Secure and Distributed Multicast Address Allocation on IPv6 Networks

M. L. Slaviero

Secure and Distributed Multicast Address Allocation on IPv6 Networks

by

M. L. Slaviero

Submitted in partial fulfillment of the requirements for the degree

Magister Scientia (Computer Science)

in the

Faculty of Engineering, Built Environment and Information Technology

at the

University of Pretoria

October 2004

Secure and Distributed Multicast Address Allocation on IPv6 Networks

by

M. L. Slaviero

Abstract

Address allocation has been a limiting factor in the deployment of multicast solutions, and, as other multicast technologies advance, a general solution to this problem becomes more urgent.

This study examines the current state of address allocation and finds impediments in many of the proposed solutions. A number of the weaknesses can be traced back to the rapidly ageing Internet Protocol version 4, and therefore it was decided that a new approach is required. A central part of this work relies on the newer Internet Protocol version 6, specifically the Unicast prefix based multicast address format.

The primary aim of this study was to develop an architecture for secure distributed IPv6 multicast address allocation. The architecture should be usable by client applications to retrieve addresses which are globally unique.

The product of this work was the Distributed Allocation Of Multicast Addresses Protocol, or DAOMAP. It is a system which can be deployed on nodes which wish to take part in multicast address allocation and an implementation was developed.

Analysis and simulations determined that the devised model fitted the stated requirements, and security testing determined that DAOMAP was safe from a series of attacks.

Keywords: address allocation, algorithms, distributed systems, IPv6, multicast, protocols, security

Supervisor: Prof. M.S. Olivier

Department of Computer Science

Degree: Magister Scientia

Acknowledgements

The following people have been instrumental in the completion of this dissertation, and without their help, encouragement and support my journey would have been that much harder.

- Professor Martin Olivier, for his thorough and professional supervision.
- My parents, Luigi and Lucy, for the foundation they provided.
- Juliette, for her patience and kind words.
- The members of the ICSA research group, especially Vafa Izadinia, for the feedback and conversation they supplied.
- The staff of the Department of Computer Science, for the vision they display and advice they dispense.
- The financial assistance of the Department of Labour (DoL) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the DoL.

Table of Contents

Chapters

List of Figures

List of Tables

List of Algorithms

Listings

Chapter 1

Research Overview and Objectives

1.1 Introduction

The Internet Protocol Multicast standard was released in 1989 [1], and heralded a new form of communication on IP networks. For the first time, a source could send a single packet and reach a set of recipients; and this break through had major implications for technologies such as conferencing and video broadcasting.

Indeed, in their proposal for on-demand video based on IP Multicast, Little et al. [2] stated that *"future multimedia information systems will have a dramatic effect on the dissemination of information to individuals . . . [providing] services including games, movies, home shopping, banking, health care, electronic newspapers/magazines, classified advertisements, etc."* It was envisaged that these services could use multicasting to streamline data delivery. Other projected multicast uses included distributed databases and file stores, according to Ngoh and Hopkins [3].

However, fifteen years after its introduction IP Multicast still has not seen popular uptake by general users. Private conversation with technicians from a large Internet Service Provider in South Africa revealed that in order for their customers to make use of multicast, special arrangements must be

made since the ISP does not, in general, provide multicast services to users.

Why is the exploitation of a resource which promised so much, so limited?

A common argument made is that many devices do not support multicast, and this is examined in a later chapter. Jonathan Barter, a self-proclaimed multicast evangelist, believes that better marketing is required to increase multicast use: *"The [satellite communications] industry keeps assuring itself about the potential for multicast services, yet has made an appalling job of communicating the advantages to the outside world"* [4].

As will be seen, there are numerous obstacles standing in the way of widespread multicast use on IP networks. This dissertation is focused on just one of these hurdles, the address allocation problem [5]. This problem manifests itself whenever applications require multicast addresses on an ad-hoc basis; such applications might include instant messaging or conferencing where sets of friends or colleagues join transient groups which fall away once their purpose has been achieved.

Current address allocation schemes are static in nature, or require the roll-out of an ungainly multi-tier architecture which contains undefined standards. These restrictions have certainly limited multicast deployment; if addresses could be assigned in a dynamic and reusable fashion, developers could more easily produce multicast-enabled software since addresses would not need to be reserved or clumsy allocation structures utilised.

Certainly the rigid IPv4 address format has not helped matters. The lack of addresses along with a flat routing space combine to limit design choices for a multicast address allocation scheme in IPv4.

The preceding paragraphs have briefly laid out the status quo, with details to follow. The aim of this work is then to determine a set of requirements which an allocation scheme should fulfill in order to be scalable and usable, and develop a model which satisfies the proposed requirements.

1.2 Problem Statement

We have chosen to explore the opportunities which the newer IPv6 standard affords. A much larger and more flexible address format allowed for the

creation of a special address type, the unicast based prefix address format, which permits the construction of multicast addresses unique to a network. This address type forms the basis for the author's proposed allocation design.

In conjunction with the address type, a network protocol is also needed to ensure that addresses are legal and unique amongst participating nodes.

The primary problem is then to develop an architecture for secure distributed IPv6 multicast address allocation. The architecture should be used by client applications to retrieve addresses which are globally unique. Secondary problems are investigating IPv6 and multicast paradigms so that an optimal solution is developed, and ensuring that address use is limited in both time and quantity.

An implementation of the model will be developed as a proof of concept in answer to the problems posed.

1.3 Research Methodology

Firstly, the current state of address allocation will be thoroughly explored in an survey of the literature. Problems which have been identified by various authors will be expounded upon, and the strengths of each scheme should also be noted. From this survey, a set of requirements which a proposed allocation architecture must fulfill, will be deduced.

Once the requirements have been identified, a model which satisfies the determined requirements will be constructed. A primary objective for the model is that it must be feasible to implement and deploy; further, it should introduce new ideas and not simply ape previous attempts.

Finally, an implementation demonstrating the viability of the model will be developed.

1.4 Overview

This dissertation has the following structure:

Chapter 2 provides an introduction to the Internet Protocol version 6. It

commences with a short history of the development of the IPv6 standard and goes on to define and explain the format of IPv6 addresses. Special attention is paid to the unicast prefix based address format and various configuration mechanisms are explained. The layout of the IPv6 packet is then provided, as well as a summary on how routing in a hierarchical address space functions. The chapter is concluded with an outline of transition mechanisms from IPv4 to IPv6.

A brief introduction to the User Datagram Protocol, on which the proposed network protocol runs, is provided in Chapter 3.

The next chapter, Chapter 4, deals with the basic questions concerning multicasting: What is multicasting and how does it improve on other delivery models? In what manner do listeners join groups? How are packets routed around IP networks? Is there more than one type of multicast model? These questions are answered here so that the reader may sufficiently understand multicasting in anticipation of Chapter 5.

In Chapter 5 we expand on the multicast address allocation problem. More background is provided by examining the state of the art, before an analysis is undertaken to determine what an allocation system should look like. A set of requirements is distilled from the analysis.

The main contribution of this work appears in Chapter 6 with the introduction of our model DAOMAP, the Distributed Allocation Of Multicast Addresses Protocol. The protocol is defined in terms of four components (data store, address generation module, network communications module and client communications module), and the functioning of each component is described and documented. It is also shown how the model fulfills the requirements proposed in the previous chapter.

Chapter 7 records the implementation produced to verify that DAOMAP works. The prototype is benchmarked, and simulations are carried out to ensure that the built-in security protections operate well.

Lastly, this research is concluded in Chapter 8, where a summary of the dissertation is presented along with pointers for possible future work.

Chapter 2

Internet Protocol Version 6

2.1 Introduction

We begin this dissertation with an overview of IPv6, since that protocol provides the platform upon which DAOMAP is built. Because IPv6 is so important to this work, it is vital that the reader understand why a newer protocol than IPv4 is required, and it follows that a short overview illustrating the progression from IPv4 to IPv6 is necessary.

Following on from that overview, the intricacies of IPv6 addressing will be recounted, with special attention paid to the unicast prefix based address, which is key to our proposal.

The IPv6 packet format is explained, and a brief description of how hierarchical addresses are assigned and how they aid routing is provided.

Finally, the chapter is concluded with an exposition of changeover mechanisms, designed to ease the transition from IPv4 to IPv6.

2.2 A brief Internet Protocol History

The fundamental technicalities of the Internet have changed little since the early 1980's when the protocols which carry data around the Internet were invented. In 1981, the Internet Protocol version 4 (IPv4) was standardised with the release of Request For Comment (RFC) 791, authored by Jon

Postel. He defined [6] the objective of IPv4: *"to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached."* He further stated that *"datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism of the internet protocol is the internet address."*

An IPv4 Internet Address is a unique 32 bit number assigned to each interface on an IPv4 network, such as the Internet. It is important to note that the term 'interface' is used, rather than 'machine' or 'computer'. This is because a single computer may have many interfaces, each with a unique address. Such a computer is generally termed 'multi-homed'.

IPv4 addresses are normally presented in the 'dotted-quad' notation, to aid readability. When writing an IPv4 address in the dotted-quad notation, consider the 32 bits[1] as a sequence of four 8 bit words. Each 8 bit word is converted into decimal, and the four decimal numbers are concatenated using a period to separate each number.

As an example, take the 32 bit address

$$\underbrace{10001001}_{137.}\underbrace{11010111}_{215.}\underbrace{00101000}_{40.}\underbrace{01110111}_{119}$$ or 137.215.40.119

There are three mechanisms for data transmission in IPv4, and the address structure determines which mechanism is to be used:

Unicast A datagram is sent from one interface to another, one-to-one communication. The range of addresses is 0.0.0.0 – 223.255.255.255, although many addresses in this range have special meaning.

Broadcast A datagram is sent from one interface to every other interface on the network, one-to-all communication. Broadcast addresses occur within the Unicast address space.

Multicast A datagram is sent from one interface to a group of interfaces, one-to-many communication. Multicast addresses are assigned from

[1] The 32 bit address is written with the most significant bit first, with each subsequent bit decreasing in significance, also known as the Big-Endian format [7].

the range 224.0.0.0 – 239.255.255.255.

The broadcast has two large deficiencies.

1. The address used in a broadcast is not easily identifiable without information about the network.

2. A broadcast sends data to all interfaces on a network, regardless of whether an application is waiting on the interface. This causes wasted processing time when a datagram arrives because even though an application might not be waiting for the data, the datagram is still passed to software where it is eventually discarded.

As we will see, IPv6 changes this paradigm by excluding the Broadcast and introduces a new address type, the Anycast.

IP provides a 'best effort' delivery mechanism. There are no guarantees that IP will deliver the datagram correctly and in the right order, if at all. We saw that IP is merely a carrier of datagrams from one node to another. Other protocols which are layered above extend the capabilities and usefulness of IP (one such example is the Transmission Control Protocol (TCP) which provides a reliable service). Figure 2.1 displays an example of this layering. IP sends packets via the Device Driver, while the Transmission Control Protocol uses IP to deliver packets, and an application uses TCP to transfer data.

The layering model should allow a module which occupies one layer to be replaced with another module which performs the same function. In reality this is not the case. For instance, TCP uses a checksum to verify the integrity of a packet and fields from the IP layer are used in this checksum. This binds TCP to IP, as the IP layer could not be replaced without changes to the TCP module[2].

[2]UDP suffers from the same limitation, as will be seen.

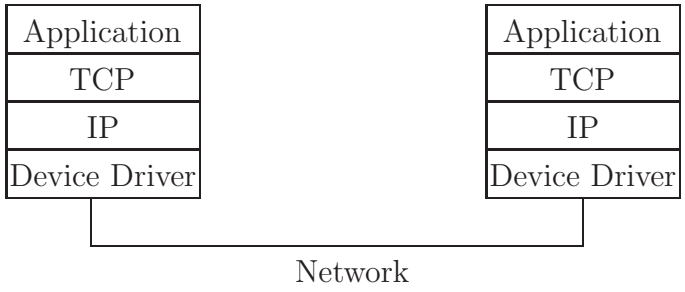


Figure 2.1: Protocol layering

2.3 Moving on from IPv4

In the previous section we very briefly examined IPv4, which has become the de facto standard for Internet communication. However, IPv4 has a number of drawbacks. The most widely published is the apparent shortage of IPv4 addresses. When the protocol was designed the engineers had no idea that their small research network would eventually become the largest computer network ever created, so 32 bits was thought to be large enough to handle future growth.

At that stage personal computers had just come onto the market, memory and other hardware resources were scarce and few computers were networked. The 32 bit format extended several advantages to programmers, as Huitema [8, p. 312] explains: *"The convenience of an address that could be stored in a standard 32 bit memory word as well as the programming efficiency of a well-aligned header were too attractive."*

He also notes however, that *"A more complex format . . . would not have attracted as many followers."* This would have hampered the explosive growth of IPv4 and the Internet; and therefore we can then observe that the decision to use 32 bits was not incorrect at that time.

A further consequence of the IPv4 design was that the routing tables on the core routers were growing extremely quickly, as more and more networks were connected (the so-called 'flat routing space' problem [9]). Although this was ameliorated with the introduction of Classless Inter-Domain Routing (CIDR), CIDR could not be seen as more than a finger in a dyke holding

the problem in check until the successor of IPv4 was ready. CIDR uses a scheme whereby IPv4 addresses are allocated in blocks, so that routers hold one entry for each block rather than one entry per network.(In fact, we have already passed the sell-by date for CIDR, according to a prediction made in 1994 [10])

A number of protocols were presented in 1993 as replacements for IPv4 as part of the process for choosing IPng (IP Next Generation). We will describe three of the main proposals very briefly, before tackling the protocol which was selected. The descriptions are a summary of RFC 1454 [9] and RFC 1752 [11] which outlined a number of candidates for IPv4 succession.

2.3.1 CATNIP

CATNIP (Common Architecture for the Internet [12]) planned to integrate various network layer protocols, specifically CLNP, IP and IPX. The authors recognized that as global communications became more tightly coupled to the Internet, ISO standards would come into play. *"ISO convergence is both necessary and sufficient to gain international acceptance and deployment of IPng. Non-convergence will effectively preclude deployment."* [12].

2.3.2 SIPP

SIPP (Simple IP Plus [13]) was the product of a merger between two proposals, SIP and PIP. SIPP attempts to retain the coding efficiency of SIP and the routing flexibility of PIP.

SIP

SIP (Simple Internet Protocol) was IPv4 with a 64 bit address space and reduced set of options [8, p. 313]. While its main advantage was its simplicity which resulted in more efficient handling, the 64 bit address space seemed dangerously inadequate considering the 32 bits used in IPv4 was not enough.

PIP

PIP (P Internet Protocol [14]) was an entirely new protocol. It introduced a new header format with fields whose meaning could differ. *"PIP gives the source the flexibility to write small 'programs' which direct the routing of packets through the network."* [9].

The most drastic change was altering the flat routing space to a hierarchical space. Packets could now be routed according to portions of their addresses, rather than having to lookup each address in a routing table. This solved the IPv4 routing table explosion problem. PIP addresses were *"effectively unlimited"* [9] in length, due to the hierarchical network topology.

2.3.3 TUBA

TUBA (TCP and UDP with Bigger Addresses [15]) was a proposal which used the Connectionless Network Protocol (CLNP). CLNP is an OSI protocol and is very similar in nature to IPv4 except that it has a variable address space of up to 20 octets. Historically the OSI protocols have been somewhat dated, as well as exhibiting *"convergence on the lowest-common denominator"* [9]. As an example we note that CLNP does not support *"multicast, mobility or resource reservation"* [8, p. 313]. The CLNP multicast standard is still deemed 'Experimental' [16] but was published in 1995.

TUBA basically offered a larger address space, TCP and UDP would remain and be modified to run over CLNP. TUBA also allowed for host autoconfiguration.

"[T]the main argument againt[sic] *TUBA is that it is rather too like IPv4, offering nothing other than larger, more flexible, addresses"* [9]. There was also an unwillingness amongst IETF members to use the OSI-sponsored CLNP and this forced them to develop the alternatives, SIP and PIP.

2.4 The Internet Protocol version 6

SIPP was ultimately selected as the basis for IPng, which was renamed IPv6 [17] (IPv5 was an experimental protocol called the Internet Stream Protocol.) Although IPv6 contains many improvements over IPv4, the basic objective stated in Section 2.2 for IPv4 is still applicable to IPv6.

According to Hagen [18, p. 3], IPv6 has five main enhancements over IPv4:

- Expanded address capability and autoconfiguration mechanisms
- Simplification of the header format
- Improved support for extensions and options
- Extensions for authentication and privacy
- Flow labelling capability

Each of these features will be dealt with in the remainder of this chapter, as we describe the IPv6 protocol.

2.4.1 IPv6 Addressing

This section is a summary of RFC 3513[19], which defines the IPv6 address structure.

An IPv6 address is 128 bits long. This is an increase of 27 orders of magnitude in the number of addresses supported compared with IPv4, and should satisfy our address space needs for the foreseeable future[3]. The address is written as a series of eight hexadecimal fields of 16 bits, separated by colons, for example

```
fe80:0000:0000:0000:02b0:d0ff:fee7:6ebe
```

[3]The precise definition of 'foreseeable future' is left as an exercise to the reader.

Abbreviations

Addresses can be unwieldy and difficult to remember which is why conventions have been introduced to help reduce the size of the written address. The addressing RFC 3513 [19] allows for leading zeros to be dropped in each field, leaving our previous address as

```
fe80:0:0:0:2b0:d0ff:fee7:6ebe
```

A further convention allows a single group of successive zero fields to be replaced with a double colon "::", so that our address now looks like

```
fe80::2b0:d0ff:fee7:6ebe
```

Note that only one double colon may appear in an address, otherwise the address is no longer determinate (for example it is not possible to state without ambiguity what address `fe80::1::` expands into).

Address Types

Similarly to IPv4, there are three types of IPv6 addresses:

Unicast An address which identifies a single interface, used in one-to-one communication.

Multicast An address which identifies a set of interfaces, used in one-to-many communication.

Anycast An address which identifies a set of interfaces. A packet[4] sent to an Anycast address is sent to the "nearest" interface, which is calculated by the routing protocol. This feature is still experimental, and implementation requirements and restrictions have yet to be fully explored.

As noted earlier, the IPv4 Broadcast has been removed and the Anycast has been introduced.

[4]Earlier we used the term "datagram" in accordance with RFC 791. The IPv6 standard speaks only of packets which is the convention that is followed from now on.

Address Prefixes

In order to route packets, an IPv6 implementation must be able to determine *"the subnet or a specific type of address"* [18, p. 30]. This is accomplished with the address prefix which indicates what portion of the address must be examined to extract the routing information. The address prefix notation has the following structure:

ipv6-address/prefix-length

Where *ipv6-address* is a legal IPv6 address and *prefix-length* is a decimal value indicating *"how many of the leftmost contiguous bits of the address comprise the prefix."* [19].

To continue the address example used previously, when written with its prefix the address would look like

`fe80::2b0:d0ff:fee7:6ebe/64`

This specifies that the first 64 bits of the address belong to the subnet. The length of the prefix depends upon the type of address.

Address Prefix Types

Whereas IPv4 initially used different "classes" of addresses in a flat-routing space, IPv6 uses an address hierarchy to decide where to send packets. Every address has a type prefix, which is a number of contiguous bits at the start of the address and determines the address type. The exact number of bits differs for various address types. Table 2.1 is taken from RFC 3513 [19] and lists the address types along with their prefixes. The prefix-length shows how many bits need to be examined in order to guarantee an address type.

The various address types will be defined shortly, suffice to say that there is no Anycast prefix as the Anycast addresses are assigned from the Unicast range.

The address types are standardised so that whenever an IPv6 application parses an address which starts with, for example `ff`, it immediately knows the address is a multicast address.

Address Type	Binary Prefix	IPv6 Notation
Unspecified	00...0	::/128
Loopback	00...1	::1/128
Multicast	11111111	ff00::/8
Link-local Unicast	1111111010	fe80::/10
Site-local Unicast	1111111011	fec0::/10
Global Unicast	(everything else)	

Table 2.1: Address Types and their prefixes

Thus prefixes can be used to determine both address type, and, by utilising a longer prefix length, determine which subnet the address is associated with in the case of a Unicast address.

2.4.2 Address Types

Address Scope

Before the various address types are defined, mention must first be made about 'address scope'. IPv6 uses the notion of address scope to define the span within the network in which an address is valid. The three scopes are Link-Local, Site-Local and Global, and below a brief description of each is given.

Link-Local An address which is used only between two interfaces for point-to-point communication, such as *"automatic address configuration, neighbor discovery, or when no routers are present."* [19]. Packets with Link-Local addresses as either Source or Destination are always discarded by routers, since these packets must never cross network boundaries.

Site-Local A unique address which is used within a network, but which is not globally routable. Routers must never forward Site-Local packets outside of the site. These addresses are analogous to the private address ranges of IPv4.

Global A unique address which can be used to route packets within the IPv6 Internet.

Every address must be unique within its scope.

Unicast Addresses

The three Unicast addresses referred to in Table 2.1 are used as unique identifiers for single interfaces according their assigned scope. Their construction varies, but generally use the form of $Prefix + InterfaceIdentifier$ and may be constructed automatically. The Prefix is determined according to the address type (Table 2.1) and network prefix, while the Interface Identifier is calculated to be unique on the link. Calculation of the Interface Identifier may use the IEEE Identifier (an identifier embedded in hardware, such as the Ethernet MAC address) of the interface or could be generated randomly for interfaces which lack IEEE Identifiers. Further details may be found in RFC 3513 [19].

An example of a Unicast address may be found at the start of Section 2.4.1.

Anycast Addresses

Anycast addresses are *"syntactically indistinguishable from Unicast addresses"* [19], as they are taken from the same range. The Anycast mechanism is new to the Internet, and at this stage only routers may be assigned Anycast addresses.

Multicast Addresses

A multicast address identifies a set or group of interfaces. These interfaces are generally (but not required to be) on different nodes. We will go into much more detail on multicast in Chapter 4, for now we will examine the multicast address structure.

The multicast address has the format $\underbrace{Prefix}_{\texttt{ff}}\ \underbrace{Flags}_{4\,bits}\ \underbrace{Scope}_{4\,bits}\ \underbrace{Group}_{112\,bits}$

eg. `ff0e:0:0:0:0:0:0:10a`

- The Prefix is always `ff`.
- Two bits of the Flags field are defined, to indicate if the address is 'well-known' or 'transient' and if the address is based on a Unicast prefix or not [20]. In our example the Flags field is `0`.
- The Scope field determines how far from the originator the packet will travel. We will examine scope more closely in a later chapter, in our example it is `e`, indicating the address has global scope.
- The Group field contains an identification for a multicast group, in this case `0:0:0:0:0:0:10a`.

There are numerous pre-defined or 'well-known' multicast addresses assigned in the RFC. Here we will look at some of the more important ones:

ff02::1 All Nodes on the local-link

ff05::1 All Nodes on the local-site

ff02::2 All Routers on the local-link

ff05::2 All Routers on the local-site

Some restrictions are placed on multicast address use:

1. They cannot be used as Source addresses.
2. They must not be forwarded past the scope indicated by the Scope field.

Of further interest to us is the Unicast prefix based multicast address defined in RFC 3306 [20]. This type of multicast address is dependent on the Unicast prefix assigned to a network by a Registry. To appreciate the importance of this address type, allow us to look at options available under IPv4 for multicast address allocation:

1. Manually register a well-known address with IANA[5].

[5]IANA, or the Internet Assigned Numbers Authority, is responsible for assigning various Internet-wide numbers and addresses, for example port numbers or common addresses.

