Address collision avoidance is a priority because, if done properly, addresses can be allocated without more than one claim on the network. A list of currently allocated addresses as well as analysis of past collisions will be stored by nodes to improve collision detection.

V. DAOMAP DESCRIPTION

We call our protocol DAOMAP, Distributed Allocation Of Multicast Addresses Protocol. It satisfies the requirements from Section IV, and we present the details here. DAOMAP is implemented by a daemon process called the Distributed Address Allocator (DAA), which provides an API for applications to access the malloc functions. It is envisaged that the API provides three calls, one to acquire an address, another to renew an address and a third to release an address.

We use the *claim/collide* mechanism for requesting addresses, under this scheme a requester or claimant announces its intention to use a resource. Other participants are informed, and if the resource is already in use then the current user of the resource replies with a collide message. The requester is then forced to choose another address and repeat the process. If no collide is received then the requester is able to use the resource.

While in other allocation algorithms the requester is able to ask for blocks of addresses, we ignore this scenario. Block assignment is useful when distributing addresses amongst nodes in a hierarchy, but since DAOMAP operates on the network edge this functionality is superfluous.

DAOMAP uses two message types, aptly named Claim and Collide, and both messages are carried in a UDP packet [2]. Each message carries a message ID, which monotonically increases with wraparound. DAAs save the value of the last seen message ID from other DAAs. If a message arrives with an ID less than or equal to the message ID stored for the sending DAA, then the message can be discarded as it has already been seen. After wraparound DAAs must be careful about discarding messages for a short period.

A Claim message consists of the 2-tuple (address, lifetime) [3], where address is a valid UPB IPv6 multicast address and lifetime is an integer value specifying how long the address is valid for. A Collide message holds only one value, a valid UPB IPv6 multicast address which is in use by the sender.

All DAOMAP messages must be sent to a previously defined address. Though this might seem as if static assignment is needed for the protocol, in reality IPv6 already defines `ff02::1` as a group which all nodes on a link must join at network initialisation. DAOMAP uses the all-nodes address for both Claim and Collide messages. It is possible that the all-nodes address, which is link specific, is not sufficient for organisation-wide collision detection if proper subnetting has not been employed. In this case, increasing the address scope will solve the problem.

A DAA is run for each network interface separately. The API should make it possible for the calling application to select an appropriate interface, or insert a default interface if one is not provided by the user.

Participating DAAs keep two lookup tables per interface, ADDRESSES and NODES. ADDRESSES stores information about allocated addresses and NODES holds information about other DAAs which is used in predicting collisions. For practical purposes these tables are limited in size. ADDRESSES keeps a number of records, one per allocated address, with the format (unicast address, address, lifetime, local). The new fields unicast address and local are respectively the unicast address of the node to which the multicast address is allocated and a boolean value. The other fields have identical definitions to their respective fields in the Claim message. NODES is made up of a number of records, one per DAA, with the fields (unicast address, address count, collision count, last allocated multicast address, message ID). Unicast address is the IPv6 unicast address of a DAA, addresses count and collision count are integer values recording the number of multicast addresses assigned to a DAA and the number of collisions that DAA has reported. Last allocated multicast address is a valid UPB IPv6 multicast address holding the last address a DAA issued a Claim message for and message ID records the last seen message.

[2] A UDP approach can easily be implemented in user-space, an ICMPv6 approach lends itself to kernel-space implementation. Both are feasible.

[3] Sans-serif font indicates a data field in a message or table.

The protocol proceeds as follows: On startup each DAA determines from the routing subsystem the unicast network prefix for the interface the DAA is running on, and this is stored along with the lifetime of the prefix. The ADDRESSES and NODES tables are initially empty. The DAA constructs an initial address where the group ID is based on some global token specific to that interface, such as a MAC address. Where such a token is not available, a random group ID is chosen. This initial address is then acquired according to the claim process below. If a collision is detected for the initial address, a new group ID should be randomly chosen.

When the unicast prefix lifetime expires, the DAA must check the unicast prefix for changes (which is advertised by local routers). If the prefix has changed then the NODES table is cleared and any records in the ADDRESSES table which have the local value set to false are cleared.