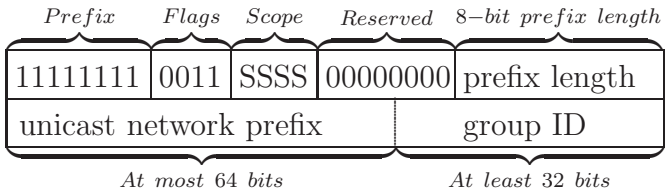


Figure 2.2: Unicast prefix based multicast address

2. Register as a user of the `233/8` address space and IANA allocates a 24 bit prefix to the registree.

3. Configure the cumbersome architecture described in Section 5.2.

With the addition of unicast prefix based multicast addresses, each of these options becomes unnecessary for IPv6. Since each organisation on the IPv6 Internet is assigned a unique Unicast prefix, an organisation can generate its own multicast addresses based on its own Unicast prefix.

Figure 2.2 depicts the format of a unicast prefix based IPv6 multicast address [20]. The four **S** bits indicate the scope of the multicast address, which ranges from interface-local to global [19]. The **prefix length** field denotes how long the unicast network prefix field is, which is at most 64 bits. Clearly the actual unicast prefix of the network 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 a subnet, and it is this field which must be unique within that subnet. Combined with a unique global unicast prefix, we can form globally unique multicast addresses.

Unspecified Address

The Unspecified address has the abbreviation `::/128` and is *never* assigned to any interface. It can be used as a Source address during autoconfiguration, but must not be used as a Destination address. The Unspecified address *"indicates the absence of an address"* [19].

Loopback Address

The Loopback address has the abbreviation `::1/128`, is a Link-Local Unicast address and is never assigned to a physical interface. A node may use the Loopback address as a Source or Destination address, but the packet is never sent on a physical interface. The Loopback address can be used by a node to communicate packets to itself.

Autoconfiguration

A major weakness of the IPv4 protocol was the fact that addresses were either assigned manually, or special services such as DHCP (Dynamic Host Configuration Protocol) had to be used. IPv6 corrects this with the inclusion of autoconfiguration, which is defined in two broad flavours[6]:

Stateless The autoconfiguration of addresses and routing information from node-held data and router advertisements. More details can be found in RFC 2462 [21].

Stateful Autoconfiguration using information which is obtained from a server, much like IPv4's DHCP. It is specified in RFC 3315 [22];

Stateless and stateful autoconfiguration complement each other. A node can obtain network connectivity using stateless autoconfiguration and then use stateful autoconfiguration to assign a host-name, domain name or DNS server. We will only cover stateless autoconfiguration here.

Autoconfiguration can only take place on multicast-enabled interfaces.

It was mentioned earlier that a Unicast address may be constructed automatically with a *Prefix* and an *Interface Identifier*. This forms a link-local address which is not yet guaranteed to be unique and is termed "tentative". When the node configures the interface, it will first send out a Neighbour Solicitation packet with the destination address equal to the tentative address. If no replies are received within a time period, the tentative address

[6]Within the IPv6 Working Group at the IETF, the terms 'stateless'/'stateful' are frowned upon as they do not adequately categorise all autoconfiguration possibilities. However, for our purposes and in the interest of simplification, their use is appropriate here.

can be presumed to be unique and the link-local address is assigned to the interface. If a reply *is* received then autoconfiguration has failed and manual intervention is required to assign a valid address.

At this point the node has link-local IP connectivity. What remains is to determine routing information, if indeed a router is present. This is done by sending a Router Solicitation message to the All-Routers Multicast group. If a router is active on the link it will respond with information necessary to obtain site-local or global connectivity. This can include prefixes to be used and indicate whether stateful autoconfiguration is in effect.

The advantages of autoconfiguration are clear:

- Users can simply 'Plug and Play'.
- Administrators do not have to deal with the propagation of erroneous information, since configuration information is either determined automatically by means of a standardised process, or provided from a central server.
- Site renumbering would be the simple case of changing the prefix which the site routers advertise and waiting for the leases on the old addresses to expire.

Address Privacy

The autoconfiguration mechanism has reduced the effort involved in configuring a network, but the use of devices with IEEE Identifiers introduces the problem of a permanent address. The specific concerns are beyond the scope of this document, but generally permanent addresses enable easier tracking of users' network habits than if they had a continually changing address.

RFC 3041, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", describes how the division of the IPv6 address into *Prefix* + *Interface Identifier* facilitates tracking of users across different networks. When a user plugs into another network, the *Prefix* will change but the *Interface Identifier* remains the same. Since the Identifier is often globally unique, the user can be monitored even if he decides to change network service provider.

The RFC goes on to propose a framework whereby each node has both "public" and "temporary" addresses. The public address, often stored in DNS, is known to other nodes and is the address to which they will connect. The temporary addresses are generated in sequence, used when initiating communication with other nodes. These temporary addresses would be based on the Interface Identifier and produced by an MD5 hash, with a random component mixed in.

When a temporary address has been used for a certain length of time (the RFC does not set limits, but mentions "hours to days"), it will be marked as deprecated and the next address in the sequence will be used for all new traffic streams originating from the node.

A continually changing address, while greatly benefiting users, may cause some level of discomfort for network administrators who must debug and maintain networks where the addresses are transient.

2.4.3 Header format

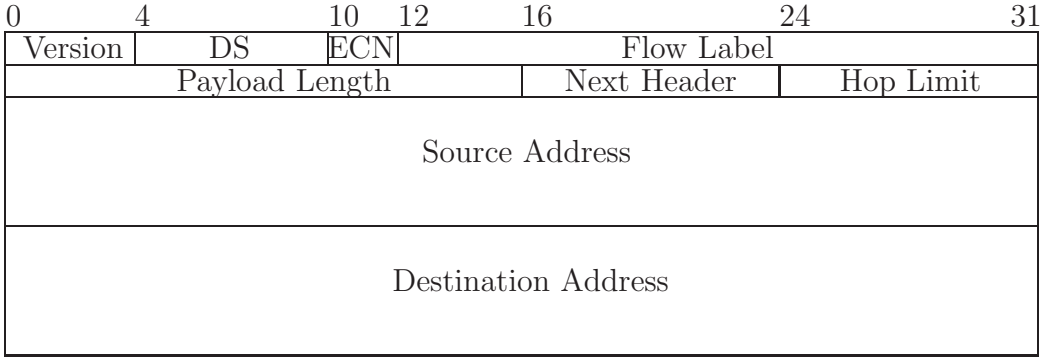


Figure 2.3: IPv6 Header

Figure 2.3 shows the layout of the basic IPv6 header which every IPv6 packet is required to have. The figure is organised in a series of 32 bit rows. The numbers across the top indicate the bit positions of the fields, from which fields sizes can be calculated. Below is a brief description of each field, with most of the information being drawn from RFC 2460, "Internet Protocol, Version 6 (IPv6) Specification" [17].

Version This field indicates the version of IP. It is always `0x06`.

Differentiated Service Modern networks may be used to carry many kinds of data, and each type of data may require different handling from the network. These may be defined in *"quantitative or statistical terms of throughput, delay, jitter, and/or loss, or may otherwise be specified"* [23].

This field is used by routers and hosts to determine the requirements of the packet with regards to the network, and to attempt (if allowed) to satisfy that request.

The DS field is backwards compatible with the IP Precedence field of IPv4 [24].

Explicit Congestion Notification Traditionally network congestion has been measured by the number of packets dropped within the network ie. packets cannot be added to full queues so they are discarded. With the addition of ECN, nodes can detect possible congestion before packets are lost and notify other communication participants of the situation.

The ECN specification requires the Transport Layer to work in conjunction with IP.

More details can be found in RFC 3168, "The Addition of Explicit Congestion Notification (ECN) to IP" [25].

Flow Label A flow can be thought of as a related set of packets (eg. packets in a specific TCP connection). This field is used to uniquely mark flows so that nodes can apply special treatment to those data streams and is specified in RFC 3697 [26].

The originator of the packet generates an ID for the flow which is identified by the 3-tuple $(FlowLabel, SourceAddress, DestinationAddress)$ and inserts the ID into the packet.

Either the Source or the Destination address may be wild-carded so that the packets from multiple hosts will be treated as part of the same flow (eg. multicast with multiple senders.)

Any host or router which does not support Flow Labels must ignore Flow Labels in packets destined for the node, leave the label as is when forwarding a packet, and insert a 0 when sending a packet.

It is recommended that all flows be labelled, even if the sending node does not require flow services as the recipient can use flows to aid load balancing.

Nodes are also required to store recently used IDs, so that they are not reused within 120 seconds.

Payload Length The length of the packet, excluding the IPv6 header which is always 40 octets. This is different from IPv4 where the header contained two length values, the header length and the total packet size.

Next Header While IPv6 has a fixed header to enable more efficient processing, there is support for so-called "Extension Headers". These are optional and carry network layer information in the packet, where they are placed between the IPv6 header and the payload. Some of the extension headers include Routing and Destination Options headers, as well as encryption headers.

The Next Header field describes the type of header which follows the IPv6 header. In the case where there are no extension headers, the Next Header field might contain a value indicating that a TCP header follows. Each extension header has a Next Header field, so it is possible to chain headers together.

eg.
$IPv6Header \rightarrow TCP \rightarrow Payload$
$IPv6Header \rightarrow Routing \rightarrow DestinationOptions \rightarrow UDP \rightarrow Payload$

More headers are defined in RFC 2460 [17].

Hop Limit Every time a packet is forwarded its Hop Limit is decremented by one. If the Hop Limit reaches 0, the packet is discarded. This ensures that packets caught in routing loops will not circulate indefinitely, and can be used to limit packets to a certain radius.

This is especially useful for multicast experiments where one might not want packets to cross over any routers. In this case one could simply set the Hop Limit to 1 and be assured that any router which receives the packets will decrement the Hop Limit and discard the packet.

The Hop Limit replaces the IPv4 Time To Live field.

Source Address The IPv6 address of the originator of the packet.

Destination Address The IPv6 address of the destined recipient. This might not be the final recipient if a Routing extension header is contained in the packet.

2.4.4 Global Unicast Routing

It has already been stated that IPv6 is designed to use a hierarchical address space, but what does this actually mean and how does this aid routing? Let us first look at the format for a Global Unicast address [19] (Figure 2.4):

Global Unicast Addresses and Assignment

Figure 2.4: Global Unicast Address Format

The Global Routing Prefix together with the Subnet ID identifies a single globally unique subnet, and the Interface Identifier points to a specific machine on that subnet.

- The Global Routing Prefix is handed out to an organisation by a Registry. The Registry may, in turn, have been delegated its address space from a higher Registry. For example, a South African university must apply to the Local Internet Registry TENET [27] for IPv6 addresses because TENET provides the universities with Internet access. TENET was assigned its global Unicast prefix from ARIN [28] and that prefix is `2001:0548::/32`.

- The Subnet ID is assigned by the network administrator.
- We have already seen how the Interface Identifier is determined.

A university which applies to TENET for IPv6 addresses will receive a /48 prefix from the `2001:0548::` netblock, assuming its application is approved. Within the university's prefix there is space for 2^{16} subnets, since the Interface Identifier is generally 64 bits.

Benefits to Routing

This hierarchical assignment of addresses allows for route aggregation. Core routers need only store a single route for `2001:0548::/32` to reach any South African university which has a TENET-assigned address. The significance of this cannot be overstated, as previously with IPv4 a route had to be stored for each network class before CIDR was introduced.

2.4.5 IPv4 to IPv6 changeover

An overnight switch from IPv4 to IPv6 is unrealistic and this was recognised in RFC 2893, "Transition Mechanisms for IPv6 Hosts and Routers". *"The key to a successful IPv6 transition is compatibility with the large installed base of IPv4 hosts and routers" [29]."* The authors list and describe various schemes which aid in running IPv6 networks in an IPv4 environment and a summary is provided of both this RFC as well as other proposals for IPv6/IPv4 integration. Unless otherwise indicated the proposals are described in RFC 2893.

Dual Stack

The node has complete IPv4 and IPv6 stacks and both stacks may be used simultaneously.

Manually configured IPv6 over IPv4 tunnel

The system administrator sets up the tunnel by designating end-points. Each IPv6 packet has an IPv4 header prepended to it and the IPv4 destination

address is the end-point of the tunnel. Once the end-point receives the packet the IPv6 portion is extracted and sent on its way. An example of this is the Freenet6 service (www.freenet6.net) which allows users without an IPv6 network to tunnel into the IPv6 Internet.

Configured tunnels are most useful in cases where small numbers of machines connect via a tunnel as there can be a reasonable amount of work in configuring tunnels for each machine in a subnet. If there are numerous machines which require IPv6 access via tunnels then automatic tunneling might be better suited.

Automatic tunneling of IPv6 over IPv4

The main difference between automatic and configured tunnels is that with an automatic tunnel the end-point can be determined from the use of special IPv4-compatible addresses. The address has the prefix `0::/96` followed by the 32 bit IPv4 address. The tunnel is created by extracting the IPv4 address from the IPv6 address, that is the 32 lower order bits.

6to4

A technique used to connect IPv6 networks over IPv4 infrastructure without the need for explicit tunnel creation. The IPv6 network has border gateways which wrap the IPv6 packets in IPv4 and send them to the destination sites where they are stripped down and sent on their way. 6to4 is different from the previous schemes in that it connects *sites* rather than hosts to hosts or hosts to sites.

Detailed in RFC 3056 [30].

6over4

6over4 is a slightly older method of gaining IPv6 functionality which uses IPv4 multicast to 'simulate' Ethernet across many hops. This is in contrast to the tunnels which utilise point-to-point connections.

Detailed in RFC 2529 [31].

Intra-Site Automatic Tunnel Addressing Protocol

ISATAP is a draft proposal which will allow IPv6 nodes within a site to interact without the need for an IPv6 router. The IPv4 address of the recipient is embedded in the ISATAP address, along with a standard 64 bit prefix. If the site wishes to connect to the IPv6 Internet or other IPv6 sites, then a border gateway must be installed. This gateway could, for example, run a **6to4** tunnel.

Detailed in draft-ietf-ngtrans-isatap-15.txt [32].

2.5 Conclusion

This first important chapter has explained IPv6, especially with regards to addressing. The unicast prefix based format was introduced and documented, and the structure of IPv6 packets was put forth.

Configuration mechanisms and routing of IPv6 packets were touched upon, and the chapter finished with a description of various transition proposals.

As stated previously, this chapter is important in laying the foundation for our upcoming proposal.

The next chapter is describes the User Datagram Protocol, which is later exploited to transport data.

Chapter 3

User Datagram Protocol

3.1 Introduction

The User Datagram Protocol (UDP) is one of the cornerstone protocols which give the Internet its solid foundation. Along with IPv6 which was discussed in Chapter 2, UDP provides a platform for fast, connectionless delivery of packets. This chapter briefly examines UDP over IPv6 and attempts to explain why UDP is required when multicasting.

Let us begin with Comer's [33, p. 331] description of transport protocols, *"Transport protocols assign each service a unique identifier. Both clients and servers specify the service identifier; protocol software uses the identifier to direct each incoming request to the correct service."*

Within the transport protocol domain, two classes for passing data exist: connection-orientated and connectionless [33].

A **connection-orientated** protocol creates a link or circuit between communicating parties, across which data can be sent *reliably* and *in order*. An analogous example is a telephone conversation; when the telephone rings and is picked up, a link is established. The link is maintained until one of the parties hangs up.

A **connectionless** protocol has no such reliability characteristic. No guarantees are made with respect to: the order in which packets arrive, duplicate packet elimination, or indeed if the packet arrives at all (A

good analogy in this case would be the Postal Services. Once a letter has been dropped in a postbox, one can only hope for a successful delivery.)

The IPv6 suite uses the Transmission Control Protocol (TCP) for connection-orientated delivery, and UDP for connectionless delivery. Both protocols function above the IPv6 layer (Figure 2.1 from Chapter 2 shows where TCP, and hence the UDP layer, fits in.)

3.2 UDP

The UDP specification was standardised in the three-page RFC 768 [34], penned by Jon Postel. The brevity of the document underlines the simplicity of the protocol; Postel states that "*[UDP] provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism.*" The benefits of the small protocol overhead include faster processing of packets and fewer resource requirements on nodes.

UDP uses the notion of 'port numbers' to identify processes[1] (fitting Comer's definition of transport protocols.) The benefit is that we can have many UDP applications running simultaneously each receiving its own traffic.

3.2.1 Header

Let us examine the UDP header which is prepended to every UDP packet:

0	16	31
Source Port	Destination Port	
Length	Checksum	

Figure 3.1: UDP Packet Header

Figure 3.1 is formatted in rows of 32 bits, the numbers across the top indicating the bit positions of the fields. The fields are explained below.

Source Port A number which identifies the process on the originating node. This field is optional, and, if not used, should utilise a zero value.

[1] TCP uses the same mechanism.

Destination Port A number which is used by the receiver to determine which process the packet should be delivered to.

Length The length of the UDP header plus the UDP payload in octets.

Checksum *"[T]he 16 bit one's complement of the one's complement sum of a pseudo header of information from the IPv6 header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets."* [34]. It should be noted that in IPv4, the checksum was optional. However, IPv6 makes the checksum mandatory, and any UDP packets which arrives via IPv6 without a checksum must be discarded. We discuss the pseudo header next.

3.2.2 Pseudo Header

The pseudo header is constructed using fields taken from the IPv6 header and the transport-layer protocol, in this case UDP. It is used when calculating the checksum required by the UDP protocol. The pseudo header is depicted in Figure 3.2, which is defined in RFC 2460 [17, p. 27].

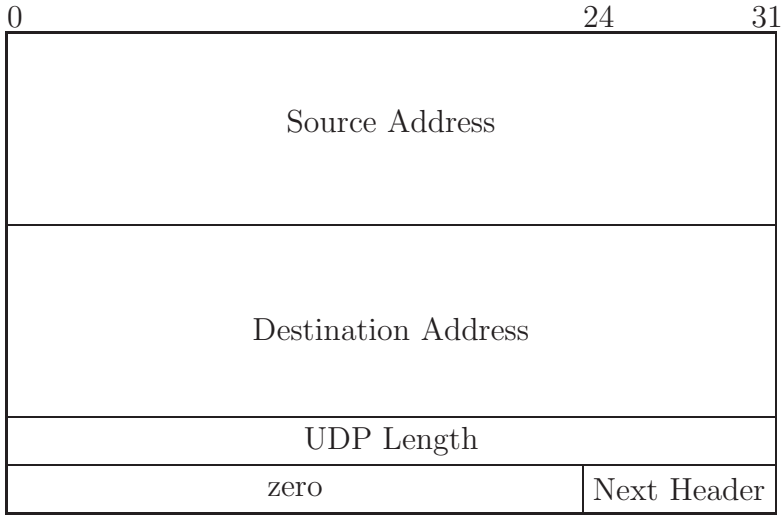


Figure 3.2: UDP Pseudo Header

The definition of the fields in Figure 3.2:

Source Address Address of the originating node.

Destination Address Address of the final recipient.

UDP Length Length of the UDP packet, including its header. This is not calculated, as the Length field from the UDP header is inserted. The pseudo header is not UDP specific, hence the Length field is not 16 bits but 32 bits.

Next Header The protocol number of the upper-layer protocol (17 for UDP).

In Section 2.2 we alluded to the fact that the dependence of the pseudo-header on the lower IPv6 layer, coupled with the mandatory checksum has tightly bound the UDP layer to the IPv6 layer. This is unfortunate considering the desire for independence between the protocol layers, but has the advantage that corrupted or incorrectly delivered packets will be detected.

3.2.3 Multicasting over UDP

There is no hard and fast rule requiring that multicast data be carried in UDP packets. Indeed IPv6 has support for numerous upper-level protocols, of which UDP is just one, and any upper-layer protocol may receive multicast packets if its designer so chooses. Whether the protocol is actually able to use multicast packets depends on its purpose. TCP, for instance, would not be a candidate for multicast packets according to Stevens; *"TCP is a connection-orientated protocol that implies a connection between* two *hosts (specified by IP addresses)"* [35, p. 169] (emphasis added).

The advantage in using UDP to carry multicast data is that UDP adds very little overhead to the process of packaging the data and sending it off, while adding the benefits of a transport protocol. UDP is a 'fire-and-forget' protocol, once the packet has left the node no assurances are provided regarding packet delivery. The host requirements for multicasting are similarly sparse: *"the datagram is not guaranteed to arrive intact at all members of the destination group or in the same order relative to other datagrams"* [1].

This close correlation between the needs of UDP and multicasting has lead to the de facto standard for delivery of multicast packets occurring over UDP.

3.3 Conclusion

This chapter briefly introduced UDP, giving a breakdown of the packet header as well as showing how the checksum is computed. We also discussed the reasons why UDP is generally used as the transport protocol when multicasting. The chapter has given the reader a basic insight into UDP and this is important because in the next chapter we place multicasting under the microscope and evaluate various multicasting schemes which utilise UDP.

Chapter 4

Multicasting Basics

4.1 Introduction

The reader has now been presented with the building blocks of multicast in IP networks: IPv6 carries the packets between nodes and UDP provides the transport layer which delivers data to services. Next we examine how multicast works both in and on top of IP networks, and describe and compare various protocols which implement multicast routing.

In Section 2.2, we defined multicasting as one-to-many communication where a sender can generate a packet and transmit it to a group of recipients. This definition clearly does *not* tie multicast to IPv4 or IPv6 networks exclusively. As we will see, there are numerous mechanisms for enabling multicast, of which IP Multicast[1] is just one.

Multicasting provides many benefits over the traditional Unicast packet delivery mechanism. If one considers Internet radio as an example, two possibilities exist for delivery of the audio stream:

- Each listener connects to the server with a unique Unicast channel.
- Listeners become part of a multicast group.

[1]When the term 'IP Multicast' is used, we are referring to multicast on both IPv4 and IPv6 networks. When there is a need to differentiate, we will make the distinction clear by using either 'IPv4' or 'IPv6' in place of 'IP'.

In the former case, bandwidth consumed it proportional to the number of clients; if b is the bandwidth required for one client to receive audio, the total bandwidth needed is $b \times n$ where n is the number of clients.

When multicasting however, the bandwidth requirements from the server's point-of-view are constant regardless of group size.

A further advantage to multicasting is that network resource discovery becomes much easier. Machines providing network services join well-known multicast groups, and hosts wishing to use those services make contact via the multicast group without knowing the address of the server on which the service is running.

Perhaps now is a good time to introduce the concept of *multicast groups*; when transmitting and receiving multicast packets, it is often necessary to be able to differentiate between multicast data meant for disparate sets of interested listeners, especially when a generic packet delivery framework such as IP Multicast is used. A *multicast group* is a subset of the multicast address space, useful in the sense that when a packet is transmitted to that group's address, only that subset of the total address space will receive the packet. Hosts may join or leave the group at any time, and no restriction is placed on the number of groups a host may join, or on the location of the host [36].

In the ensuing chapter we use the terms 'multicast address' and 'multicast group' interchangeably; both indicate a set of recipients with the property that a particular set can be uniquely identified from other similar sets by means of the unique address assigned to that particular set.

Since this dissertation is not specifically focused on the delivery of multicast packets but at multicast address allocation, we will briefly examine general multicast mechanisms in order for the reader to have a superficial understanding on how packets may be sent to multiple recipients, before explaining how multicast works in IPv6 networks.

We can broadly categorise distribution of multicast packets as follows:

- Router-dependent Multicast
- Application-layer Multicast
- Hybrids of the above two approaches

4.1.1 Router-dependent Multicast

The distinguishing property of Router-dependent Multicast is that *"[it] imposes dependency on routers"* [37]. Multicasting in this manner is reliant on support in the software modules of the router to process and forward packets correctly; without the software being available and functioning correctly, multicasting cannot take place.

The prime example for router-dependent multicast is IPv4 Multicasting [1], which is widely supported by both hosts and routers on the Internet, but often not enabled. In order for routers to utilise internetwork IPv4 Multicasting, they must implement IGMPv3 (Internet Group Management Protocol), which provides both hosts and routers with the ability to *"report their IP multicast group memberships to any neighboring multicast routers"* [38], and support an internetwork multicast routing protocol, such as DVMRP [39], PIM-SM [40] or MOSPF [41].

Included in this category is multicasting inside a LAN which lacks a router or other network-layer device. In the case of Ethernet, when a link-layer device might be in use, generally all the hosts will receive the packet and it is either filtered by the network interface (if supported by the hardware) or passed up to the host's network-layer where an internal routing decision is made to accept the packet or discard it.

We defer a deeper examination of IP Multicast internals to Section 4.2 when we cover multicast on IPv6 networks.

4.1.2 Application-layer Multicast

Application-layer multicast refers to multicast solutions implemented above the network layer, at the application layer. It is also referred to as Application-level multicast [42], End-host multicast [37, 43] or End System multicast [44] because the only systems which need to run the multicast software are those hosts interested in receiving multicast packets. These hosts are generally located at network edges.

The primary advantage of application-layer multicast is that only hosts that wish to participate in a multicast group need install the multicast soft-

ware. Multicast packets are conducted along unicast channels, and intermediate routers do not have to support any multicast services whatsoever. A further extension of this advantage is that no changes need to be made to the network layer module, which, considering the importance of the role played by this layer in ensuring interoperability, should remain as stable and compact as possible.

In Section 4.3 we will mention the functioning three Application-layer multicast protocols: YOID, Narada and the Banana Tree protocol.

4.1.3 Hybrids

The hybrid approach strives to combine both router-dependent and application-layer multicast into one protocol which has the benefits of both. Typically, application-layer multicast is used for interdomain multicasting, while IP multicast is used for intradomain delivery.

Our candidate for discussion is HMTP, which is reviewed in Section 4.4.

4.2 IP Multicasting

When perusing multicast literature, is inevitable that the name Stephen Deering crops up. In 1988 he published a paper [36] describing a method for passing multicast packets between different networks. A year later he solidified his ideas by releasing RFC 1112, 'Host Extensions for IP Multicasting' [1], which became the standard for IP based multicast. He is also an author of RFC 2710, 'Multicast Listener Discovery (MLD) for IPv6', which introduces multicast group management to IPv6[2] [17].

Unfortunately, IPv4 Internet-wide multicast has proved to be a difficult target to achieve, for a number of reasons. Many papers and projects make the basic presumption that IP Multicast does not have widespread deployment, and seek ways to overcome this limitation with extensions above the network layer. Later we look closely at the drawbacks which currently face IP Multicast.

[2]Deering also penned RFC 2460, which is the IPv6 specification.

What follows is an in-depth examination of multicast on IPv6. We divide the discussion into three parts, Multicast Listener Discovery, multicast packet handling and internetwork multicast routing.

4.2.1 Multicast Listener Discovery

The reader might recall from Section 2.2 that a primary drawback of the IPv4 broadcast was that it caused unnecessary processing in the IP stacks of hosts. With the introduction of IP Multicast, this processing burden has been pushed downwards to the hardware level, and inwards towards routers and other non-edge devices. Routers examine IP packets and if the packets form part of a multicast traffic flow, the packets are only sent out on links if hosts who wish to receive packets for that multicast group exist on a link.

This strategy clearly forces routers to keep track of what multicast groups the attached listeners are interested in, and Multicast Listener Discovery (MLD) provides the functionality. The current version of MLD is version 2, or MLDv2, and it is standardised in RFC 3810 [45]. In the upcoming discussion, the term 'hosts' refers to nodes on the network which are not part of any multicast group, and 'multicast listeners' or 'listeners' refer to nodes which have somehow registered to receive multicast packets.

MLD defines different behaviours for routers and multicast listeners. If a router is part of a multicast group, then it will act as both router and multicast listener.

Router Behaviour

For each link attached to a router, a list of records per multicast address is kept. A record is the 4-tuple

(Multicast Address, Filter Mode, Filter Timer, Source List)

This is significantly different from MLDv1, where a basic list of subscribed groups per link was stored, as MLDv2 has been extended to handle Source Specific Multicast (SSM) [46] as well as Source Filtered Multicast (SFM) [45]. Previously when a listener registered for multicast group G with a router,

the router forwarded packets destined for G regardless of the source. This is termed Any Source Multicast, or ASM, and is the model for multicast service which complies with RFC 1112 [1].

SSM allows a host to inform its router which 'channels' the host wants to become a listener for. Channels are formed by pairing the address of the target multicast group G, with S, the source of the packets which the listener wants to accept packets from. Together they constitute the 2-tuple (S,G) which is stored on the listener and the router. Multicast packets destined for G are only forwarded by the router if the source of the packets matches S. The use of SSM is not forced on application developers, who are free to decide which model best suits their needs.

A further extension to ASM is Source Filtered Multicast (SFM). SFM provides the capability to filter based on the source address of a multicast packet, it is different from SSM because no single channels are subscribed to, rather normal multicast traffic is filtered. The SFM-capable host provides the router with a list L of acceptable sources for group G, as well as a filter directive to either

(i) only forward packets destined for G where the source address exists in L

(ii) or to forward packets destined for G from all source addresses *except* those in L.

In the interests of consistency, listeners also keep track of what filter directives are set for the groups they are listening to. These directives can be changed at any time by the listener.

MLDv2 supports all the multicast models given here, namely ASM, SSM and SFM.

It is important to note that routers running the MLDv2 protocol do not need to know the identities of the individual listeners for G on a given link, nor is it required to keep count of the listeners. The reason for this is that if a router receives notification of a listener on link L for G, then it is already has enough information to distribute packets destined for G to all other listeners of G on L.

Returning to the 4-tuple listed earlier, we can now make some sense of the fields. Multicast Address is an IPv6 multicast address which a listener is interested in. Filter Mode is either INCLUDE or EXCLUDE, and corresponds to one of the two filtering directives for SFM shown above. Filter Timer is a countdown timer and Source List is a list of zero or more IPv6 Unicast addresses. These Unicast addresses are used in conjunction with Filter Mode to provide SFM.

Routers must configure every interface MLD is running on to accept all IPv6 Multicast packets on that interface, using some link-layer multicast address specific to the network technology. The RFC puts forth the example of IPv6 multicast over Ethernet[47]: each interface on the router is configured to accept packets whose Ethernet address starts with `0x3333`, which under Ethernet indicates a multicast packet. In this fashion the router receives all packets originating from its links destined to any multicast address.

At varying time intervals, routers send out different Query messages on attached links in order to discover information about which groups or channels listeners are interested in. Each listener replies with Report messages informing the router of the listener's state. State refers to the list of subscribed multicast groups, as well as associated ancillary data such as source filter lists and filter directives. State information is held in a state store.

Query messages have a hop-limit of 1 in the IP header, which ensures the queries never cross subnet boundaries.

If there are multiple routers on a single subnet, an election takes place and the winner becomes the Querier; only the Querier may issue Query messages. The other routers stay silent but may still process Reports and multicast packets, and if the Querier fails an election takes place and the new Querier can step into the breach seamlessly (more details are available in the RFC.)

There are three types of Queries a Querier may issue:

General Queries These messages have destination address `ff02::1`, and discover which groups the attached listeners are interested in.

Multicast Address Specific Queries Sent to a particular multicast ad-

dress to discover if there are listeners for a particular group.

Multicast Address and Source Specific Queries Sent to a particular multicast address to discover if any listeners exist for a particular channel. The message carries a list a sources which the Querier knows some listeners are interested in packets receiving from.

Further, there are two type of Reports which a listener may respond with:

Current State Reports Contains information about the state of multicast addresses currently being listened to, triggered by a General Query.

State Change Reports Different State Change Reports exist, depending on the nature of the change. These Reports are triggered when the state of a listener is changed.

General Query messages are generated periodically by the router and transmitted on each link L where the router is the elected Querier. When a **General Query** is received by a node, the listener creates a **Current State Report** message, which contains a record of each group G the listener is listening to on link L, as well as any associated Source list.

At unpredictable intervals the Querier may receive **State Change Reports**, indicating that a listener has altered the state of an address record. This could be a listener ceasing to be interested in a group, or changing its filter directive for a group. The router then issues either a **Multicast Address Specific Query** or a **Multicast Address and Source Specific Query**, dependent on the nature of the change reported by the listener, and the listeners respond with **Current State Reports**. This forces a coherent view of the network to be maintained, within certain timing limits.

(In this distillation we have ignored a number of details, including timing issues. The interested reader is referred to the RFC).

Listener Behaviour

The above describes MLD from the routers point-of-view: routers can issue three kinds of Queries, and expect Reports from interested listeners. What remains is MLD from a listener's perspective; the specific Reports which

listeners issue in response to Queries and the Reports listeners issue in response to internal events. Events may be fired by countdown timers reaching threshold values, or state in the listener changing.

When a **Multicast Address Specific Query** is received by a node on link L querying listeners of group G, a **Current State Report** containing the listener state for G is generated if the node is currently a listener for G on L and sent to the address listed in the query. The router, which is listening for all multicast packets, picks up the response and adds the information to its state store.

Similarly, when a **Multicast Address and Source Specific Query** is received on L, with the list of addresses S, then the listener responds with a **Current State Report** containing listener state for G when either of the following are true:

(i) The INCLUDE directive is set, and S contains at least one source address the listener is prepared to accept packets from.

(ii) The EXCLUDE directive is set, and S is non-null when all the addresses in the listener's exclude list for G on L are removed.

When a host becomes a listener on a particular link, it immediately sends an unsolicited Report message for the group the host has joined. This ensures that, if the host is the first listener on that link for that group, the relevant router is notified. If there is already a listener on that link for that address, the Report is spurious but causes no harm.

Two methods for group leaving have been are available: 'soft leave' and 'fast leave'.

Both methods make use of timers and the Report messages previously discussed, but briefly: the soft leave mechanism involves no **State Change Report** being issued, and no **Current State Reports** indicating any listeners on the link forces routers to remove state information for that group, while the fast leave method requires a listener to inform all the routers of the cancellation of the listening socket with a **State Change Report**. The Querier then issues a **Multicast Address and Source Specific Query** for the address which is in the process of being unsubscribed. If no positive

Reports are received within a time-frame, all the routers remove the group from their respective state stores.

One of the important differences between MLDv1 and MLDv2 is that MLDv1 required listeners to suppress Reports; if the Querier issued a Query then listeners had to issue a Report after a random period of time, with the condition that if a Report from another listener for the same Query arrived before the listener's Report had been sent, then the Report was to be suppressed. The rationale behind this decision was that routers only needed to know about one listener on a link in order to send the message to all other listeners on that link, and that any reduction in the number of packets sent was viewed as an efficiency gain. However, this approach had a number of drawbacks (see [45], so this restriction has been dropped in MLDv2. The new requirement that every listener responds to Queries to which the listener has Reports for produces these benefits [45]:

- Routers can track group size.
- Avoids implementation issues on bridged LANS.
- Complexities in the state machine for nodes have been removed.
- More than one address can now be carried by MLDv2 Reports.

MLDv2 packets

MLDv2 is a subset of the ICMPv6, the Internet Control Message Protocol for IPv6. The various Queries and Reports which make up the MLDv2 standard are contained in ICMPv6 packets, which in turn are carried by IPv6. Remember from Chapter 2 that IPv6 only provides a 'best effort' delivery service; no guarantees are in place to ensure packets reach their destination intact and in order, unlike the TCP which adds reliability to the IP layer. The designers of MLDv2 therefore built-in mandatory retransmission of important packets into the specification.

The format of the ICMPv6 packet has been drawn in Figure 4.1. At first glance it appears highly simplistic, and this is true of base ICMPv6. However one must consider it is merely a carrier and does not offer much in the way of functionality, as a piece of copper wire is useful in conducting

electro-magnetic signals. The strength of ICMPv6 is in the type of messages transmitted. To extend the copper wire analogy, it is pointed out that basic copper wire can be used for such disparate applications as turning electric motors or passing data between computers. MLD messages are just one application of ICMPv6, the interested reader is directed to RFC 2463 [48] for more detail.

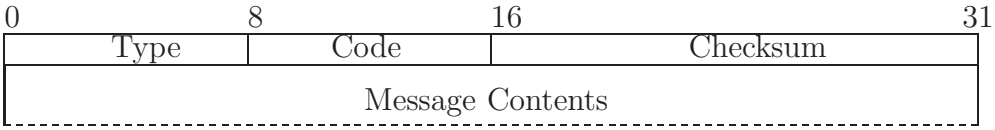


Figure 4.1: ICMPv6 Packet Header

A breakdown of the fields in an ICMPv6 packet:

Type Indicates what ICMPv6 message the packet is carrying. For MLDv2 Queries, the value is 130 decimal and for MLDv2 Reports the value is 143 decimal.

Code Ignored by MLDv2, set to zero.

Message Contents Application specific data, presumably has some meaning in the context of legitimate Type and Code values. When carrying MLDv2 messages, this field holds the data of the message.

4.2.2 Multicast packet handling

In previous chapters both IPv6 and UDP were presented, but we have yet to demonstrate how one can send and receive multicast packets. From a practical point of view, these functionalities are provided by software calls. Depending on the operating system, they may be library or system calls. This is covered in more detail in Appendix B, but a superficial understanding at this stage should help crystallise multicast programmatic concepts for the reader.

The calling semantics presented here are applicable to version 2.6 of the Linux kernel.

A running program that wishes to become a listener on a multicast group first creates a BSD-style socket with `socket(2)`. The newly created socket is then bound to a specific address with `bind(2)`. Finally, the program calls `setsockopt(2)` and passes the value `MCAST_JOIN_GROUP` to the socket. This causes the underlying multicast module to issue a **State Change Report** and create the state for the group. The listener is then able to receive multicast packets by iteratively using `recvfrom(2)` which returns the data from received multicast packets.

Because MLDv2 requires the state record for a group to have a Source list and Filter mode, when a host issues a join request for an ASM group the Source list is set to $\{\}$ (the empty set) and the Filter mode is set to EXCLUDE. All packets addressed to the group will be forwarded by the routers involved.

Extending this, when a host wishes to join a SSM group the Source list is set to the address of the acceptable source A, $\{A\}$, and the filter mode is set to INCLUDE. Now only those packets with source $\{A\}$ will be forwarded.

Finally, if a host wishes to make use of SFM, it can manipulate the Source list and Filter mode to achieve the desired effect.

A listener will use the `setsockopt(2)` call to set the Source list as well as the relevant Filter Directives.

If a listener wishes to leave a group, then `setsockopt(2)` is called and passed the value `MCAST_LEAVE_GROUP`. This instructs the multicast module to remove the group state, and issue a **State Change Report**.

When an application wishes to send data to a multicast address, the sender uses `socket(2)` to create a BSD-style socket, and then calls `sendto(2)` with the multicast address and data to be sent as arguments. The data is passed to the kernel where it is packed and sent on its way.

4.2.3 Multicast Routing

At this point the reader should be comfortable with the notion of multicast groups and the management thereof, as well as the steps an application needs to perform in order to participate in a multicast group in the role of sender or listener. Our explanation has covered multicasting on local networks, but

what of multicasting across subnets or domains? We have only mentioned routers in passing, and our coverage of the routers involved has treated them as black boxes: they somehow provide packets addressed to locally joined groups and they whisk away packets sent from local nodes to the group and distribute the packets to all other listeners. In this subsection we examine the evolution of multicast routing, as its packet delivery capabilities expanded from LANs to the Internet.

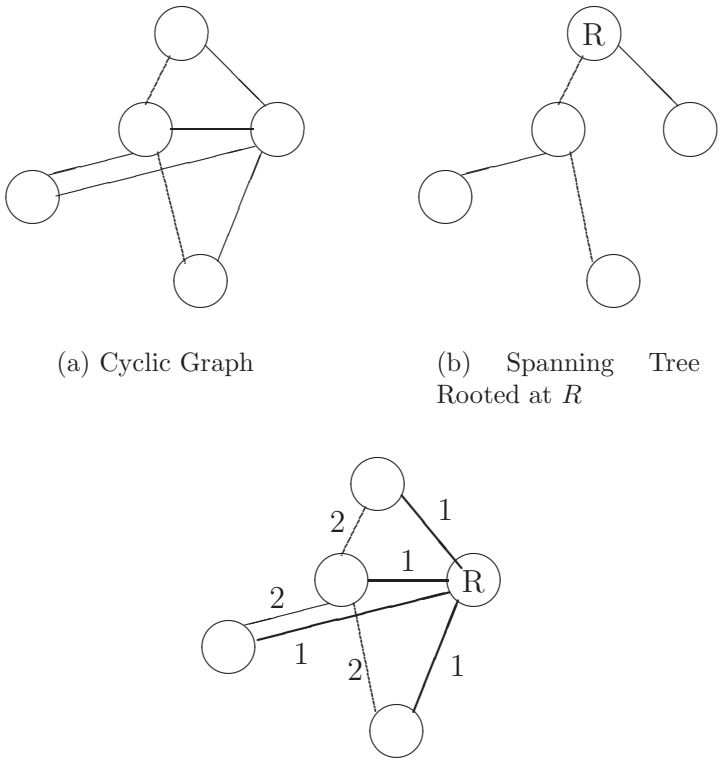
Before we start, an informal refresher on graph theory is in order, as the routing protocols make extensive use of this topic. A graph is simply a connected set of objects. The objects are referred to as *vertices* (sometimes the term *node* is used in computer network theory, the plural *vertexes* may also be found in the literature) and the connections between the vertices are termed *edges*. A *cyclic* graph is one where a path can be traced from a vertex across at least two edges and arrive back at the starting vertex without traversing any edge more than once. Conversely, an *acyclic* graph contains no cycles. Figure 4.2(a) is an example of a cyclic graph.

A connected graph has a *path* between every vertex pair, where a path is a list of vertices where each adjacent vertex pair in the list is connected by an edge. Each of the graphs in Figure 4.2 is connected.

A *spanning tree* is a "*connected, acyclic subgraph containing all the vertexes of a graph*" [49]. From this definition it should be clear that a graph may have more than one spanning tree. The *root* is a vertex which has been arbitrarily designated, and forms the starting point when traversing the tree. A *parent* node is one which is connected to at least one other node away from the root. A *child* node is any node which is not the root. One of the spanning trees for Figure 4.2(a) is given in Figure 4.2(b), with the root indicated by **R**.

The *reverse-path* in a tree can be traced by starting at a child node and recording the path to the root node.

Suppose a weight is assigned to each edge in the graph. The *minimum spanning tree* or *shortest path tree* is the single spanning tree in a graph with the least sum along the edges. We have assigned weights to the graph from Figure 4.2(a) as shown in Figure 4.2(c), and indicated the minimum spanning



(a) Cyclic Graph

(b) Spanning Tree Rooted at R

(c) Minimum Spanning Tree Rooted at R

Figure 4.2: Graph Types

tree in bold lines. Again, the root node is denoted by **R**.

In Deering's introduction to multicasting [36], he describes the state of the art with regards to multicast routing. At that stage the field included Single Spanning Tree, Distance Vector and the then brand new Link-State routing protocols. Following on from that early work, CBT, PIM-SP, PIM-DM and BGMP have been developed. What follows is a short description of each of these models for multicast packet routing.

Single Spanning Trees

Before router-use became widespread, most sites connected LANs by means of bridges. At the link-layer, bridges need to compute a single spanning tree

of all bridges in the network to ensure that packets are not duplicated across the network, and this spanning tree is ideal for use with multicast. Each bridge needs to keep track of what groups the machines connected to that bridge are members of, and the solution proposed was for group members to periodically issue membership report packets with the source address set to the group address. Thus the bridge could learn on which links listeners for a given group were present.

When a packet addressed to a group was received by the bridge, it would send it out on links where the bridge believed listeners were present.

Banikazemi's [50] description of the IEE-802 MAC bridge implementation for spanning tree multicasting shows that it requires less features, *"Whenever a router receives a multicast packet, it forwards the packet on all the links which belong to the spanning tree except the one the packet arrived on"*. The only state a bridge has to store is whether a given link is part of the spanning tree.

The disadvantages to this scheme are twofold:

1. Multicast traffic is sent on links regardless of whether a listener is present or not.
2. A single set of links are used (the spanning tree), introducing possible bandwidth contention issues

Further, this scheme in non-scalable in the extreme since, by design, it is limited to bridges. However knowledge gained on spanning tree use was later applied when designing internetwork multicast routing protocols.

Distance Vector

Distance Vector routing protocols have been with us for a long time now. Also referred to as Bellman-Ford routing algorithms, Deering notes that the first routing on the Arpanet was accomplished with a distance vector protocol [36]. In terms of multicast routing, distance vector protocols were established to route multicast packets across networks (but *not* autonomous systems).

Routers store a table containing all reachable destinations (destinations can be hosts or networks), as well as distance to and the next-hop of the

destination. These records are regularly sent to neighbouring routers who update their own stores if better routes are advertised and in turn send out their own records. (This description suffices for our purposes, but the curious reader is advised to consult other sources if distance vector routing is new territory. An introduction may be found on page 628 of Forouzan's "*Data Communications and Networking*" [51].)

We saw that single spanning trees have disadvantages, and one possible solution is to compute multiple minimum spanning trees, each rooted at a different vertex of the tree. However this is not viable with distance vector protocols as a minimum spanning tree would have to be computed for each group at each source. Instead the notion of a *broadcast* tree for each source S is defined: all possible minimum spanning trees for S are subtrees of the broadcast tree. A packet sent on the broadcast tree reaches every node in the tree, and the broadcast tree used in a number of distance vector algorithms, including **Reverse Path Flooding (RPF)**, **Reverse Path Broadcasting (RPB)**, **Truncated Reverse Path Broadcasting (TRPB)** and **Reverse Path Multicasting (RPM)** [50].

In RPF, the router forwards a multicast packet on all links except that link on which the packet arrived, provided the packet arrived on the link which is the shortest away from the source. The disadvantage of RPF is that packets may traverse a link more than once since, if more than one router is present on a link L, each router on L will send the out the multicast packet to L.

This weakness was addressed in RPB, where each router determines which of its links are child links in the minimum reverse-path to a source node. Once this information is calculated, the router only forwards a packet from a source outwards on a link if that link is a child link for that source. The information needed to compute the minimum reverse-path can be found from the unicast distance vector routing protocol. The failing of RPB is that, while it eliminates duplicate packets, unwanted packets are still transmitted on networks where no listeners are present.

TRPB is a routing protocol which requires listeners to inform routers of their desire to receive traffic for particular groups. A *leaf link* is defined to

be a link on which no routers are present, only listeners. If a router with a leaf link L discovers that no listeners for group G are present on that link (via some group membership reporting protocol, MLDv2 mentioned earlier is an example), then it can drop outgoing packets on L addressed to G.

Distance Vector Multicast Routing Protocol (DVMRP [39]) was the first standardised internetwork multicast routing protocol. It is based on TRPB and is defined as an *"interior gateway protocol"*, that is it *cannot* connect different autonomous systems together. The MBONE, which is now defunct, was the first internetwide multicast network, and used a modified version of DVMRP [52].