Whenever an application requests an address via the API, the DAA generates a group ID G, which is $(96 - unicast\ prefix\ length)$ bits long. G is formed by inputting the last successfully allocated address into a deterministic algorithm, and selecting the required number of bits from the output [4]. It is imperative that **all** nodes using DAOMAP use the same address generation algorithm. An address A of the form shown in Figure 1 is then constructed with G. A lookup is performed in the ADDRESSES table for A. If A exists in the table, then two facts become clear; (1) an address collision will occur if the DAA tries to claim A and (2) any new addresses generated with A as input to the generation algorithm may also produce a collision at some point in the future. Because of (2), the protocol departs from a deterministic approach and randomly generates group ID $G\prime$. The collision avoidance is also applied to $G\prime$ (lookup and compare in ADDRESSES) and this selection process proceeds iteratively until a $G\prime$ is found which does not produce a clash in ADDRESSES.

The next step in collision detection tries to predict future Claims from other DAAs. The last allocated address from each record in the NODES table is passed into the group ID generator to produce the sequence of group IDs $G_1, G_2, ..., G_n$. If any $G_i = G\prime$, where $0 \leq i \leq n$, then future collisions are likely. A new $G\prime$ is generated randomly and collision detection restarts. When a group ID is found which the DAA believes will not cause a collision, the UPB multicast address A is formed.

Once A is determined, it is placed inside the address field of a new Claim message. The lifetime field is set to some length of time, dependent on the application. The message ID is set to a previously unused value and stays constant for this message, which is sent three times separated by t number of seconds. t is dependent on the time expected for a packet to reach all nodes on the `ff02::1` group; one second should be sufficient for most modern networks.

If no Collide messages are received by the claimant within t seconds after the third Claim message is sent, then the record $(address, lifetime, true)$ is added to the ADDRESSES table, and the address is returned to the requesting application for

[4] While we will not discuss a particular deterministic algorithm, any pseudo-number generator which produces a uniform distribution across the address space will suffice.

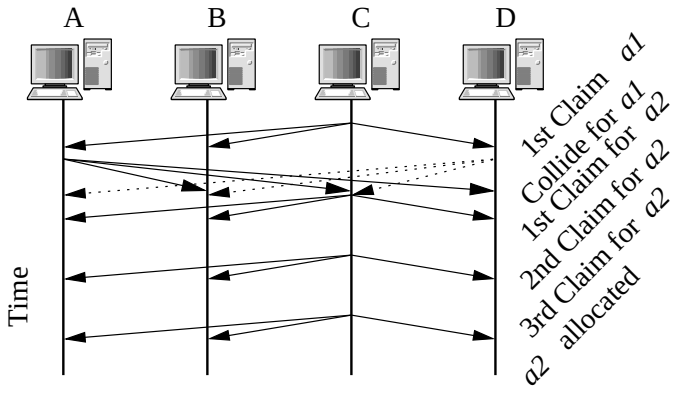

Fig. 2. DAOMAP packet exchange

use. Typically the application will then invoke a protocol such as MLDv2 [11] which informs routers of the nodes interest in a group, and multicast routing commences.

In the case where a Collide message has been received before the address is added to the ADDRESSES table, a new group ID is generated randomly as above, and the allocation procedure restarts. In the very unlikely event that a Collide message arrives after the address is added to ADDRESSES, we assume this is the result of some error or malfeasance on the part of a DAA and ignore it.

If the lifetime of an address expires it is removed from the ADDRESSES table and the address count field for the relevant DAA is decremented in NODES. Further, when the local field is true, three Claim messages for the address are issued (with constant message ID) as for normal address claiming, but with the lifetime set to zero. This effectively removes the address and decrements the address count from every ADDRESSES and NODES table on the link. Note this does not prevent the application from continuing to use the address, but we assume a well-behaved application will renew its address if it wishes to continue using the soon-to-expire address. When an application uses the API to explicitly release an address, the DAA proceeds as for an address expiration by sending three claim messages with the lifetime equal to zero.

When a DAA receives a previously unseen Claim message, the DAA extracts both the source unicast and malloc addresses, as well as the lifetime from the message and performs a lookup on the relevant record R in its ADDRESSES table. (An unseen message is verified by comparing the message ID of the Claim message with the message ID recorded in NODES.) If R exists then one of two scenarios is playing out. (1) A DAA is changing the lifetime of an address due to renewal or expiration, or (2) a collision has occurred. If the former is true then the DAA checks that the source of the Claim matches the unicast address field from R and updates R accordingly. If the latter is true and the local flag is set, then three Collide messages (with identical message IDs) are immediately dispatched containing the address from R. If an address is not local then the program generates a random number and tests whether this number exceeds a threshold (Section VI). If so, a Collide sequence as described above is issued. The collision count field in NODES for the sender of the Claim packet is also incremented.

Records in NODES are created whenever a previously unknown DAA issues either a Claim or Collide.