An extension to TRPB is RPM, where routers can prune not only leaf nodes, but branches in the tree as well. This is accomplished when a router R determines that no listeners on leaf links exist for a group G, and that no routers in the shortest reverse-path on child links exist. The router R then sends a prune message to its parent containing the destination group address, which forces the parent not to forward packets for G to R. A pruned branch has a specific lifetime; once the lifetime expires packets are sent once again throughout the network, and routers need to issue new prune messages.

Link-State

Link-state routing protocols are newer than distance vector protocols, and work slightly differently. Instead of routers sharing information about the network amongst each other, they share the information about the state of their links [33]. This value can be a combination of many factors, including *"security levels, traffic, or the state of the link"* [51]. This is unlike distance vector protocols, where the only metric stored is hop count.

Whenever the state of a link on a router changes, the new state is flooded throughout the network to all other routers. This allows each router to store the complete topology of the network, and each router can then calculate a minimum spanning tree rooted at itself using Dijkstra's algorithm [51, p. 637].

In applying link-state routing to multicasting, it is again required for

routers to determine whether listener for a group are present. When a packet from source S destined for group G first arrives, the router calculates a minimum spanning tree for the tree rooted at S. Once completed, the router knows which links to send the packet out on, and does so provided listeners are present for G. The minimum spanning tree is cached to facilitate faster processing.

When the state of a link is altered, the changes are flooded across the network as described above, and cached entries are removed from all routers.

Drawbacks for link-state routing include the memory requirements on routers, flooding of state packets throughout the network and the need to calculate minimum spanning trees which can be complex operations.

Multicast Extensions to Open Shortest Path First (MOSPF) implements multicast routing in a link state protocol, OSPF. Similarly to DVMRP, MOSPF is an interior gateway protocol and requires listeners to report group membership. A single router is elected the Designated Router (DR), and only the DR queries listeners about group membership and sends out link-state advertisements. Because of MOSPF's basis in OSPF, the protocol responds very quickly to changes in topology, which can be needful depending on the importance of the packets.

Core Based Trees

In an attempt to eliminate the calculation of a spanning tree per potential group source, Ballardie et al. proposed Core Based Trees [53] in 1993. The essential idea when using Core Based Trees is that only *one* spanning tree is computed for the entire group, called the *shared tree*. Packets may enter the shared tree at any point, and will be distributed to all other nodes.

Figure 4.3 is reproduced from the paper referred to above and shows the path of a packet sent to a Core Based Tree. The drawing illustrates how using a single core creates a possible weakness; if for some reason the core fails then distribution on the tree becomes impossible and links may become congested. The tree will also very probably not be a shortest-path tree for all sources. The advantage to CBT is that it scales linearly with the number of

routers which join the tree [53] and has reduced multicast routing state [54].

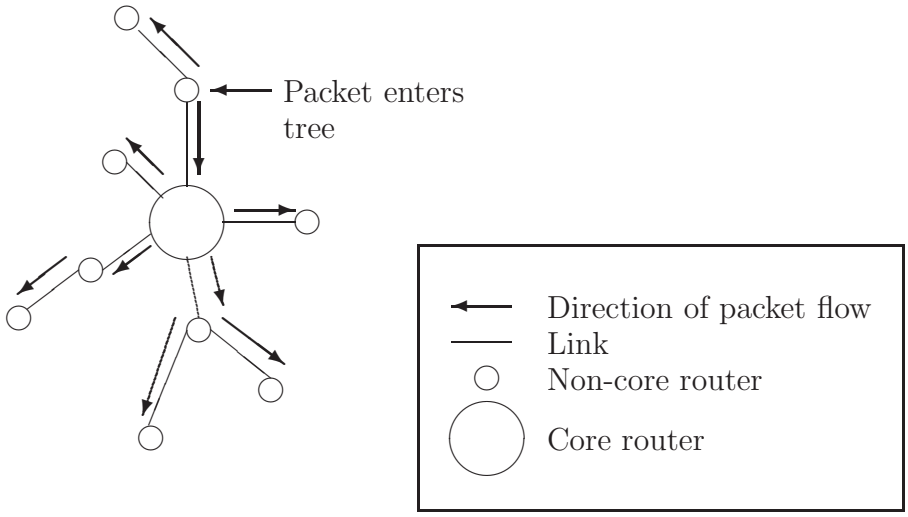


Figure 4.3: Core Based Tree

PIM-SP

Traditional distance vector and link-state multicast routing protocols suffer from a crippling problem: scalability [54]. Both methods at some stage require packets to be sent throughout the network, for example with DVMRP when a source first transmits to a group, the packet is forwarded by all routers and only then are prune messages triggered. While this is fine within an autonomous system, it cannot possibly scale Internet-wide. Further, packets addressed to group G are sent on links which might have no interest in the G.

A new architecture was required for routing multicast packets across autonomous systems, and the first effort was Protocol Independent Multicast - Sparse Mode (PIM-SP)[3]. The designers of the PIM-SP model recognised that shortest-path trees and shared trees have different strengths, and allow a group to chose whether to use a shortest-path or shared tree.

[3]The addition of Sparse Mode indicates that another mode is available, and this is true. PIM-Dense Mode exists but is very similar to DVMRP, and so will not be discussed at all.

While PIM-SP is dependent on an underlying unicast routing protocol, it is not associated with any particular unicast routing protocol. Sparse mode indicates that the receivers are scattered around the Internet, dense mode then being the situation when receivers are close to each other.

PIM-SP is different from the previous multicast routing schemes we have examined in two key areas[54]:

1. In dense mode multicast routing (like DVMRP, for instance), initial packets reach all routers with leaf links before pruning takes place. In PIM-SP, routers need to explicitly join a tree.
2. Senders and receivers need to meet at a nominated *Rendezvous Point* (RP), instead of tree construction taking place from data flooded on the network.

A host wishing to join group G, issues a request which is picked up by a Designated Router, which performs the same group management functions as the DR in MOSPF. A separate mechanism is used to determine the RP for G (this could be manually configured, for instance, or a host-driven protocol). The DR then sends a PIM-Join request out on an interface towards the RP. Each upstream router receives the join message and issues a new PIM-Join towards the RP. Once the RP receives the join message, a note is made of the link L on which the join request arrived, and period of keep-alive messages are transmitted on L to reassure down-stream routers that the RP is still active.

At the start of a transmission from source S to group G, the DR for S encapsulates the packet in a Register message and sends the message to the RP via unicast. The RP then unwraps the message and injects it into the shared tree. The packet will then be delivered to all members.

If an application needs to take advantage of shortest-path tree characteristics, eg. bandwidth requirements are such that a shared tree is insufficient, it may request the RP to form a shortest-path tree. Once the shortest-path tree has been formed, routers will stop accepting packets for G from the RP, and only process packets which arrive from S.

Much detail concerning switching from the shared to the shortest-path

tree has been omitted, the entire protocol is explained in [54] and published for comment in [40].

BGMP

The Border Gateway Multicast Protocol is an extension to the well known Border Gateway Protocol [55], and BGMP provides support for interdomain routing of multicast traffic. Under IPv4, BGMP was bundled together with Multicast Address-Set Claim (MASC), which supplied domains with blocks of multicast addresses for use inside their domains. Because of the hierarchical assignment of IPv6 addresses, MASC has been abandoned in favour of multicast addresses which are based on the network prefix.

Border routers construct a shared tree for each group with each other between their domains, and run either DVMRP or PIM-DM on their interior interfaces.

4.2.4 IP Multicast Drawbacks

Application-layer multicast has received much attention in recent years, due to the perceived slow rate of IP Multicast deployment. In fact, as we alluded to earlier, a number of papers seem to take this as a truism [37, 43, 56] or base this on now out-dated information [44]. In any case, anecdotal evidence suggests that since many modern routers support IPv4 Multicast, the problem appears to be either system administrators who have no desire or necessary skill to implement IP Multicast, or a lack of demand from customers.

If one looks beyond the 'lack of deployment' argument (which is tenuous at best, given the state of current internetworking equipment), there are certainly legitimate reasons to question the effectiveness of current IPv4 Multicast. Diot and his colleagues [57] make several important points, which we introduce and discuss in the next four paragraphs.

"A long-term problem for multicast deployment is that it upsets the router migration model *that ISPs follow."* In this model, new routers are installed on the network's core, while the replaced routers are shifted towards the edges of the network. In this way the network is slowly upgraded in a step-wise

fashion and newer functionality is propagated from the core outwards. The problem with this model is that generally IP Multicast is not supported by older routers. If vendors do not supply patches to support multicast, then the routers have to be replaced before they have properly depreciated. Without this disruption, a situation may arise where full multicast is available in the network core, but the edges lack any multicast support.

"ISPs using ... core-based protocols face a number of problems regarding domain independence." Many multicast protocols (not just IP Multicast) use the notion of a rendezvous point (RP) to facilitate group management. When an RP is situated outside of an ISP's administrative scope, the ISP is not able to influence the functioning and configuration of the RP in order to improve delivery. When an RP is situated within the ISP, the ISP might not want to waste resources for a service which none of their customers utilise.

"The current service model does not consider group management." There are a number of issues to be worked on:

- No support for receiver authorisation. Anyone can join a multicast group and listen in, there are no access controls.
- A sender authorisation protocol is still in draft form [46].
- Address assignment is not integrated, so two disparate applications transmitting to the same group will confuse listeners.
- It is possible for an attacker to send large volumes of unwanted data to the group, causing network congestion and loss of packets.

"Providing security for multicast-based communication is inherently more complicated than for unicast-based communication." Specifically, encryption combined with multicast requires complex handling of the keys used, and positioning of access control mechanisms.

The introduction of IPv6 is a double-edged sword. On the one hand multicast is an integral part of the standard; nodes which support IPv6 must include multicasting support and this should make multicast application development and deployment many times easier. The nuisance as far as vendors are concerned are twofold: new IPv6 software has to be developed for network

devices and many standards are not yet concrete[4]. In many cases the protocols are dissimilar from their IPv4 counterparts, and so a simple substitution of address formats is not sufficient.

Returning to the problems with the current address model mentioned recently, we add the address allocation problem to that list. In IPv4 multicast addresses were assigned the range `224/8 - 239/8`. MASC [55] allowed for dynamic allocation of this space, but has not become widely deployed.

On the MBONE, applications assigned addresses to themselves in a semi-random fashion. Once assigned, the address was advertised in a session directory (obviously this was only available to those running MBONE specific applications). The directory contained the address of the group which a session would use, and thus also performed the function of address allocation. If two application advertised the same address, a conflict would be detected and resolved.

Handley [58] points out that a fundamental issue arises with this kind of address allocation and it involves the scope or time-to-live (TTL) of the address requester. The TTL field in an IPv4 packet limits the number of hops a packet would be forwarded over. To illustrate the problem, assume address requester $R2$, with TTL 10, generates and advertises A in the session directory. If another requester $R1$ is more than 10 hops away from $R2$ (say the distance is 15 hops), it will not see the session advertisement. In a symmetric network where all TTL values are the same, this does not constitute a problem. However in the real world different TTL values are used in disparate networks. If $R1$ generates A, and has a TTL of more than 15, the advertisement will cause a clash in the session directory.

He also advocates splitting the session directory from the address allocation mechanism, which is what the author has done in Chapter 6

A more general examination of the address allocation problem can be found in the paper by Zappala et al. [5].

The IPv6 multicast address allocation problem has not been fully studied at this point. The address allocation problem is the topic of this dissertation,

[4] An example of this is the deprecation of the site-local unicast address, which occurred during the writing of this dissertation.

and an analysis of the problem is provided in Chapter 5, which is followed by a proposed solution.

4.3 Application-layer multicast

There is somewhat of a naming quandary for this model of service delivery; the various names have already been mentioned in Section 4.1.2 and we feel application-layer is most descriptive, so this is the convention used in the remaining part of this chapter.

At the end of the previous section a number of concerns which have dogged IP Multicasting were noted, and the open nature issues lead many people to think about alternatives (a mere smattering of the available application-layer multicast protocols may be found in [37, 42, 43, 44, 56, 59]). Key was the fact that support for multicast exists in most desktop operating systems, but intermediate routers are not configured for or do not support IP Multicast. Most of the papers cited acknowledge that scalability is an issue with application-layer multicast, but believe the benefits outweigh this.

The obvious solution was to move the multicast logic in the network layer higher, and implement it at the application layer. The benefit of this approach is that no changes are required to the network infrastructure [56]. Three application-layer multicast protocols are explained here, to provide the reader with a counterpoint to the IP Multicast protocols.

All application-layer multicast protocols overlay their topologies on an existing unicast network, which in most cases is IP. The topology of the overlay is not necessarily a tree, in the case of Narada [44], a mesh is overlayed on the underlying physical network. Only then are minimum spanning trees computed for each source. The advantages of this approach are: *"group management functions are abstracted out, and [not] replicated across multiple (per source) trees"*, mesh repair is easier because maintaining an acyclic graph is not a constraint and standard routing algorithms may be employed to calculate the delivery tree.

An example of a protocol which follows a tree-first strategy is YOID. A *Rendezvous Point* (RP), similar in function to the RP of PIM is used. A

URI-like notation for encoding the address of the RP is used, eg. `yoid://foo.bar.org/wb:55555`[59]. The RP does *not* forward data unless it is a member of the group, its only function is to keep track of current group members. The first member to join the group contacts the RP and becomes the root for the group. Following an initial join, other members who join the group obtain a list of current members of the group. One of the current members is chosen by the joining member to be its parent, and a connection is made to the parent. The parent can refuse the request, in which case the joining member selects a new parent. Data is disseminated by each root, who sends packet to its children, who in turn distribute to their children. This recursively repeats until the packets have reached all leaf nodes.

Both Narada and YOID make provision for refinement of the overlay network, the difference is in the pace at which these improvements occur as well as metrics used when creating the trees.

The Banana Tree Protocol [43] (BTP) is our final example of application-layer multicast. It is much simpler than the two other protocols presented so far, and makes use of a tree topology. It does not perform well under real world circumstances as the refinement operations allowed on the tree are limited. It is included so as to demonstrate that complex solutions are at this point required to solve the application-layer multicast problem.

4.4 Hybrids

The final service delivery model for multicast data are hybrids between IP Multicast and Application-layer Multicast. They are a synthesis between the speed of IP Multicast and the network independence of Application-layer Multicast. Interdomain packets are routed between multicast 'islands' by application-layer multicast and once the packets enter these islands, IP Multicast takes over and delivers the packet to group members. The example protocol here is HMTP, the Host Multicast Tree Protocol [37].

Again, a RP is employed to keep track of current groups. In each island one host is elected as the Designated Member (DM) whose purpose it is to

establish connections with a current member of the group, obtained from the RP. This forms a single shared tree. Wrapped packets are passed between DMs via tunnels and when a DM receives a packet from another DM the packet is unwrapped and placed on the local island using IP Multicast. When a DM receives a packet from its local island, the packet is wrapped and sent out to all neighbours.

HMTP provides a workable, but limited-scale, solution to multicasting on the Internet. Figure 4.4 shows what a hybrid IP Multicast/Application-layer network might look like [37], the lines connecting the DMs are the tunnels.

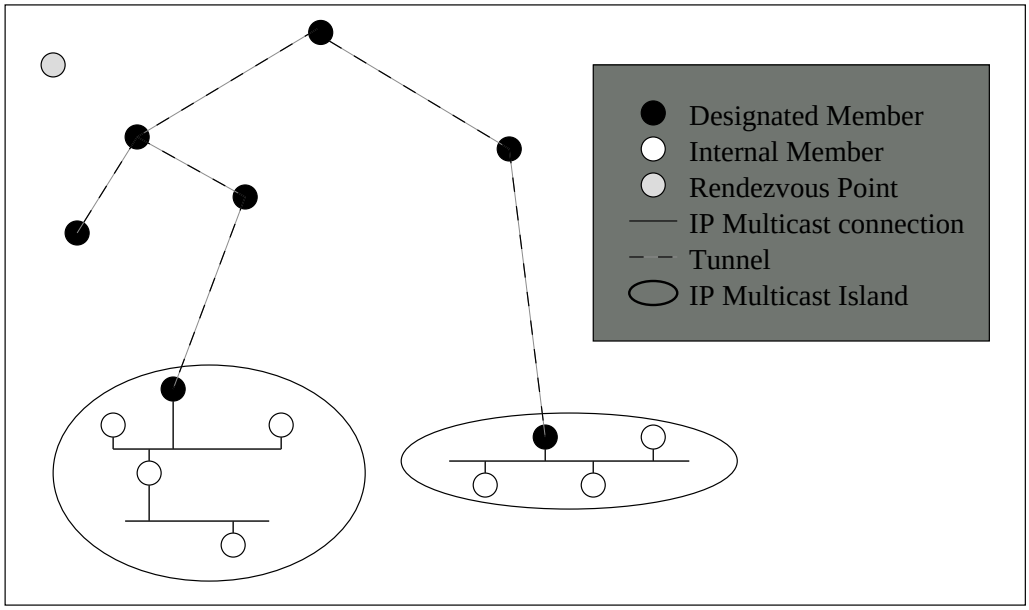


Figure 4.4: Hybrid Multicast

4.5 Conclusion

This chapter has introduced the reader to the very broad world of multicasting. It started by defining the three multicasting categories, namely IP Multicast, Application-layer Multicast and Hybrids.

IP Multicast was then covered in some depth, examining the mechanisms for group management, briefly touched on programmatic semantics and discussed the many IP Multicast routing protocols. We also pointed out the shortcomings of current multicast models.

The chapter finished off with a look at Application-layer protocols and Hybrids, and discovered that, though they have their uses, they suffer from scalability issues which will make their widespread deployment unlikely.

Now that the reader has an understanding of how packets are multicast on IP networks, let us move onto Chapter 5 where the address allocation problem is discussed, analysed and the solution requirements enumerated.

Chapter 5

The Multicast Address Allocation Problem

5.1 Introduction

This chapter serves to elucidate the problem of multicast address allocation (or the *malloc problem*). A little more background is provided on the issue, followed by an analysis on aspects of the problem and from that a set of requirements is distilled that the author believes a proposed solution should fulfill. This work is preparation for Chapter 6, where the author's solution is presented.

5.2 Address 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 be transmitted, and addresses enable the data to be received by the correct party; who can in turn verify the source of the data. Generally links are a hardware feature, are static in nature and beyond the scope of this dissertation; so they are not considered further. Addresses on the other hand can be volatile; they are often altered, both manually and automatically. To verify that this is true, South Africans need only look at

how local ADSL is implemented: the link is always on and constant, but the subscriber's IP address changes every 24 hours [60]. This address change occurs automatically, and the assigned addresses are thus termed *dynamic addresses*. *Static addresses* are those which need to be manually altered.

Generally machines which provide services have static addresses; this allows potential clients to have a high degree of confidence in the fact that the correct machine is reachable when seeking a particular service. On the other hand, applications should be able to request unique multicast addresses from the operating system as and when such addresses are needed. It is illogical to force a machine or site administer to manually configure every application on her machine or network which needs to use a multicast address. In this case, a dynamic solution is more appropriate than a static solution, and, as will be demonstrated, no effective dynamic solution exists at the moment.

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, as was seen in Chapter 2.

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 uses a 3-tier address hierarchy as follows [61]:

1. A super-block of addresses is split up amongst MASC [55, 62] (Multicast Address-Set Claim) servers which provide interdomain allocation. MASC provides a protocol for distributing addresses amongst these top level servers.

2. Once a block of addresses is allocated to a domain, it is further split up for intradomain allocation to the Multicast Address Allocation Servers (MAASs [61]).

3. In the last step of the architecture, applications can request addresses from a MAAS with the MADCAP [63] mechanism.

Figure 5.1 illustrates the architecture described above, and is self-explanatory except, perhaps, for the Undefined Links indicated by dotted lines. These undefined links show where no standard protocol has been

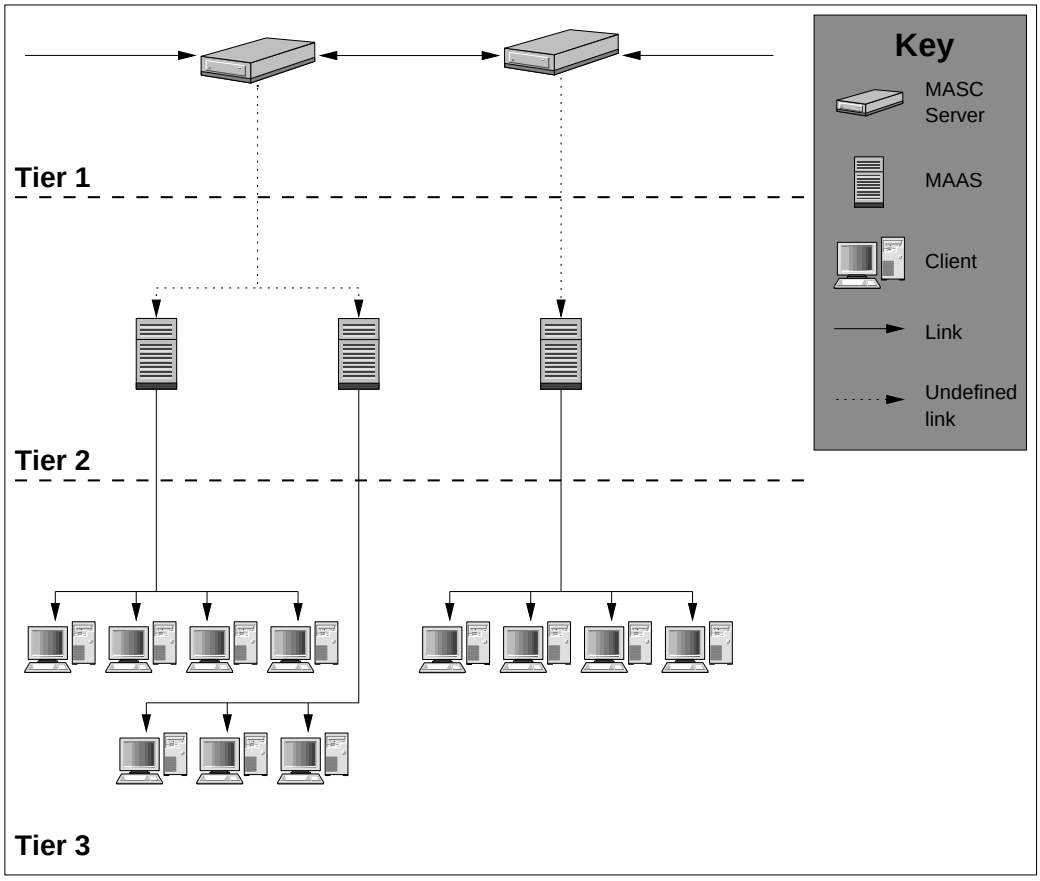


Figure 5.1: Current Address Allocation Architecture

developed to connect objects. In this case, there is no specification for passing addresses from MASC servers to MAASs. The contents of these protocols are beyond the scope of this work and so details are omitted, but they were investigated by the author since they provide a solution to the malloc problem.

Unfortunately the architecture is unworkable for the following reasons:

- Three separate protocols are required for a working architecture.
- Management of the first tier is laborious because address blocks need to be registered and manually injected into the MASC servers, and connections between the servers are hand-configured.

- Currently no protocol exists for allocating addresses from MASC servers to intradomain servers (MAASs). While a draft protocol was developed, it was never adopted as standard [64].
- MADCAP is a cumbersome protocol with eight different message types and a client/server model.

Combining the above negatives, it is difficult to see how the architecture would work successfully. In their description of the state of the malloc problem, Zappala et al. [5] report that *"MASC (and the rest of the hierarchical allocation structure) [have] never been deployed"*. In light of the our rejection of a centralised allocation scheme, we are lead to explore a solution to the malloc problem which uses some kind of distributed and stateless paradigm, where the administrative burden is much lighter.

Separate from stateful/stateless arguments is the notion of a *session directory*, briefly mentioned in the last chapter. 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.

On 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. In Section 4.2.4, we saw how Handley [58] showed that a session directory which combined the functions of session announcement and address allocation has fundamental design issues, and does not scale well. He concluded that these two functions should be split.

Handley's paper was seminal; he made several other important observations which affect how addresses ought to be generated. He noted that purely random based address allocation is insufficient since clashes are expected once the square root of the available addresses have been allocated. He also espoused the idea of an address hierarchy, which would simplify the problem to allocating addresses from a local rather than a global pool of

addresses (this is what the 3-tier address architecture just described tried to achieve), and advocated third party defence. Third party defence occurs when a node prevents allocation of a dynamic address not owned by itself.

In the first comprehensive analysis of the malloc problem, Zappala et al. [5] 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. This fact forms the basis for the author's allocation scheme that will be described later.

5.3 Group Identifiers

In Section 2.4.2 the unicast prefix based address format was presented, and it was explained how the format could be used to generate globally unique multicast addresses. To recap, a unicast network prefix is prepended to a Group Identifier (group ID) and together they form a unique address suitable for global multicast.

Clearly then, the scheme requires nodes in a subnet addressed by a particular prefix to generate group IDs which are unique on that subnet. Before we move on, let us examine some statistical properties of the group IDs which nodes must generate.

The first property measured is the probability of disparate nodes randomly generating identical addresses. Such an event is termed an '*address collision*' or '*clash*'. If nodes choose addresses in a purely random fashion, then Figure 5.2 depicts the probabilities of collisions for varying number of addresses generated up to 100000. The figure includes probabilities for group ID sizes of 32, 40, 48, 56 and 64 bits (note that all points, except those for the 32 bit curve, are very close to the horizontal axis.) Interpreting Figure 5.2, the reader will deduce that only when the group ID is 32 bits in size, will

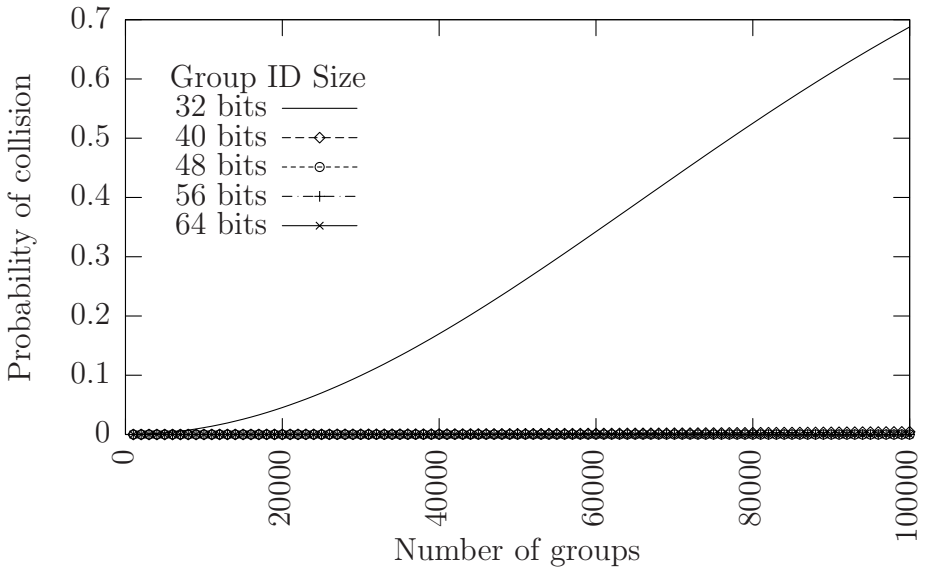


Figure 5.2: Collision Probabilities up to 100,000 addresses

random selection become a limiting factor. However even in the case when the group ID is 32 bits, the situation is not a grave as one might imagine for the following reason. A group ID of 32 bits in size implies that the network prefix is 64 bits in size (see Figure 2.2 and its accompanying discussion.) Currently, 64 bits prefixes are only assigned to single subnets, and such large numbers of groups in such a single subnet are difficult to imagine. Further, when a 64 bit prefix has been assigned to a single machine, the machine can perform other non-random transformations to generate addresses which do not cause conflicts.

Figure 5.3 graphs the same probabilities as Figure 5.2, with a maximum of 10 million addresses allocated. The expansion of scale shows that 32 and 40 bit group ID sizes are insufficient when millions of addresses are required, and 48 bit group ID sizes also top out. 56 and 64 bits appear sufficient up to at least 10 million addresses.

Continuing with the assumption that addresses are randomly generated, Handley [58] states that we can expect a collision once approximately $\sqrt{n}$ addresses are allocated, where n is the size of the address space. The second group ID property examined is presented in Table 5.1, which contains the

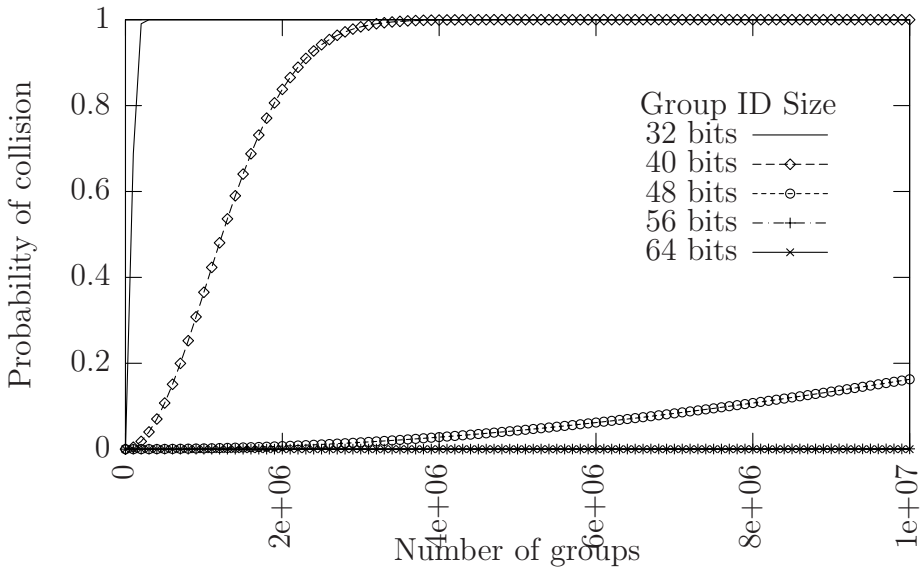


Figure 5.3: Collision Probabilities up to 10,000,000 addresses

calculated number of groups 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 values listed in Table 5.1 are used to determine whether a single node is allocating too many addresses. A threshold similar to that in the table is calculated by each node, if any node tries to allocate more addresses than the threshold then the allocation attempt is rejected. While this means

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

Table 5.1: Number of addresses per node before collisions are probable

that at most $\sqrt{n}$ of the address space will be used, the author feels it is a useful simplification at this stage. The use of thresholds is expanded on in Section 6.9.

5.4 Requirements

Thus far the reader has been shown how an address may be constructed with a unicast prefix based multicast address and a group ID of some size. However this is not enough to ensure that the address is indeed unique (remember that random generation will eventually produce a collision). What is needed is some uniqueness verifying mechanism, and this mechanism will have to operate between disparate nodes on a network. In other words, a network protocol must be designed which ensures that generated addresses are unique amongst all nodes on the network participating in the protocol.

In developing a protocol for multicast address allocation, it is wise to first ask what the requirements of such a protocol should be. If such a list can be compiled, then a base can be established against which any address allocation protocol can be measured.

Below are eight factors which the author believes are core in a good allocation protocol. These factors have been distilled from numerous sources by examining the pros and cons of the current allocation architecture [55, 61, 62, 63] and theoretical studies of the problem [5, 58]. They are first listed, then a discussion on each factor is undertaken.

1. Dynamic allocation.
2. Distributed structure.
3. Integrable.
4. Lifetime limitation.
5. Secure.
6. Fair-use enforcement.
7. Robustness.
8. Address collision limitation.

5.4.1 Dynamic allocation

Firstly, allocation must occur in a dynamic manner [61]. While static allocations have their place, they are mainly used by well known services such as the Network Time Protocol (which is assigned the addresses `ff0`*x*`::101` [65].) A dynamic mechanism would enable applications to solicit addresses from the underlying operating system on an ad-hoc basis, when the need for addresses arises. The mechanism should limit the amount of interaction required by the administrator.

5.4.2 Distributed structure

When considering processing and its location in computer networks, (at least) two options exists. The first is that processing occurs on central machines, which non-central nodes connect to and draw data from. The second is a decentralised or distributed paradigm, where processing is shared out amongst network nodes which communicate with each other.

Each has its merits and faults: centralised processing makes administration easier but provides potential bottlenecks or a single point-of-failure, while distributed processing can be difficult to maintain but a failure in one machine does not bring the whole system to its knees.

A distributed structure was chosen for three reasons. Foremost was the denial-of-service concern; that in a centralised system a single machine failure could halt address allocation on a subnet. Secondly, a centralised system would require two separate software programs for the client and the server. Lastly, a distributed system does not require any additional hardware since it runs on pre-existing network nodes. An example of a centralised system which failed may be found in [61], and strengthens the case for a distributed architecture. Handley's study of session directories also indicated that distributed directories were preferable [58].

5.4.3 Integrable

Simply put, the scheme must be easily integrable in current networks [58, 63]. This factor, while not essential to an allocation scheme, is intended to ease roll-out. Various elements affect the integration constraint. For example, only those nodes interested in running the allocation tool should have to make changes, and no alterations should be required in routers and other network equipment. Ideally the protocol should operate at the Application Layer.

This ties in closely with the distributed structure factor listed above, since administrators are not burdened with deployment of extra servers and software upgrades for routers.

5.4.4 Lifetime limitation

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 allocated addresses. In light of this, addresses must be issued with a lifetime [55, 61, 62]. Once the lifetime of an address expires, it should then be added back into the pool of unallocated addresses.

Nodes should be free to renew the lease on allocated addresses.

5.4.5 Secure

Security is a term with many different meanings, and even more interpretations. Certainly the basic aspects of Confidentiality, Authorisation and Integrity form three pillars amongst which a balance must be obtained, and is a topic which numerous sources expand on [55, 61, 62].

A *secure* protocol used in allocating multicast addresses can have multiple security needs. For instance, a designer might wish her protocol to only prevent unauthorised nodes from acquiring addresses. Here some form of authenticated protocol will suffice. However if the designer wishes to keep the contents of the messages private and confidential, then encryption will need to be employed. Note that if the objective of encrypting messages is to

hide which addresses have been allocated, the designer would do well to rethink the inclusion of encryption. This becomes obvious when one considers that all an attacker has to do is monitor which addresses multicast messages are sent to, after a sequence of encrypted messages have been noted. This is mentioned in more detail in Section 6.9. Integrity ensures the data arrives as it was transmitted and is vitally important to network communication.

5.4.6 Fair-use enforcement

It is untenable that a single application or node can allocate the entire address space to itself [5, 55, 63]. If a rogue can allocate all possible addresses, then other applications and nodes will suffer from a denial-of-service whenever they request new addresses. This is difficult to satisfy in a distributed environment, due to the lack of a distinct arbitrator, and is the major downfall of a distributed system. However it is not insurmountable, and the judicious use of quotas and thresholds can improve the situation significantly.

5.4.7 Robustness

Robustness should be a cornerstone for any network protocol [55, 61, 63]; it is included here for completeness. Specifically in this case, robustness indicates that when error conditions such as node failures or address collisions occur, the protocol should still be able to allocate legitimate addresses.

5.4.8 Address collision limitation

Earlier it was remarked that the generation of group IDs cannot simply take place in a random fashion. An allocation scheme should contain some method for decreasing the chance of an address clash [61]. Address collision avoidance is a priority because, if done properly, addresses can be allocated without causing excess network traffic.

Advanced collision avoidance is notably absent from a proposed dynamic stateless allocation mechanisms [66].

5.5 Conclusion

In this chapter the malloc problem was examined by providing a background against which a solution can be formulated.

The reader was shown how a dynamic stateless protocol would arguably provide a best-fit for possible solutions. Two statistics of the group ID were measured, namely the probabilities of collisions and the number of addresses per node before collisions are probable.

Lastly, eight requirements for an allocation mechanism were listed and defined. They are dynamic allocation, distributed scheme, integrable, limitation of address lifetimes, security, address space fair-use enforcement, robustness in the face of failure and address collision limitation.

In the next chapter a proposed solution which fulfills the eight requirements is presented.

Chapter 6

The DAOMAP Model

6.1 Introduction

At this point the reader should be fully conversant with the multicast address allocation problem, and perhaps have an idea how the solution presented will function. Chapters 1 to 4 provided the requisite background which details the function of IPv6 networking (specifically addressing, configuration and routing), the UDP protocol and the basics behind IP Multicast. Next, Chapter 5 clarified the malloc problem. Two statistics were examined which will aid in fine-tuning a potential protocol, and eight factors which will influence our protocol design were spelled out.

What remains is to design an architecture which satisfies the eight factors, implement and test it, and present the results. This chapter is concerned with the first of these, the design of a malloc architecture. We have called this architecture the Distributed Allocation Of Multicast Addresses Protocol, or DAOMAP.

First some assumptions and basic design philosophies are discussed. Then the system is presented followed by its analysis. Security issues are also worked over and the chapter is then summed up.

6.2 Assumptions

The architecture presented will be used to assign a globally unique IPv6 multicast address to an application residing on a host. It is not concerned with the allocation of addresses from the IPv4 address space, and from this point the general terms '*address*' or '*multicast address*' when applied to the DAOMAP architecture refer to a globally unique IPv6 multicast address.

While no application has so far been selected for use with DAOMAP, it is envisaged that the type of application which will utilise DAOMAP requires multicast addresses on a recurring ad-hoc basis. Examples might include conferencing and local instant messaging applications. In such software, addresses are required whenever a new communication group is intiated. If we consider a conference application, users are not constantly involved in many conferences continually, rather one or two at the most.

Unlike other allocation structures such as MASC and its friends, DAOMAP is concerned only with the assignment of single addresses. This vastly simplifies the allocation methodology, since the complex issues of expanding and moving blocks of addresses in an address space [5] are simply non-existent.

This simplification can take place without any significant loss of functionality. While DAOMAP cannot assign blocks, the author feels that this extension is unnecessary for the conferencing type of applications which will make use of DAOMAP. Block assignment is efficient in a multi-tier allocation architecture such as MASC, but wasted under DAOMAP. Since conferencing type applications do not require masses of addresses, only single addresses may be assigned.

Although the need for ad-hoc addresses will be continuous, an allocation rate in the order of tens of addresses per node per second is unwarranted for conferencing style applications, as pointed out above. This restriction is reflected in the rate limiting and threshold design, as will be seen.

In the previous chapter, security was mentioned as an important factor to keep in mind when designing a malloc protocol. Since security has already been the subject of many sizable discourses a current solution is partially

used instead of building our own. Section 6.9 contains a discussion on the chosen solution.

6.3 The DAOMAP Architecture

6.3.1 Components

A functional implementation of DAOMAP is termed a Distributed Address Allocator (DAA) and has four logical components:

1. Data store
2. Address generation module
3. Network communications module
4. Client communications module

A data store provides state retention facilities. The stored data is used by all three modules listed below, and its format is laid out in Section 6.4.

The address generation module is responsible for producing addresses which are probably unique. It utilises data found in the state store in order to avoid generating addresses which have already been allocated, or which might be allocated in the future. Section 6.5 contains more details.

The network communications module passes data between disparate DAAs, by means of a standardised protocol. It is the responsibility of this module to ensure that an address is unallocated and to lay claim to the address in such a way that all other DAAs mark the address as allocated. The protocol also makes provision for informing DAAs when an address has expired, as well as protecting addresses which have already been assigned. This protocol is simple and easy to implement, is distributed in nature and is discussed in Section 6.6.

The last component is the client communications module. This module provides calling applications with methods to invoke the functions of the local DAA. An API is presented in Section 6.7.

Figure 6.1 shows the four components of a DAA, as well as where the DAA is situated in a computing environment. The digram also illustrates which

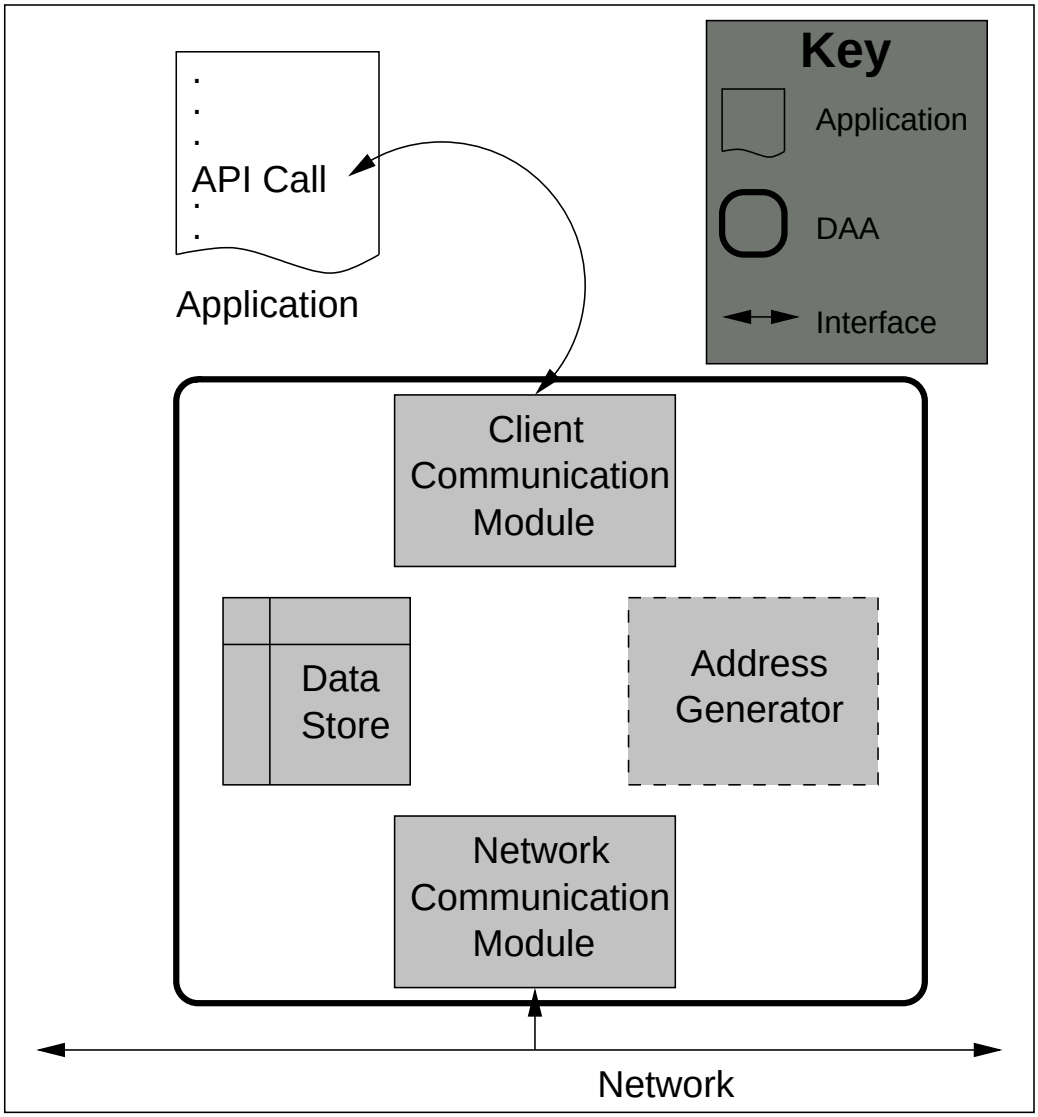


Figure 6.1: DAA Architecture

components are accessible from outside the DAA, and which are private. Two of the components are communication modules which provide interfaces to the DAA and are used by local client applications and other DAAs running on the network. Their interfaces are indicated by arrowed lines. The pair of internal components service the DAA and are therefore unaccessible to any agent other than the DAA.

In Section 2.2 it was mentioned how a multi-homed machine has more than one network interface. Since each interface has different address attributes, a separate DAA is run on each interface. Thus each machine will have one DAA per link. However if, through some arcane setup, a machine has more than one DAA per link, the only sacrifice will be in terms of resource usage; DAAs on a machine do not interact with each other except via the network communications module and will therefore not adversely affect one another. When referring to a DAA in the description below, it is assumed the description is specific to a DAA on a single link; thus 'participating DAAs' refers to all DAAs on a single link rather than all possible DAAs in a multi-link network.

6.3.2 Functional Overview

In order to better understand how DAAs interact with one another, a broad overview is needed before delving into the specifics of each component in individual DAAs. A number of details are glossed over here in favour of a clear description, however the intricacies of each step are provided later.

When a client application requests an address from the DAA; the DAA generates an address that it believes is not 'owned' by another DAA. Two items are worth noting here: the owner of an address is defined as the DAA which reserved the address by means of the process set out below, and the DAA does not have to generate an address guaranteed to be unowned. DAAs use a predetermined method in order to select an address to allocate. The method is a function which, when applied to the address last allocated produces a new address. This deterministic process allows DAAs to predict what addresses are going to be allocated by other participants, and so pre-

emptively avoid allocation attempts on those addresses. Addresses without owners are also called 'unallocated addresses'.

The DAA places the unallocated address in a packet and sends out a Claim for the address on the attached link; the address is now in the 'allocating' stage. n Claims are sent spaced t time intervals apart, after which the address is termed 'allocated', assigned to the DAA and the requesting application is passed the address. The sender of a Claim message is the claimant, and n has been set to three for our purposes.

If an address is actually allocated to another DAA, then a portion of the DAAs respond with a Collide message which informs the claimant that the address is, in fact, already allocated (the sender of the Collide might not be the actual owner of the address; this is part of the distributed nature of the protocol.) The sender of a Collide message is the collidant, and the occurrence of a Collide being issued for a Claim is termed an 'address clash' or 'collision'. When an address clash takes place the claimant immediately halts the claim process, generates a new probably unallocated address and restarts the claim process with the new address.

Figure 6.2 portrays a bird's eye-view of the process a DAA undertakes when claiming an address.

6.4 Data store

The data store consists of two tables: ADDRESSES and NODES. ADDRESSES stores information about allocated addresses and NODES holds information about other participating DAAs. The tables should be optimised for efficient lookup, since that is the operation will occur most often.

The ADDRESSES table contains a record of each address allocated on the link, regardless of the owner. Each record holds at least the following information:

$$(MulticastAddress, UnicastAddress, Lifetime)$$

$MulticastAddress$ A globally unique multicast address.

$UnicastAddress$ The multicast address owner's unicast address.