If a DAA receives a previously unseen Claim message for an address A, and no collisions are discovered by the DAA, then the address count for the claimant is incremented in the NODES tables, the last allocated address field is set to A and a record for the address is inserted into ADDRESSES.

In the event that a previously unseen Collide for address A is received from sender S, the DAA checks whether the collision involves an address currently in the process of being acquired by the DAA. If so, the claim is cancelled and address generation restarts, otherwise the claimant's unicast address A_{clm} is found by searching for A in ADDRESSES (and ignoring the entry where address is A and unicast address is S.) Secondly the Collide message issuer's unicast address A_{col} is extracted, and collision count is incremented in NODES for both A_{clm} and A_{col}, address count for A_{clm} is decremented and its last allocated address field is cleared.

Figure 2 illustrates how Claim and Collide packets are sent: C tries to allocate address $a1$, which is in use by A, who replies with a Collide packet. D also replies with Collides, to ensure with high probability that C receives at least one Collide. C then tries to allocate $a2$, by sending out three Claim messages. No Collides are received, so $a2$ is allocated to C.

VI. Security Issues

Previous texts dealing with the malloc problem have been short on details concerning security. Possible failings include forging or spoofing attacks, replay attacks or various types of Denial-Of-Service attacks.

The use of the IPv6 Authentication Header [12] (AH) and/or Encapsulating Security Payload [13] (ESP) secures the contents and ensures the integrity of messages. Attackers cannot change the contents of messages, and so messages cannot be forged, spoofed or altered. The use of these headers requires key distribution amongst DAOMAP participants, which is beyond the scope of this paper.

Including a changing message ID in a AH or ESP packet prevents replay attacks, since DAAs drop previously seen messages without taking any action.

Numerous DOS attacks exist; an attacker could try claim all addresses in an address range, or issue Collide messages to all Claim messages, or repeatedly Claim messages already allocated, simply flood an existing group with meaningless traffic or even a combination of the above attacks while continually changing MAC and IPv6 addresses.

The first case is dealt with by DAAs having a common threshold on the amount of addresses a single DAA can be allocated. If the address count of a DAA exceeds this threshold, then each DAA generates a random number and counts the number of entries n in NODES. If the random number is smaller than $1/n$ then a Collide message is issued. The reason for generating a random number is so that not all DAAs issue a Collide. This is advantageous because whenever a Collide is issued the issuer's collision count is increased. As we will see, this is not healthy for DAAs. Possible thresholds for address count can be found in Table III, it is a simple

matter to calculate them based on the expected number of nodes and the size of the group ID.

The second DOS is prevented by DAAs keeping a count on collisions issued by other DAAs. The collision count field in NODES is incremented whenever a collision is detected between two parties. If this value crosses a threshold for DAA D, then other DAAs assume D has evil intent and ignore messages originating from D. Thresholds for preventing this type of attacks are envisaged to be time-based; x Collides per timeunit are allowed, after which the node is ignored.

The third DOS occurs when an attacker issues Claims for addresses known to be allocated to a DAAs. As Collides are issued, the collide count for both attacker and allocatee are incremented, possibly leading to both being ignored. This attack is difficult to prevent, we resign exploration to later work. Our protocol also includes the facility for other DAAs to reply on the allocatee's behalf, which could lead to the DOS just described. Avoiding this is a matter of DAAs randomly responding based on a threshold, we suggest this threshold be such that more than one Collide is issued.

The fourth type of DOS is a flooding of the multicast group by an attacker. These crude but effective attacks can be stopped with the use of Source Specific or Source-Filtered Multicast (SSM [14], SFM [11]) which filter multicast traffic based on sources the listeners want to receive from.

Finally, external applications such as `arpwatch` [15] can be used to keep track of assigned addresses.

VII. Related Work

A stateful solution is MADCAP [7] which provides DAOMAP-like capabilities on network egdes. However it is not distributed and requires a MADCAP server to lease addresses. MADCAP is run in conjunction with MASC [5] which distributes addresses between domains.

DHCPv6 [16] can be used to provide multicast addresses, as [2] shows.

Handley, Perkins and Whelan developed SAP [17], which is a combined session announcement and implicit address allocation scheme. Addresses are randomly assigned from the free pool.

The Zeroconf Working Group at the IETF issued two drafts for the Zeroconf Multicast Address Allocation Protocol [18] (ZMAAP) before work was abandoned. Out of all the protocols examined, ZMAAP is the closest in function to DAOMAP. It is distributed and works on a claim/collide basis, but addresses are allocated randomly and no support is in place for solving the security concerns we listed in Section VI. The objective of the Zeroconf effort was different from ours, they were trying to produce an address allocation scheme which worked without any manual configuration while we are striving to develop secure distributed address allocation.