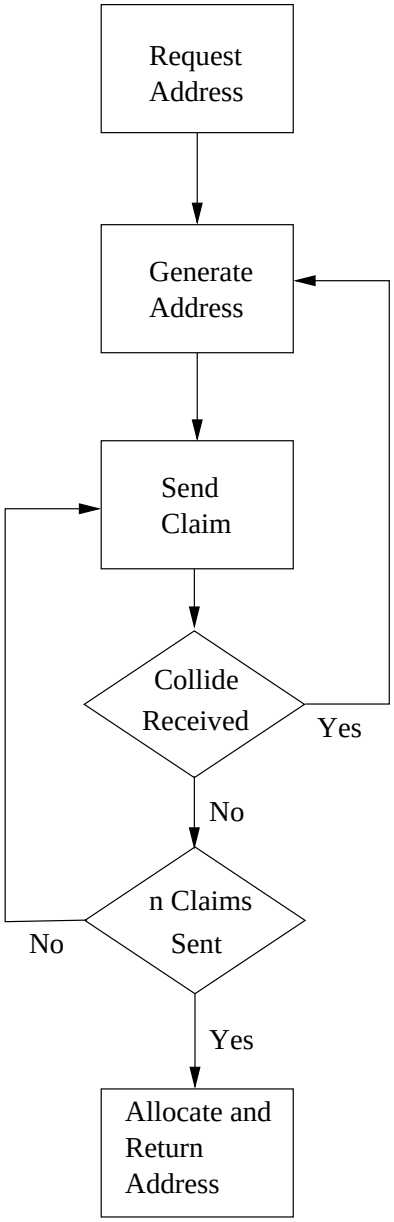


Figure 6.2: Claim process overview

Lifetime Length of time in seconds that the address is allocated to the owner, commencing from when allocation takes place. Effectively this is a countdown timer which, once expired, forces the removal of the record from the ADDRESSES table.

Lookups will be performed using the field *MulticastAddress* as the key.

The second table, NODES, stores information about specific DAAs on the link. This information consists of these fields:

(*UnicastAddress*, *AddressCount*, *ClaimCount*, *CollideCount*, *CollisionCount*, *LastAllocatedAddress*, *NextAllocatedAddress*, *Ignore*)

UnicastAddress The address of a DAA.

AddressCount The number of addresses allocated to that DAA.

ClaimCount The number of claim messages sent by that DAA.

CollideCount The number of collide messages sent by that DAA.

CollisionCount The number of collisions that DAA has shared with the local DAA.

LastAllocatedAddress The last multicast address that DAA was allocated.

NextAllocatedAddress An address which that DAA will probably attempt to allocate next. See Section 6.5.

Ignore A boolean value indicating whether messages from this node should be ignored.

Lookups will be performed using the fields *UnicastAddress* or *NextAllocatedAddress*.

6.5 Address generation module

The claim/collide mechanism used is hindered by the fact that the allocation process requires at least $n \times t$ time units, where n is the number of Claim packets sent by a claimant in the successful claim and t is the time

interval between packet transmission. If an address clash takes place, then the allocation process has to restart, more packets are transmitted, the process is extended and the overall time taken increased. Given N, the number of Claims packets sent in a successful claim without any collisions, a key performance factor is then the number of packets sent before the address is allocated. Expressed differently, the goal is to keep the difference $nt - Nt$ as close to 0 as possible. This is done by striving to complete a successful first allocation. This requires the careful choosing of addresses, such that the allocation process performs better than simple random selection of addresses (which does not scale well, as mentioned in Section 5.2).

Assuming there is a general method to improve on random selection, then presumably the first allocation attempt be more probable to succeed than pure random address selection. If the method is general enough, then subsequent allocation attempts in the case of collisions would also be more likely to succeed than pure random selection. Luckily there is a method by which collisions can be reduced, and it has two pillars: a deterministic function for address generation and a state store of allocated addresses.

The deterministic address generator coupled with the state store empower the DAA to predict what addresses other DAAs will try to assign at some point in the future. Since the DAA can also predict what addresses it will generate, potential clashes can be avoided with other DAAs, making the allocation process more efficient.

Of course, it is vitally important that all participating DAAs implement the same deterministic function otherwise address generation is no better than random allocation. Possible algorithms will now be discussed, followed by a general method which improves address selection, presented as an algorithm.

6.5.1 Deterministic Algorithms

The deterministic function chosen must, when given a set of inputs, reliably produce a corresponding set of outputs which remain identical no matter how many times the experiment is repeated. For our purposes, the set of possible

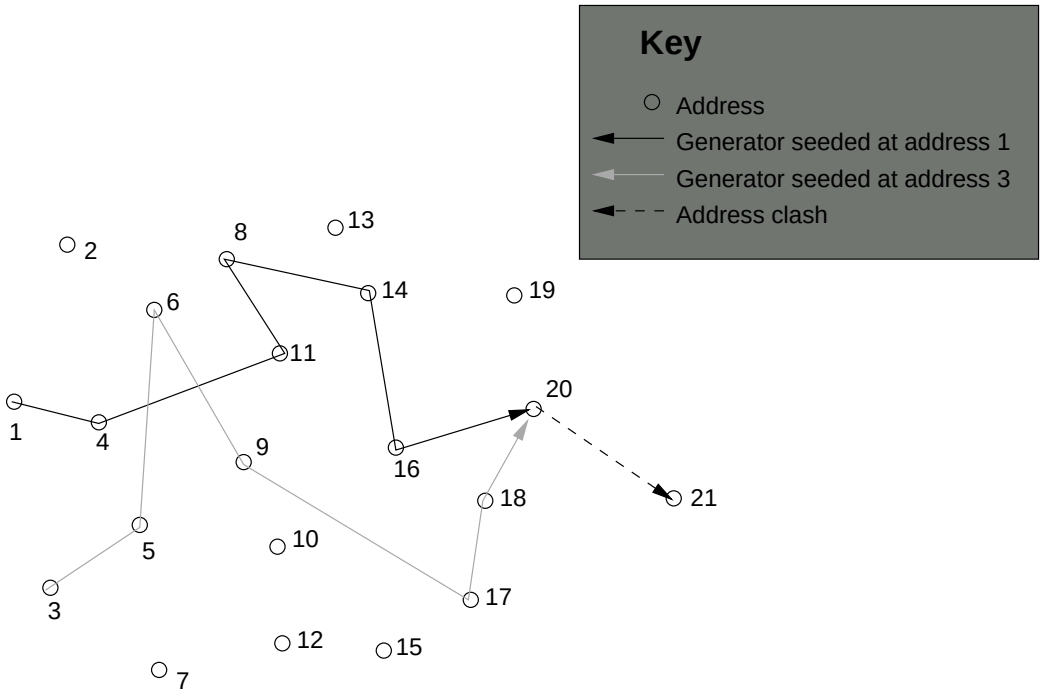


Figure 6.3: Address Generation Streams

inputs are all unicast prefix multicast addresses (described in Section 2.4.2), and the outputs must be suitable for transformation into a unicast prefix multicast address.

Figure 6.3 shows an abstracted view of a sequence of generated addresses. The paths between addresses represent the actions of the deterministic function. Note that the sequence is fixed; whenever the function is given the input `3`, its output will be `5`. Chaining the output of one iteration of the function to the input of the next iteration results in a sequence of addresses. Thus any time the function is initialised with `3`, the sequence will *always* be `3,5,6,9,17,18,20,21`. Likewise, when the function is seeded with `1`, the sequence will always be `1,4,11,8,14,16,20,21`. Observe how the two sequences converge at the address `20`, and note that from this point on the streams will be identical. It should then be clear that once an address clash occurs, sequences need to be reseeded otherwise their uniqueness property falls away.

Concerning the function itself, the only requirement is that the generated addresses are spread across the address space. The difficulty of inverting the function is less important, as security in this regard holds little benefit. If an attacker can easily deduce the input address used to generate a given address, all that has taken place is information leakage. However, a function which hides its input and does not add enormous complexity should, in this case, be chosen.

Three possible generation functions are presented, after which one will be selected. They are: the `rand(3)` function from the GNU C Library [67], the MD4 algorithm [68] and lastly the MD5 algorithm [69]. The innards of the three functions are not examined here; that is beyond the scope of this dissertation. What is of interest is the applicability of each function in generating multicast addresses.

In order to benchmark the range and distribution of each function, each was randomly seeded and run through 5000 iterations to produce 5000 group IDs of 32 bits. This process of random seeding and iterative generation was then repeated for a group size of 64 bits. (32 and 64 bits are the minimum and maximum group sizes, and demonstrate the limits of the functions.) The 5000 generated group IDs in each experiment were then sorted in ascending order by treating each group ID as a whole number. In the upcoming graphs, plots of sorted data aid in determining if the set is evenly distributed throughout the range and not biased towards a particular area, and plots of unsorted data visualises the distribution of the original sequence.

This process then leaves each function with four data sets: sorted with 32 bit group size, unsorted with 32 bit group size, sorted with 64 bit group size and unsorted with 64 bits group size. A chart is plotted for each of the four data sets, for each function leaving sixteen graphs in total. These will be discussed along with the function they represent.

GNU Libc rand(3)

The `rand(3)` function supplied with the GNU C Library version 2.3.2 implements a *"non-linear additive feedback random number generator"* [67],

according to the appropriate entry in the system manual. After conducting an experiment with the function, data was collected in the manner described previously, and the plots in Figure 6.4. The unsorted points are widely scattered, and the sorted points show an even distribution throughout the range. This means that the function produces values which are pseudo-random and evenly distributed, and are not concentrated in a few regions of the group ID space.

There is, however, a major drawback to the `rand(3)` function, which is hidden in the scale on the vertical axes. The maximum value of the range never exceeds $2^{31} - 1$. The reason for this is that the `rand(3)` call returns a 32 bit *signed* value. This drawback effectively halves the available address space for a group size of 32 bits, and reduces the 64 bit group to a quarter of its potential size. Such a weakness is untenable, especially since a large address space supports a more efficient architecture.

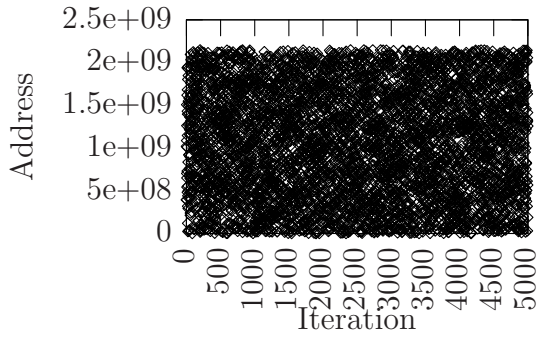
One small point about Figure 6.4(d): the bend in the curve at iteration 2500 is due to a shift in scale on the vertical axis, rather than any peculiarity of the function. In fact, the distribution is linear.

A further reason to disregard the `rand(3)` function is that it has different implementations on different platforms. For example, even when seeded with identical values, the `rand(3)` function produces a different output under Linux (with glibc 2.3.2) to NetBSD 1.6.2.

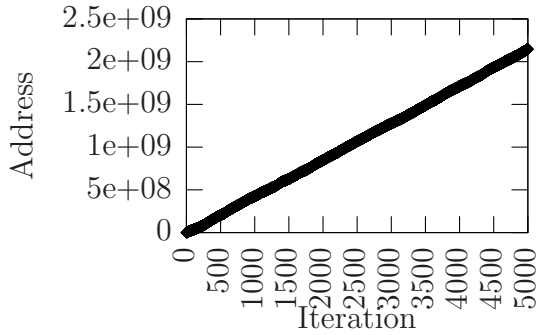
The search to find a suitable function continues.

MD4 and MD5

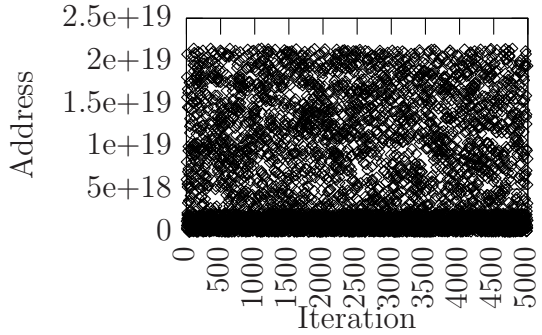
The MD4 and MD5 algorithms were invented by Ron Rivest to be used for the creation of message digests. They are published standards [68, 69] and produce a 128 bit output from an arbitrary length input. Features of these functions include speed, security, compactness and optimisation for Little-Endian machines [70]. They are examined together since they are very similar, the main difference being that MD5 is a little more *"conservative in design"* [69] and is thus slightly slower. Figures 6.5 and 6.6 show the plotted data for the two functions.



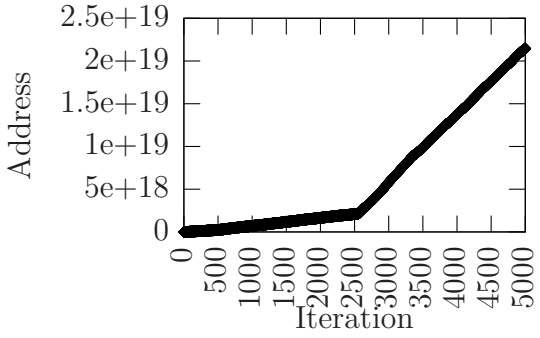
(a) Unsorted 32 bit group size



(b) Sorted 32 bit group size



(c) Unsorted 64 bit group size



(d) Sorted 64 bit group size

Figure 6.4: Function: rand(3)

The graphs show how effective the functions are, with uniform distribution virtually throughout both the sorted and unsorted sets (again, the slight bend in the sorted plots at 64 bit group sizes are a trick of scale.)

Since it has already been stated that security (ie. the level of effort required to invert the function) is not a priority, MD4 is more attractive because of its speed advantage over MD5[1]. It is thus selected for use in the address generation module.

6.5.2 General Address Selection Algorithm

We are now in a position to develop a general address selection algorithm which will reliably produce addresses that have a high chance of being unique. The algorithm is presented in pseudo-code form, and then discussed. The algorithm is not optimised; however it will be seen that optimisation is possible given certain assumptions about the data store.

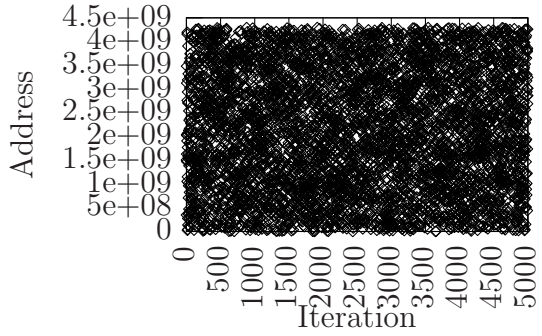
Functions used in Algorithm 1 are:

MD4 This function returns the MD4 hash of the given address, with the network prefix mapped over the hash. This is more than just the MD4 algorithm invented by Rivest; computing the digest of an address produces some random looking value which is not very useful. The MD4 function referred to here modifies the produced hash so that it conforms to the unicast prefix based multicast address format, by replacing the first few bytes with the standard prefix type, and inserting the network prefix in the correct part of the address. See Section 2.4.2.

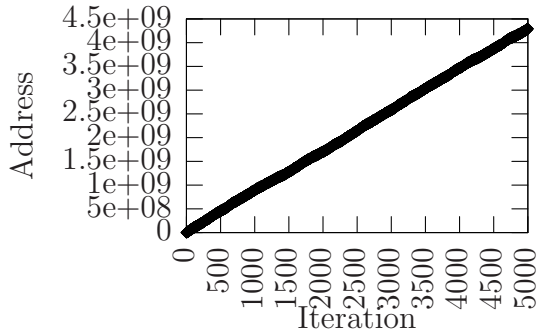
GenerateRandomAddress When called, a random address is generated.

The algorithm has two stages: the first constructs an address while the second checks if the address exists, or is likely to exist. Construction of the address occurs on line 1, where the address to be checked is built by running our MD4 function on the old address and transforming it into a new address, and line 25, where a random address is selected. The reason for this random selection is that if execution has proceeded to this point, then

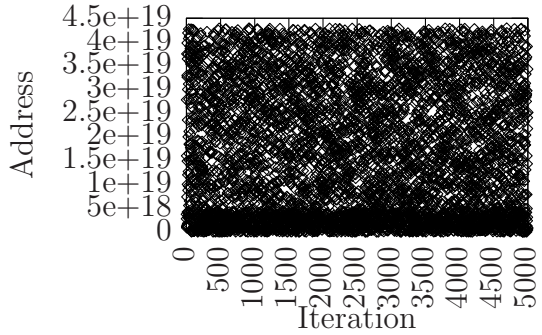
[1] In spite of this slight advantage, one could easily pick MD5 here as well without loss of functionality. The choice is near arbitrary.



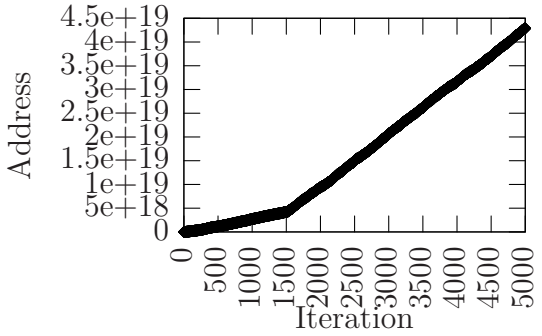
(a) Unsorted 32 bit group size



(b) Sorted 32 bit group size

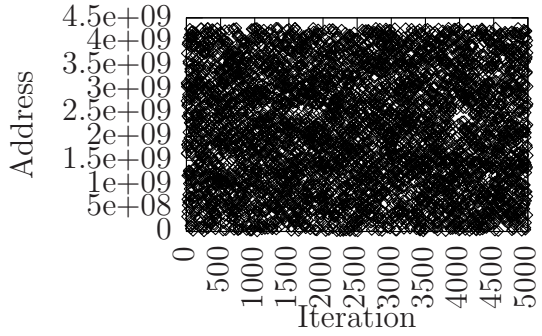


(c) Unsorted 64 bit group size

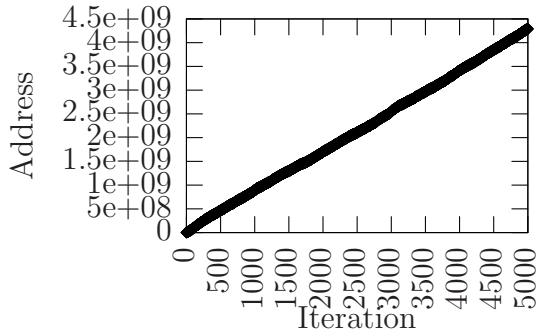


(d) Sorted 64 bit group size

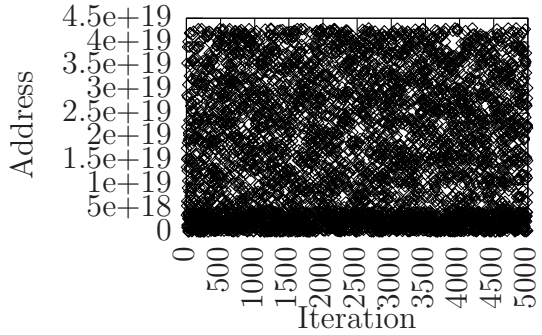
Figure 6.5: Function: MD4



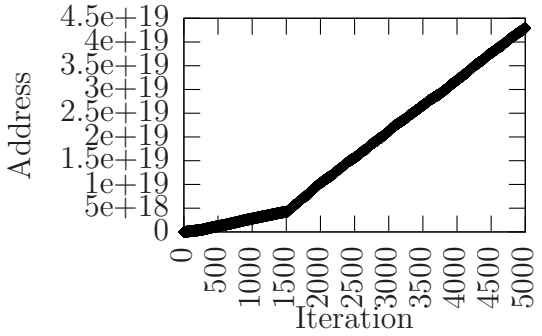
(a) Unsorted 32 bit group size



(b) Sorted 32 bit group size



(c) Unsorted 64 bit group size



(d) Sorted 64 bit group size

Figure 6.6: Function: MD5

Algorithm 1 Address generation algorithm

```
1  new_address ← MD4(old_address)
2  repeat
3          clash_found ← FALSE
4          for each record rec in ADDRESSES
5          do
6                  if new_address = rec.MULTICASTADDRESS
7                  then
8                          clash_found ← TRUE
9                          break
10                 endif
11         endfor
12         if clash_found = FALSE
13         then
14                 for each record rec in NODES
15                 do
16                         if new_address =
                                   MD4(rec.LASTALLOCATEDADDRESS)
17                         then
18                                 clash_found ← TRUE
19                                 break
20                         endif
21                 endfor
22         endif
23         if clash_found = TRUE
24         then
25                 new_address ← GENERATERANDOMADDRESS()
26         endif
27 until clash_found = FALSE
28 old_address ← new_address
```

an address clash has occurred and the address sequence must be reseeded with a random value.

The verification stages (lines 3—22) check that the address is not currently allocated, by looking in the ADDRESSES table, and that it is not going to be allocated, by applying the MD4 function to the last allocated address of each entry in the NODES tables.

Algorithm 1 is unoptimised, as mentioned before, but is generic and can be implemented without the need for additional support. The main limiting factor is the capability of the data storage mechanism; if it only supports sequential access then each record needs to be retrieved to examine it. However, if the data store supports advanced indexing then improvements can be made. A further restriction is the running of MD4 on each record in the ADDRESSES table as the record is retrieved. It would be better to reposition the MD4 function at record insertion; this ensures that the running time is not compounded by the number of entries in the NODES table. An optimised version of the address generation process is given in Algorithm 2.

This new algorithm assumes that a database of some kind is supported, and that it allows the defining of secondary indices (that is, a key other than the primary key can be used to look up entries. In this case, the *NextAllocatedAddress* field in the NODES table is used.) Also, the MD4 function has been removed from line 16, and it is now run when data is inserted (as will be demonstrated when Algorithm 4 is discussed later.)

New functions used in Algorithm 2 are:

AddressLookup Given a Multicast address, return the corresponding entry from the ADDRESSES table, otherwise return NULL if no such entry exists. This function is used in numerous locations to read entries from the ADDRESSES table.

NodeLookupIndex Given a Multicast address, return the corresponding entry from the NODES table using the *NextAllocatedAddress* field as the key, otherwise return NULL if no such entry exists. This function is only called from within the address generation module, since it is searching for possible future address allocations.

Algorithm 2 Address generation algorithm

```
1  new_address ← MD4(old_address)
2  repeat
3          clash_found ← FALSE
4          if AddressLookup(new_address) ≠ NULL
5          then
6                  clash_found ← TRUE
7          endif
8          if clash_found = FALSE
                   and NodeLookupIndex(new_address) ≠ NULL
9          then
10                 clash_found ← TRUE
11         endif
12         if clash_found = TRUE
13         then
14                 new_address ← GenerateRandomAddress()
15         endif
16 until clash_found = FALSE
17 old_address ← new_address
```

6.6 Network communications module

So far the internal structure of data storage has been introduced, along with the method which will be used to generate addresses. Our collective attention is now turned to the network component that a DAA must implement. A protocol is defined by which DAAs may communicate, as well as the steps to be taken when data is received or sent.

All messages on the network are sent to the All Nodes address, `ff02::1` (Section 2.4.2), which is highly suitable since every IPv6 capable host on the link is required to join this group. The data is transferred using UDP packets (see Chapter 3) on port 49152. The selection of 49152 as the port number is not arbitrary; this is the start of the port range defined by IANA to be private or unregistered [71]. If DAOMAP were to become widespread, then an official port would be requested.

DAOMAP messages are simple, and have the following format:

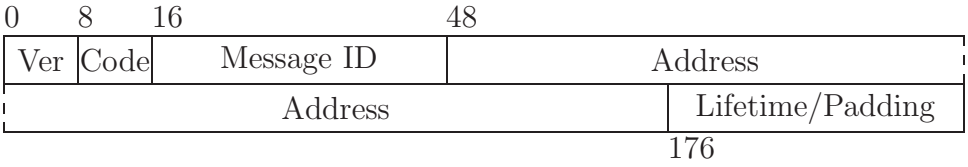


Figure 6.7: DAOMAP Message Format

The fields are given below:

Ver DAOMAP version number, currently `0x01`. (1 byte)

Code Indicates whether the message is a Claim or Collide. A Claim has code `0x03` and a Collide has code `0x04`. (1 byte)

Message ID In a Claim message, this field determines the packet number in a Claim sequence, otherwise in a Collide message it is zero. (4 bytes)

Address A unicast prefix based IPv6 multicast address. (16 bytes)

Lifetime/Padding A Claim message contains the lifetime a message is valid for, in a Collide message this field is zeroed out. (4 bytes)

Thus DAOMAP messages are always 22 bytes in length. The two 32 bit fields (**Message ID** and **Lifetime**) are always in network byte order [7]. The

Address field is treated as a sequence of four 32 bit words with the most significant bit of the address in the first word, and each word is converted to network byte order before transmission. The 8 bit fields require no meddling.

It is not enough to only define how messages are exchanged; behaviour on receipt of messages as well as API triggered procedures must also be prescribed. The events and their responses which need clarification are:

- Address allocation.
- Receipt of a Claim message.
- Receipt of a Collide message.

Address Allocation

When a client application issues a request for an address via the API, the DAA generates an address, issues 3 Claim messages spaced t time units apart and then returns the address to the application, assuming no Collides are received. The pseudo-code is given in Algorithm 3. In that algorithm and subsequent ones, *message* refers to a DAOMAP message of the format described previously and the fields of the message are concatenated to *message* with a period (.).

Unseen functions used in Algorithm 3 are:

GenerateAddress Returns a probably unique address. The internals of the function is the general allocation process, described in Algorithm 2.

SendPacket Transmits the DAOMAP message to the group.

Pause Halts execution for the specified time period.

Important to note here is that if a Collide is received, then GenerateAddress on line 17 will produce a random address.

Claim Received

When a DAA issues a Claim message as above, the receivers verify that the sender is not to be ignored and determine if the claim attempt is for an address listed in the addresses table. If not, and another allocation attempt

Algorithm 3 Address Allocation

```
1  message.VERSION ← 0x01
2  message.CODE ← 0x03
3  message.ADDRESS ← GenerateAddress()
4  message.lifetime ← RequestedLifetime
5  allocated ← TRUE
6  repeat
7          i ← 3
8          message.MESSAGEID ← message.MESSAGEID +
                   message.MESSAGEID mod 3
9          while i > 0 and allocated = TRUE
10         do
11                 SendPacket(message)
12                 Pause(t)
13                 if receive Collide for message.ADDRESS
14                 then
15                         allocated ← false
16                         Increase the CollideCount field in NODES
                                   for src, create entry if none exists.
17                         message.ADDRESS ← GenerateAddress()
18                 endif
19                 i ← i − 1
20                 message.MESSAGEID ← message.MESSAGEID + 1
21         endwhile
22 until allocated = TRUE
23 return message.ADDRESS to application
```

for the same address is not ongoing, then a timer event is queued to add the address in *message*.MESSAGEID modulus 3 seconds, since a successful allocation requires three Claims. If an allocation attempt for the same address is in progress by another node (either this node or any other node) then a Collide is randomly issued ie. not all DAAs will respond. If the address already exists and is owned by the sender of the Claim message, then the *Lifetime* field in ADDRESSES is updated. Lastly, if the address already exists in the ADDRESSES table but the source of the Claim does not match the owner as listed in the table, the message has no effect.

This process has been formalised in Algorithm 4, where *message* has the same meaning as before, *src* is the IPv6 unicast address of the sender of the Claim, *rec* is a row entry from the ADDRESSES table returned by a lookup of an address and *myaddress* is the IPv6 unicast address of the local DAA. The number of DAAs which will respond with a Collide when a clash is detected, is determined by the variable RESPONSE_RATIO. This variable is expressed as a probability between 0 and 1 that a node will transmit a Collide message.

New functions used in Algorithm 4 are:

IGNOREMESSAGE Returns TRUE if the sender is to be ignored, FALSE otherwise. See Section 6.9.

SEARCHQUEUEDALLOCATION Returns a record containing the details of an allocation attempt present in the timer queue.

SENDCOLLIDE Constructs a Collide message for the provided address and transmits the message to the group.

INSERTADDRESS Inserts the given address into the ADDRESSES table. The MD4 function is run on the address to produce the *NextAllocatedAddress* field.

TOTALNODES Returns the total number of rows in the NODES table.

RAND Generates a random number between 0 and 1.

REMOVEQUEUEDALLOCATION The allocation event for the given address is removed from the timer queue.

QUEUEALLOCATION Creates an allocation event for the given address, and places it on the timer queue.

Collide Received

The last event of interest arises when a Collide message is received. This message is either in response to an allocation attempt by the local DAA, or involves other DAAs. If it is the former case, then the allocation attempt is cancelled and restarted with a new address as per Algorithm 3 (the generic address allocation process). On the other hand, if a Collide arrives for an address which a foreign DAA is attempting to allocate (evident by the presence of an allocation event in the timer queue), then that allocation attempt is cancelled and a Collide is randomly issued.

When a Collide arrives for an address which has already been allocated (simply, it is present in the ADDRESSES table), no action is taken. This ensures that the ADDRESSES table remains consistent across all nodes.

Algorithm 5 specifies the steps to be taken when a Collide is received.

6.7 Client communications module

The last of the four internal components is the client communication module. This takes the form of an API, and three calls are specified:

MCAST_ADDR_ACQUIRE($lifetime$) Returns a unicast prefix based IPv6 multicast address valid for $lifetime$.

MCAST_ADDR_UPDATE($address, lifetime$) Renews the lease on $address$ to $lifetime$.

MCAST_ADDR_RELEASE($address$) Informs the DAA that it may release $address$.

The DAA must carefully determine that the application calling either one of MCAST_ADDR_RELEASE or MCAST_ADDR_UPDATE on an address is the same application which called MCAST_ADDR_ACQUIRE. Otherwise it is

Algorithm 4 Claim Receipt Process

```
1  Claim message received from src
2  Increase the ClaimCount field in NODES for src,
           create entry if none exists.
3  issue_collide = FALSE
4  if IGNOREMESSAGE(src) = TRUE
5  then
6           return
7  endif
8  rec = ADDRESSLOOKUP(message.ADDRESS)
9  if rec = NULL
10 then
11          event = SEARCHQUEUEALLOCATION(message.ADDRESS)
12          if event ≠ NULL and event.SRC ≠ src
13          then
14                  issue_collide = TRUE
15                  Increase the CollisionCount field in NODES for src,
                            create entry if none exists.
16                  if event.SRC = MYADDRESS
17                  then
18                          Cancel and restart allocation process.
19                  endif
20                  REMOVEQUEUEDALLOCATION(message.ADDRESS)
21          elseif event = NULL
22          then
23                  QUEUEALLOCATION(message.ADDRESS, src)
24          endelseif
25 else
26          if rec.UNICASTADDRESS = src
27          then
28                  Update Lifetime in ADDRESSES table for src
29                  INSERTADDRESS(message)
30          else
31                  issue_collide = TRUE
32          endif
33 endelse
```

```
34  if issue_collide = TRUE
35  then
36          total_nodes ← TOTALNODES()
37          response_nodes ← total_nodes * RESPONSE_RATIO
38          for i = 1 → 3
39          do
40                  if RAND() < (response_nodes/total_nodes)
41                  then
42                          SENDCOLLIDE(message.ADDRESS)
43                          PAUSE(t)
44                  endif
45          endfor
46  endif
```

Algorithm 5 Collide Receipt Process

```
1   Collide collide_message received from src
2   Increase the CollideCount field in NODES for src,
            create entry if none exists.
3   if IGNOREMESSAGE(src) = TRUE
4   then
5           return
6   endif
7   If Collide is for an address this DAA is attempting to allocate,
            then Algorithm 3 applies, otherwise continue.
8   event = SEARCHQUEUEDALLOCATION(collide_message.ADDRESS)
9   if event ≠ NULL
10  then
11          REMOVEQUEUEDALLOCATION(collide_message.ADDRESS)
12          if event.SRC ≠ MYADDRESS
13          then
14                  Clear NextAllocatedAddress field in NODES for src.
15          endif
16  endif
```

possible for a rogue application to change lease details of addresses not allocated to that application.

6.8 Operational Aspects

Now that the functions of the four units have been specified, a closer examination of their inter-operation is warranted. In particular, an example of packet exchange would benefit the reader and various, minor, details need to be expounded.

Dealing with the last of these inter-operation aspects first, on startup the ADDRESSES and NODES tables are assumed to be empty. The DAA chooses an interface (which is not a loopback device) to bind to and determines the best prefix to use on that interface. The best prefix would be the longest globally routable prefix available. Prefixes can have limited lifetimes; if this is the case then the DAA needs to refresh the prefix when its lifetime expires. In the same vein, when the lifetime of an address expires, that address should be removed from the ADDRESSES table and the relevant entry in the NODES table has its *AddressCount* field decremented. Of course, nothing stops the group owner from continuing to use the address; however that type of lifetime enforcement is beyond the scope of this work. The initial seed for the deterministic function would preferably be based on a hardware address present in the network interface. If no such address is present, a random seed is selected.

With regards to an example of how packets are exchanged, consider Figure 6.8. Node C wishes to allocate an address a_1 and so generates a Claim message which is distributed to all participating DAAs. Unfortunately this address is already owned by Node D, but Node A generates the appropriate Collide, again to all DAAs. Node C receives the Collides and restarts the allocation process, this time with new address a_2. Luckily no conflicts are caused and the address is successfully allocated after 3 Claims have been sent. The other nodes also issue Collides, spaced some time units apart, until at least three Collides have been sent.

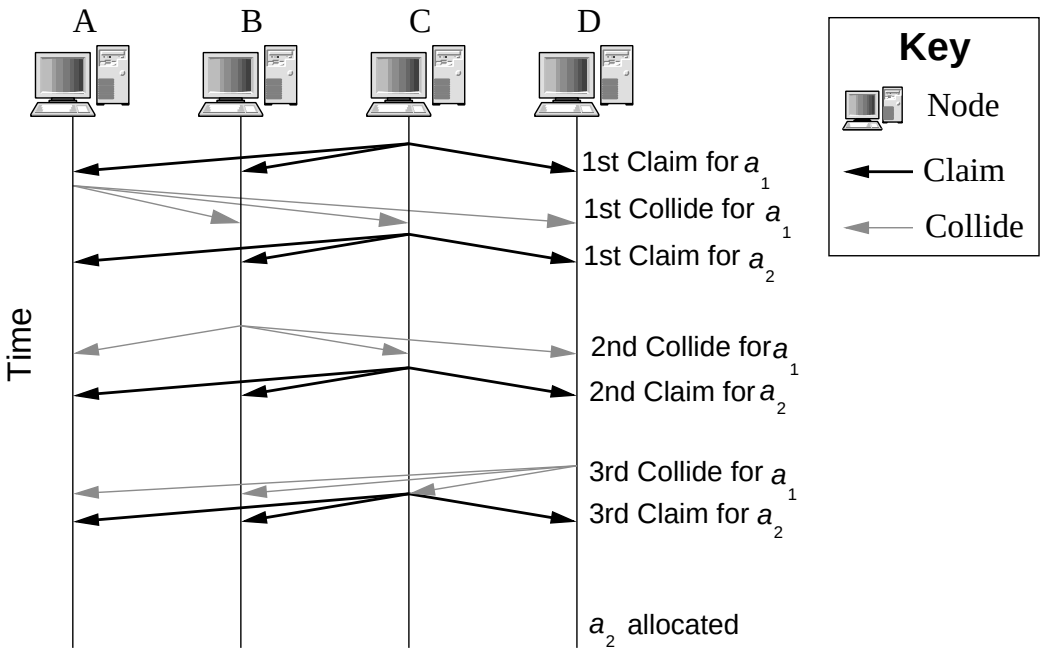


Figure 6.8: DAOMAP Packet Exchange

6.9 Security

In Section 5.4.5 the author stated that security is a vital feature of address allocation. The problem is quantifying that statement. How can one ensure that a system is secure? Can a system ever be considered completely secure? Can a system be both secure and usable? Many more such questions can be asked, with each answer contributing a small piece to the overall idea of 'security'.

The author has chosen to demonstrate DAOMAP's security by first identifying possible weaknesses and well known attacks against multicasting. The task is then to test against the catalogued risks. Kruus [72] lists six general threats against IP Multicast traffic, namely: eavesdropping, unauthorised data generation, alteration, destruction, denial of service and illegal data use. Two basic security requirements as stated by Canetti et al. [73] are *secrecy* and *authenticity*. Keeping in mind all this, it is not the author's intention to provide a complete introduction to the area of secure multicast; indeed there

are numerous texts which do just that (Kruus' and Canetti's provide a departure point.) What is important is the application of an external security mechanism to DAOMAP.

The current Internet standard for security at the IP level is IPSec [74], which provides for encryption of packets at the network layer. Since it was not designed to support multicast groups, we will not delve into too much detail, since the application of IPSec to DAOMAP is a stop-gap until a multicast alternative is available, however a brief introduction will aid the reader in following the discussion below. IPSec allows hosts or gateways to specify Security Associations (SA) between itself and other hosts or gateways (in the case of DAOMAP, between the host and a multicast group.) Each SA has various fields, detailing how packets applicable to the SA should be dealt with. For example, what encryption algorithms are to be used, what keys to use and anti-replay devices.

IPSec's Encapsulating Security Payload (ESP) [75] in transport mode was selected to secure the multicast data used by DAOMAP. Two modes are available for ESP, transport and tunnel. Tunnel mode is normally used to create Virtual Private Networks, where whole IP packets on a secure network are encrypted by an end-point, transmitted across an insecure network and arrive at the other end-point where the encrypted packet is decrypted and placed on the secure network on the far side. Transport mode on the other hand secures everything *after* the IP header, and is normally used by applications such as DAOMAP. Features of this mode include *"confidentiality, data origin authentication, connectionless integrity, an anti-replay service . . . and limited traffic flow confidentiality"* [75]. Of interest are origin authentication and connectionless integrity.

In practical terms, whenever a packet leaves a DAA addressed to the group, it is encrypted using a specified encryption scheme by a common group key which is manually set[2]. The recipient's IP stack decrypts the payload and passes the clear data to the application. Unfortunately this manually keying is required since automatic keying with IKE [76] is not possible. Further, since all messages to that group are encrypted, it may

[2]Distribution of the key is beyond the scope of this work.

impact on the functioning of other protocols.

A drawback of IPSec applied to multicast is that, while messages can be secured within a group, it is not possible to authenticate individual sources. Hence it is only possible to say that a message has arrived from a node which possesses the common group key, rather than from a particular node[3]. This is partly mitigated by assuming if the node has knowledge of the group key, then it is 'trusted', however more protection within the sheltered confines of the group is needed.

Returning to Kruus' threats, eavesdropping, data generation and alteration and illegal data use by attackers outside the group are solved by the use of IPSec. The remaining two, denial of service and data destruction, are seemingly unsolvable under current Internet conditions. If either of these attacks occur, a more coordinated response involving numerous parties (for example, network and system administrators as well as service providers) is required. Thus IPSec is used as far as possible to protect against outside attacks, and at the point IPSec fails to provide external security, DAOMAP will become vulnerable from an external position.

One additional attack which does exist for external adversaries is the monitoring of MLD messages which emanate from a DAA (See Section 4.2.1). In this way it is possible to build a list of addresses which are allocated to a DAA by observing which unicast prefix based multicast addresses the node joins. This information leakage attack has denial of service implications, and is discussed shortly.

The process of securing DAOMAP inside the group also asks which of the six threats are applicable. Eavesdropping is a non-issue since all members ideally need to receive all messages. Data generation suffers from the problems mentioned previously about source authentication. Alteration in the absence of source authentication reduces to the problem of data generation, since an attacker might intercept a message, change data and encrypt it and transmit the message. Destruction of messages is a network issue and

[3]Work by the Multicast Security (msec) Working Group at the IETF with regards to many aspects of multicast security is dealing with this source authentication problem [77, 78], however little has been standardised.

would involve more than DAOMAP could provide, although judicious use of the MESSAGEID field in the packet combined with source authentication may provide a possible solution in keeping track of messages which have not arrived. Illegal use of the data is limited to knowing which addresses are assigned to any particular DAA.

The meaty concern facing DAOMAP are denial of service attacks (DoS), both from outside as well as inside the group. As mentioned before, if an external attacker E notices that node N has issued a MLD Report for address a, then E can flood the link with messages addressed to a. These may or may not overcome N, depending on the nature of resources available to N; however the DAA on N will never receive the messages since the network layer will have filtered them out as they will not be encrypted with the group key. Source Specific or Filtered Multicast (Section 4.2.1) would allow for filtering at the router of possible external DoS attacks.

Internal DoS attacks are most important, and tie in closely with fair use enforcement. Four instances of DoS attacks are relevant to DAOMAP, where an attacker could:

1. try claim all possible addresses in the address space.
2. force collisions by issuing Collides in response to legal Claims.
3. force collisions by issuing Claims for addresses known to be allocated or in the process of being allocated.
4. combine the above attacks with changing MAC and IPv6 addresses.

6.9.1 Claiming all addresses

To prevent a single DAA from claiming all addresses in the address space, a ratio is required to determine whether a node has too many addresses. Table 5.1 provided estimates at which address collisions become likely; however in terms of ratios this becomes $\frac{\sqrt{2^{prefix-length}}}{number\ of\ nodes}$. Once a node has more than its share of the space, that is, $AddressCount$ in the NODES table rises above $\frac{\sqrt{2^{prefix-length}}}{number\ of\ nodes}$, Collide messages are issued randomly as per Algorithm 4. This prevents a reflection attack whereby an attacker can force a single DAA

to respond to all Claims with Collides, and thereby find itself ignored.

6.9.2 Collide attacks

If a maleficent DAA starts issuing Collides for every Claim message received then its *CollideCount* will increase. A time-based metric is needed; so many Collide messages per second are allowed, after which the source is ignored.

6.9.3 Claim attacks

Perhaps a DAA decides to send Claim messages for addresses known to be allocated. In line with DAOMAP, a certain ratio of the nodes will respond with Collide messages. While this happens, the *ClaimCount* and *CollideCount* of the various parties will be increasing. Again, a time-based threshold is suitable for protection against Claim attacks. Since the attacker cannot predict which nodes will respond, the adversary effectively finds itself attacking the entire group, rather than a single node.

An adversary could also attempt to issue Claims for addresses which are in the process of being allocated (that is, the attacker receives a Claim for an address and immediately issues a Claim for the same address.) The goal of such action might be to prevent allocation of an address. In this particular case, the *CollisionCount* field is increased only for the sender of the second Claim (evident in Algorithm 5). Once this field crosses another time-based threshold, the node is ignored. We discuss the relative functionings of the thresholds shortly.

6.9.4 Combined attacks

The last of the attacks surveyed uses spoofing of source MAC and IPv6 addresses in conjunction with any of the above three attacks. The issue is that entries in the NODES table are per-IP address. If an attacker can continually change her address, then the safeguards afforded by the thresholds are rendered useless. A program such as `arpwatch` [79], which keeps track of MAC

and IPv6 addresses would allow group members to determine where attacks originate from.

6.9.5 Thresholds and packet discarding

In Algorithms 4 and 5 the function IgnoreMessage was used with the proviso that it be defined later. That point has arrived: Algorithm 6 details how IgnoreMessage operates. Once a DAA has been flagged as ignored by this function, it remains ignored.

The only new function introduced in Algorithm 6 is NodesLookup() which returns that entry from the NODES table where the primary key matches the passed address.

The time-based thresholds operate by increasing the *ClaimCount*, *CollideCount* and *CollisionCount* values at particular events such as Claim or Collide receipt, and decrementing the values at regular intervals (say, every second reduce the count by 1.)

A relation needs to be defined between the three maxima of the values, in order that attacks are correctly prevented. To summarise the functions of each threshold, *ClaimThreshold* prevents simple Claim flood attacks, *CollideThreshold* prevents an adversary from preventing allocation from taking place by issuing Collides for every Claim received and *CollisionThreshold* stops an evil-doer from denying allocations by duplicating allocation attempts.

We define two relations:

Relation 1 $ClaimThreshold > CollideThreshold$

Relation 2 $ClaimThreshold > CollisionThreshold$

The reasoning for Relation 1 is that to prevent a Collide attack, the attacker should be ignored before the legal claimant. Relation 2 is similar in ensuring that an adversary is ignored whenever a attempt is made to interfere with the allocation process by issuing duplicate Claim messages.

As will be seen, when defining the actual numerical relationships we chose to use rate limits in order to maintain the relationships, rather than absolute thresholds. However the relationships still hold.

No relation is defined between *CollideThreshold* and *CollisionThreshold*, since the attacks they prevent are unrelated.

Now that the function has been presented, the general mechanism for rate limiting is described. Only the theory is dealt with here; simulations follow in Chapter 7.

As described earlier in Section 6.4, each node has an entry in the NODES table where counts of Claims and Collides received from that node, as well as number of Collisions caused, are stored. These counts are periodically decremented by a time event (Figure 7.4). If, at any stage, a count exceeds the threshold as presented in Algorithm 6, it is marked as ignored and all future packets from that node will be discarded.

Eight parameters affect the protecting function IGNOREMESSAGE(), and each is examined in detail. Three thresholds, respectively CLAIMTHRESHOLD, COLLIDETHRESHOLD and COLLISIONTHRESHOLD, as well as the RESPONSE_RATIO, are used directly in the function to determine when rate boundaries have been crossed. The remaining four, namely *DecrementInterval*, *ClaimReduce*, *CollideReduce* and *CollsionReduce* are timer-related and provide limits per time interval for various events.

RESPONSE_RATIO As mentioned in Section 6.6, this parameter controls how many nodes respond when Collides need to be issued. Depending on the number of nodes present, this value should shift; the justification for this is easily understood. Suppose RESPONSE_RATIO is defined as 0.1, and the total number of participating DAAs is less than ten, then it is very probable that occasionally no Collides are sent when they should be. If the opposite extreme is visited and RESPONSE_RATIO is set to 0.9, then generally 90% of the nodes will issue Collides when such messages are needed, which is undesirable due to scalablility concerns.