VIII. Conclusions and future work

In this paper we have explored the state of current malloc solutions. We found that most suffered limitations specifically due to legacy IPv4 standards, and distilled our requirements into a set of eight components. We proposed a new malloc protocol which took advantage of superior IPv6 address formats and satisfied each one of the eight constraints.

Our protocol introduces enforcement of security policies by limiting the numbers of groups a node may claim, while ignoring those nodes which issue too many collide messages. Along the way we discussed the relationship between network size and the expected number of collisions, and produced tables which will help implementors in deciding on values for key thresholds.

DAOMAP forms that basis for a secure multicast protocol we are working on. Once an application has a degree of confidence in the validity of a multicast address it has been assigned, new approaches to securing the traffic to and from the group become possible.

References

[1] D. Zappala, V. Lo, and C. GauthierDickey, "The multicast address allocation problem: theory and practice," *Computer Networks*, vol. 45, no. 1, pp. 55–73, 2004.

[2] R. Droms, J. Durand, P. O'Hanlon, J. Pansiot, B. Tuy, and S. Venaas, "IPv6 multicast address allocation study," 6net, Tech. Rep. 32603/RENATER/DS/3.4.3/A1, Apr. 2003.

[3] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *IEEE Network*, vol. 14, no. 1, pp. 78–88, / 2000.

[4] D. Thaler, M. Handley, and D. Estrin, "The Internet Multicast Address Allocation Architecture," Internet Engineering Task Force, RFC 2908, Sept. 2000.

[5] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley, "The MASC/BGMP architecture for inter-domain multicast routing," in *SIGCOMM'98*. ACM Press, 1998, pp. 93–104.

[6] P. Radoslavov, D. Estrin, R. Govindan, M. Handley, S. Kumar, and D. Thaler, "The Multicast Address-Set Claim (MASC) Protocol," Internet Engineering Task Force, RFC 2909, Sept. 2000.

[7] S. Hanna, B. Patel, and M. Shah, "Multicast Address Dynamic Client Allocation Protocol (MADCAP)," Internet Engineering Task Force, RFC 2730, Dec. 1999.

[8] M. Handley, "Session directories and scalable internet multicast address allocation," in *SIGCOMM'98*. ACM Press, 1998, pp. 105–116.

[9] B. Haberman and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses," Internet Engineering Task Force, RFC 3306, Aug. 2002.

[10] R. Hinden and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture," Internet Engineering Task Force, RFC 3513, Apr. 2003.

[11] R. Vida and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6," Internet Engineering Task Force, Tech. Rep. draft-vida-mld-v2-08.txt, Dec. 2003, expires June 2004.

[12] S. Kent and R. Atkinson, "IP Authentication Header," Internet Engineering Task Force, RFC 2402, Nov. 1998.

[13] ——, "IP Encapsulating Security Payload (ESP)," Internet Engineering Task Force, RFC 2406, Nov. 1998.

[14] S. Bhattacharyya, "An Overview of Source-Specific Multicast (SSM)," Internet Engineering Task Force, RFC 3569, July 2003.

[15] http://www-nrg.ee.lbl.gov/.

[16] R. D. (Ed.), "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," Internet Engineering Task Force, RFC 3315, July 2003.

[17] M. Handley, C. Perkins, and E. Whelan, "Session Announcement Protocol," Internet Engineering Task Force, RFC 2974, Oct. 2000.

[18] O. Catrina, D. Thaler, B. Aboba, and E. Guttman, "Zeroconf multicast address allocation protocol (zmaap)," Internet Engineering Task Force, Tech. Rep. draft-ietf-zeroconf-zmaap-02.txt, Oct. 2002, expired April 2003.

Marco Slaviero Marco Slaviero is frantically pursuing an MSc in Computer Science at the University of Pretoria, to add to his BSc (Hons) in Computer Science from Rand Afrikaans University, Johannesburg. In between IPv6 multicast research and Linux experimentation, he finds time to rock climb.

Martin S Olivier Martin S. Olivier is a full professor in the Department of Computer Science of the University of Pretoria in Pretoria, South Africa. His research interests include security of databases and applications, privacy, data communications and operating systems. His home page is available at http://mo.co.za

M. Slaviero and M. S. Olivier, "ĐAOMAP: A distributed IPv6 multicast address allocating protocol," in *Southern African Telecommunication Networks and Applications Conference 2004 (SATNAC 2004)*, D. Browne (ed.), Stellenbosch, South Africa, September 2004. Published electronically.