Algorithm 6 Ignore Message Function

```
IgnoreMessage(source)
 1  rec = NodesLookup(source)
 2  if rec.ignore = true
 3  then
 4          return true
 5  endif
 6  fair_portion ← sqrt(2^(prefix−length)) / TotalNodes()
 7  if rec.addresscount > fair_portion
 8  then
 9          Update nodes such that the Ignore field is true for source.
10          return true
11  endif
12  Calculate thresholds (Section 6.9.5).
13  if rec.claimcount > ClaimThreshold
14  then
15          Update nodes such that the Ignore field is true for source.
16          return true
17  endif
18  if rec.collidecount > CollideThreshold
19  then
20          Update nodes such that the Ignore field is true for source.
21          return true
22  endif
23  if rec.collisioncount > CollisionThreshold
24  then
25          Update nodes such that the Ignore field is true for source.
26          return true
27  endif
28  return false
```

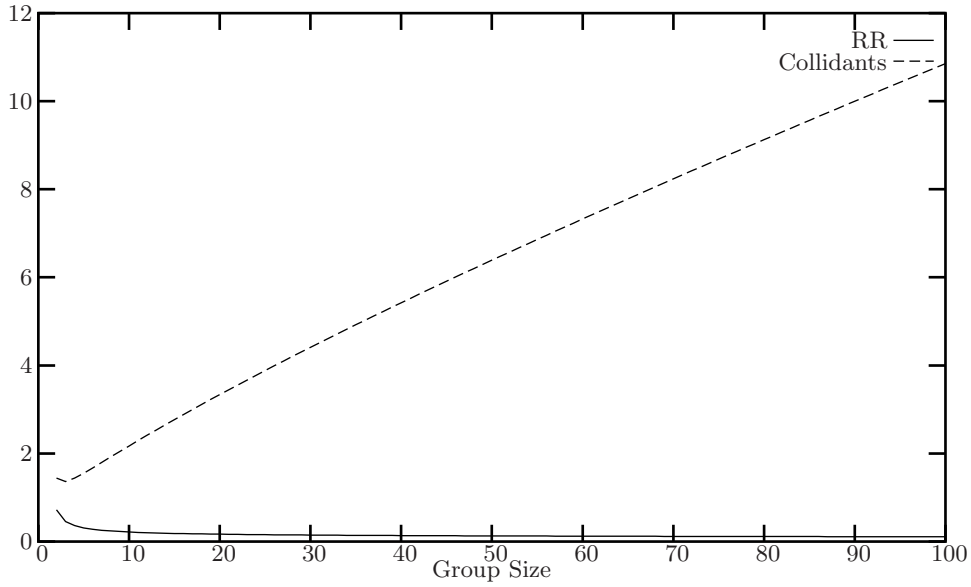


Figure 6.9: Behaviour of RESPONSE_RATIO

We define the RESPONSE_RATIO as follows:

$$\text{RESPONSE\_RATIO} = \frac{1}{2\ln(\text{TOTALNODES}())}$$

This definition has two major implications. One is that, on average, at least two nodes will respond when a Collide is required. The second is that the RESPONSE_RATIO does not grow linearly, which is important if hundreds of groups members (or more!) are expected. The graph depicted in Figure 6.9 shows how this definition of the RESPONSE_RATIO produces a flattening curve in the average number of Collidants which will reply when a collision is detected.

The graph also shows how the RESPONSE_RATIO gradually decreases when the size of the group is increased.

DECREMENTINTERVAL This is the period between all counter decrements described below, and should be less than the period between legal Claim or Collide transmissions. This restriction ensures that legal DAAs can transmit at the peak rate without being punished. By default, the period between Claim or Collide messages is one second, so DECRE-

RESPONSE_RATIO	0.108574
COLLIDEREDUCE	10.857363
CLAIMREDUCE	108.573624

Table 6.1: Parameter values with 100 nodes

MENTINTERVAL should be some interval marginally shorter than 1 second (for instance, $950ms$.)

COLLIDEREDUCE Every DECREMENTINTERVAL, each *CollideCount* in NODES is decremented by this amount. In effect, it is the maximum number of Collides allowed per DECREMENTINTERVAL. The value is tightly bound to RESPONSE_RATIO (Section 6.6) since the more nodes on the link, the more Collides would need to be issued. It has been set to RESPONSE_RATIO $\times$ TOTALNODES(). Basically, this value defines the maximum rate a node may legally send packets at.

CLAIMREDUCE Similar to COLLIDEREDUCE, but limits the rate of Claim packets. Since it has already been determined that more Claims than Collides are allowed (Section 6.9.5), it is set to 50% over the maximum rate for Collide packets, or $1.5 \times$ *CollideReduce*.

The determination of this constant has taken place carefully; a subtle attack occurs when the number of Claims allowed far exceeds the legal rate for Collides. To demonstrate the problem, assume that the group size is 100 and a node may, without fear of penalty, issue Claims at a rate ten times that permitted for Collides. Using the definitions of RESPONSE_RATIO, COLLIDEREDUCE and CLAIMREDUCE as given above, then Table 6.1 provides the numerical values for each of the three parameters.

Now an adversary who is able to send Claims does the following: he notes at least 108 addresses which are already allocated. He then creates a Claim for each address and sends these off rapidly. Every other node that receives these Claims and also has a record for each address showing the address to be allocated, schedules a Collide sequence for that address, in keeping with DAOMAP.

Thus, every second each node examines all 108 Collides and (using the RESPONSE_RATIO, Algorithm 4 abstracts this process), decides whether or not to send a packet. In this example, on average the node will send a packet 0.108574% of the time, or

$$0.108574 \times 108 = 11.725992$$

packets. But this is greater than the allowed COLLIDEREDUCE of 10.857363. Thus the attacker could force some nodes to exceed the legal Collide transmission rate. To ensure this does not happen, Relation 3 needs to hold for all possible group sizes. While this relation remains true, an attacker cannot force legitimate nodes from exceeding their allowed Collide sending rate.

Relation 3 RESPONSE_RATIO $\times$ CLAIMREDUCE $<$ COLLIDEREDUCE

Clearly, setting the CLAIMREDUCE to ten times that of the COLLIDEREDUCE is excessive; and so a factor of 1.5 was chosen. The difference between the two multipliers is clear in Figures 6.10 and 6.11.

In each graph, the left and right side of Relation 3 are plotted against group size. If an intersection occurs between the two lines, then at some stage Relation 3 does not hold. In the first graph, Figure 6.10, the only overlap occurs when there are 2 nodes in the group. In can be argued that this simple case is of no interest; if the attack succeeds, only the attacker will ignore the victim. The remaining points show a widening of the lines; this indicates that the larger the group size, the greater protection is provided against this type of attack.

However, when examining Figure 6.11 an overlap is visible until the group size exceeds 148, and the constant 10 is then completely unsuitable for our purposes.

COLLISIONREDUCE Similar to COLLIDEREDUCE, but counts the number of duplicate Claims a node can issue. It is set to RESPONSE_RATIO $\times$ TOTALNODES().

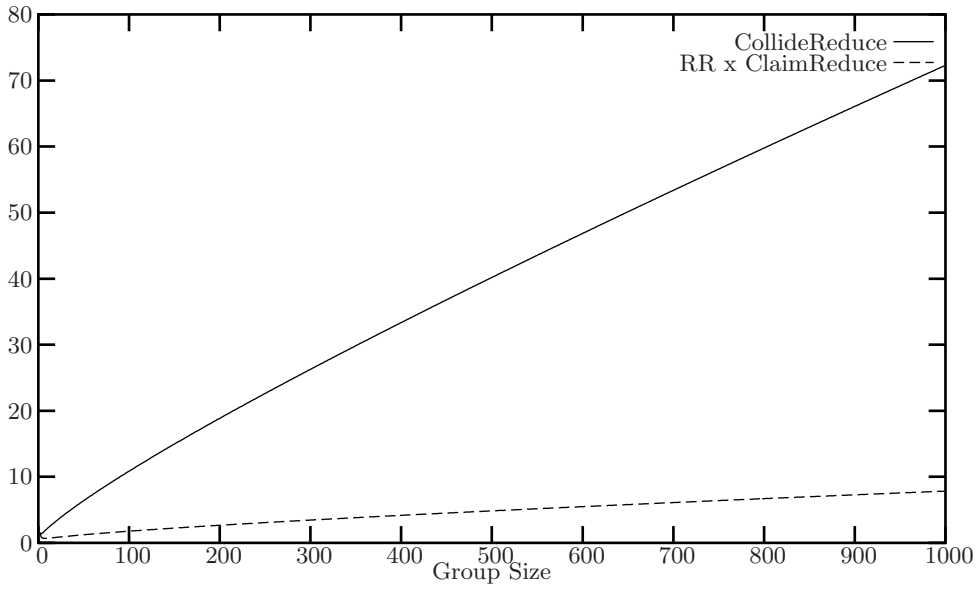


Figure 6.10: ClaimReduce = 1.5 × CollideReduce

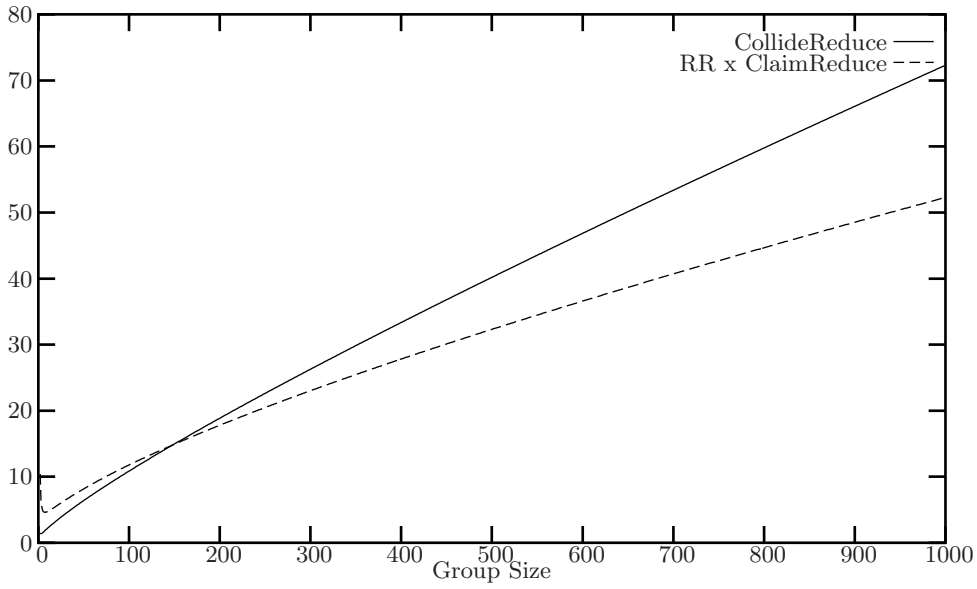


Figure 6.11: ClaimReduce = 10 × CollideReduce

CollideThreshold Once a *CollideCount* exceeds this value, the node is ignored. Effectively this is the number of DecrementIntervals a node can maintain peak transmission rate for, prior to its packets being dropped. Set to $3 \times$ CollideReduce.

ClaimThreshold Similar to CollideThreshold, once a *ClaimCount* oversteps this boundary the node is ignored. Set to $3 \times$ ClaimReduce.

CollisionThreshold Similar to CollideThreshold, once a *CollisionCount* oversteps this boundary the node is ignored. It has been set to $3 \times$ CollisionReduce.

6.10 Compliance to our requirements

Now that the DAOMAP architecture and mechanism have been presented, it is a good idea to ascertain how well it fits in with the requirements listed in Section 5.4.

Dynamic allocation DAOMAP provides for completely dynamic address allocation. No addresses are statically assigned, save for the single address which is used for message exchange. This goal has been achieved.

Distributed structure The core enabler of DAOMAP is the unicast prefix based IPv6 multicast address. This address format allows DAOMAP to exists without any hierarchy of address servers. The distributed structure is scalable, since it only has to be deployed at the network edge. This goal has been achieved.

Integrable Because the design specifically chose UDP for packet delivery, any system which supports UDP over IPv6 can run a DAA. The running of such a DAA does not affect other services, nor places heavy burdens on administrators save for manually keying the IPSec rules. This goal has been achieved.

Lifetime limitation This feature is present in DAOMAP, although enforcement thereof in applications is not. This goal has been mostly achieved.

Secure Two positions of attack on the group were considered, external and internal. The use of IPSec nullifies most external attacks, and those which are not covered are not specific to DAOMAP. Internal attacks could be vastly reduced by the addition of source authentication, which it not available at present. However the definition and positioning of three thresholds, each designed to prevent a specific attack, enforces the need for legal traffic to outlast illegal traffic. The goal has been mostly achieved.

Fair-use enforcement By defining thresholds for messages and calculating average number of groups per DAA, fair-use of the address space and network resources is enforced. This goal has been achieved.

Robustness DAOMAP is robust in that re-sends are built-in to realise a high level of probability that packets get transmitted correctly. When address collisions occur, the protocol can resolve such cases. When a node fails, other nodes are still able to allocate addresses independently of the failed node. This goal has been achieved.

Address collision limitation Another cornerstone of DAOMAP, collision prediction and limitation is provided through a deterministic address generator. This goal has been achieved. A side issue here is that only $\sqrt{n}$ of the address space is utilised. Future work should focus on utilisation of the entire address space.

To summarise, six of the eight targets have been attained, whilst two were mostly attained. Introduction of source authentication would increase the number of requirements reached to seven. The remaining issue of lifetime enforcement is not currently possible.

6.11 Conclusion

This lengthy chapter has introduced the architecture behind the author's proposed solution to the malloc problem, DAOMAP. Four components, the data store, address generator, network and client communication modules were defined and their behaviour described.

Six algorithms providing in-depth behavioural details were listed, along with their descriptions. The functions required for the correct operation of the algorithms were enumerated and their arguments and returns values loosely described.

The subject of securing DAOMAP messages was examined, and concluded with the application of IPSec to DAOMAP.

Lastly, DAOMAP was analysed with respect to the requirements described in Section 5.4 and it was found that DAOMAP achieved six of the eight goals named, with future enhancements possibly leading to seven of the eight fully achieved. The remaining requirements were mostly achieved, and are currently beyond the control of DAOMAP.

In the next chapter the implementation of DAOMAP is presented.

Chapter 7

Implementation

7.1 Introduction

In this chapter, the author's implementation of DAOMAP is documented and benchmarked.

We begin by describing the internal structures used, and how they interact. Flowcharts are used to depict the nature of execution, and provide insight into the program design. Next, benchmarks illustrate how the system functions under normal conditions, and the chapter is concluded with simulations of identified attack scenarios.

7.2 Platform

Before the details of the implementation are discussed, let us first examine the tools used to create it. The DAOMAP prototype consists of more than 3000 lines of code, written in the C programming language [80]. While crafted for Linux, it will run on most Unix-type operating systems with minor modifications in the networking code. Specifically, the 2.6 series of the Linux kernel was used, as this has the best support for IPSec and IPv6 amongst the various Linux releases. The GNU[81] toolchain was used to build the software, composing of `make`, `gcc` (the GNU Compiler Collection), `as` and `ld`.

Libraries used include the standard libraries plus the pthread, math, Berkeley DB and OpenSSL libraries.

The application consists of two parts: the first runs as a daemon which processes DAOMAP network traffic and allocates addresses to applications; whilst the second is a library which other applications can link against. This library provides the client APIs mentioned earlier which allow these applications to request addresses. The API calls connect to the daemon over a Unix domain socket, as will soon be seen. This prototype only supports one running `daa` per node, but is easily extensible to multiple interfaces.

The daemon is called `daa` and the library `libdaomap`.

7.3 Internal Components

7.3.1 Data Store

The most common operation performed on the data store are lookups. A linear search through the table would have been inefficient, and so version 4 of the Berkeley DB [82] was selected to provide data storage functionality. From their website, it is "*the most widely-used embedded data management software in the world, is open source and runs on all major operating systems, including embedded Linux, Linux, MacOS X, QNX, UNIX, VxWorks and Windows.*" Its small footprint (less than 500k) makes it ideal for use by DAOMAP, and the API provides calls for direct access to data, without the need for a separately executing database engine and query layer [82].

Two separate databases are kept, for the ADDRESSES and NODES tables respectively. Each database uses a balanced tree to store data, with the key being an IPv6 address in bit format. Duplicate keys are not allowed, and a secondary index was created for the NODES table. This secondary index allows lookups to be performed on both the address of a node, as well as the next address it is probably going to attempt to allocate (Algorithm 2).

In Appendix A, a brief tutorial displaying the Berkeley DB functionality is provided.

7.3.2 Deterministic Function

As was declared previously, the MD4 algorithm was selected as the deterministic function. The implementation provided by the OpenSSL[1] [83] library was utilised.

7.3.3 Network Module

The network module makes use of the standard BSD compatible socket interface [84, p. 57] available under Linux. A UDP over IPv6 socket is created, and the kernel is then informed via the `setsockopt(2)` call that the socket is a Multicast socket. Interestingly, no MLD reports are issued when the kernel is instructed to join the `ff02::1` group, in accordance with the MLDv2 specification. This is a special case, as all IPv6 capable nodes must join this group on network initialisation.

While host Endianness is accounted for with the correct byte-swapping procedures, the code used to determine the prefix length is specific to Linux. When the software is ported to other OSes, only that code portion will require modification.

IPv6 socket code is provided in Appendix B.

7.3.4 Client API

Interaction between the daemon and client is achieved with the use of a Unix domain socket [84, p. 373] which the daemon monitors. The client calls a stub in `libdaomap` which connects to the Unix domain socket and passes an instruction over the socket. This instruction is either to allocate an address or update the lifetime of an allocated address. If a client wishes to release an address, the lifetime is merely updated to zero, and the address will then expire.

Since all clients on a single machine use the same Unix domain socket to communicate with the daemon, some kind of authentication is required

[1] OpenSSL provides SSL/TLS functionality, as well as general purpose cryptographic functionality.

when lifetime update operations are executed, otherwise malicious clients could force the release of addresses owned by other applications. Unix domain sockets provide for sender credentials; that is, whenever a packet is received over a Unix domain socket which has the `SO_PASSCRED` [85] option set, the user and group identifiers as well as process identifiers are passed as ancillary data to the message. The daemon records which process an address was allocated to, and then any operations concerning that address are only allowed from the owning process. These credentials are filled in by the kernel, so applications cannot trick the daemon by spoofing credentials.

An example on how to use sender credential passing is provided in Appendix C.

7.4 Operational Flowcharts

Each `daa` has two threads of execution: the main controlling thread which communicates with clients, generates addresses, sends and receives from the network and queues events, and a timer thread which acts on the time-based events queued by the controlling thread. Four flowcharts are presented: one for each of the threads and two which depict sub-processes. The sub-processes are explained first, followed by the thread flowcharts.

7.4.1 Sub-processes

Figure 7.1 provides the mechanism for accessing the ADDRESSES and NODES tables. As can be observed, serial access to the data store is enforced by means of a lock. Our application uses the mutexes provided by the Linux implementation of the IEEE's POSIX thread library [86], or `libpthread` [87]. A single lock must be acquired before accessing the data store. This ensures that consistency and integrity of the data stores is maintained.

In order for the two threads of execution to communicate, they share a common area in memory. This area is protected identically to the data stores above; a mutex ensures that only one thread can ever access the memory segment at a time. Figure 7.2 displays the steps taken by a thread when

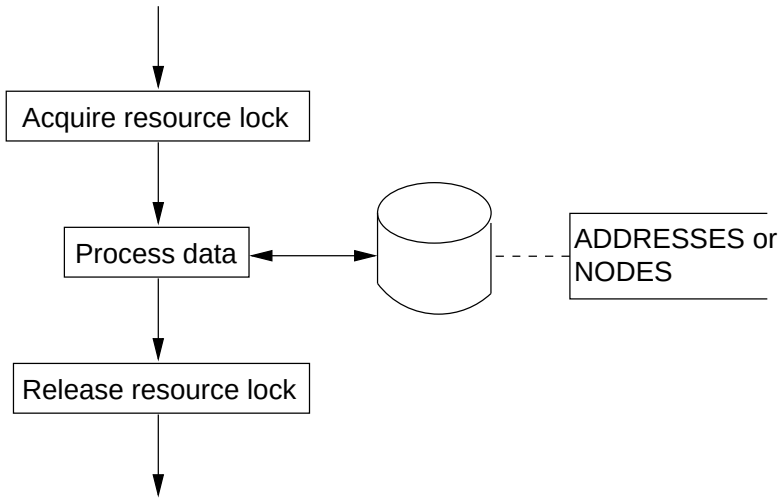


Figure 7.1: Processes A and N

accessing the shared memory (note: this memory area is *not* POSIX shared memory obtained with `shmat()` and `shmget()`, it is merely a global variable both threads have access to.)

7.4.2 Main thread

At first glance Figure 7.3 can be quite daunting; there are numerous branches with a number of steps in each as well as copious amounts of flow-lines. However the diagram need not be too intimidating; it is divided into three main components: network traffic handling, Unix domain packet handling and communication with the timer thread. These three are linked by the thick line on the left of the flowchart. Scattered throughout the flowchart are references to the sub-processes **A**, **N** and **Q**. These have already been provided in Figures 7.1 and 7.2.

Each component is handled in isolation:

Network Once a packet arrives on the network socket, it is copied from the transport layer, after the necessary IPSec transforms have been completed. A decision is then made as to whether the packet is to be accepted. This judgement is the result of Algorithm 6, which outlines the IgnoreMessage function. If the packet is allowed through, then,

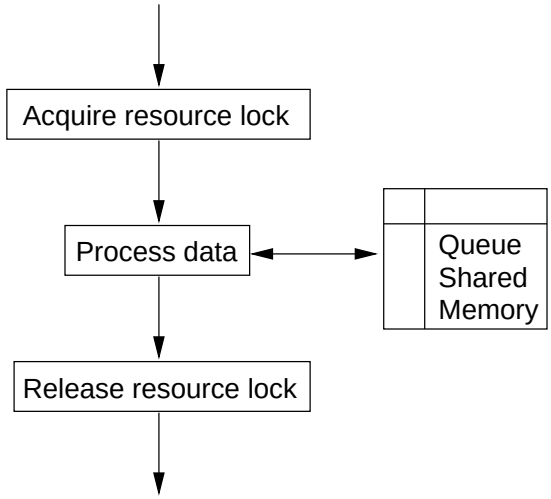


Figure 7.2: Process Q

depending on the type of packet, it is processed accordingly.

Collides for address which are in the process of being allocated force the cancellation of the allocation process for the address contained in the Collide message. If the allocation attempt was locally initiated, then a new attempt commences restart with a new address.

Claims for addresses which have not been allocated cause the address to be queued for allocation, while address clashes with allocated addresses force the generation of Collides.

Note that no IPv6 packets are sent by the main thread, all transmission is deferred to the timer thread.

Unix Two types of messages are accepted over the Unix domain socket: allocations and updates. Allocation requests cause the address generation algorithm given in Algorithm 2 to proceed. A Claim event is then placed on the timer queue. Updates are preceded by credential examination; only the process owning an address may alter its lifetime. Once authorised to do so, the lifetime is changed locally and a Claim message carrying the new lifetime is placed on the timer queue.

This system has no capacity for forcing clients to release addresses. If a process carries on using an address past expiry, the `daa` cannot interfere.

Inter-thread communication The smallest of the three components, it is only concerned with checking if an address has been allocated by the timer thread, and if so, returning the address to the correct client over its open Unix domain socket.

7.4.3 Timer thread

Unfortunately it is not as easy to discern a common path in the flowchart representing the timer thread, Figure 7.4, as it was for the main thread. Simplifying, the thread stores a queue of timer events, sorted by the absolute time of each event's timeout. When a timeout triggers, or the status of the queue changes (for example, when the main thread alters the queue), the queue is re-examined for events which might have timed out. The list of timed out events is removed from the queue, and processed. Once processing of each event has completed, the thread halts execution once again, waiting for either a timeout to occur or queue status to change.

Events are limited in the number of times they can be repeated (Claim and Collide messages are only sent three times each, for instance); and actions depend on whether events are finalised or being repeated. Of interest is the Collide branch for repeating events, where a decision is made whether to send a Collide for a non-local allocation. That decision is detailed in Algorithm 4, lines 36—46.

Importantly, when a Claim event has passed through the queue three times, it is assumed that allocation was successful since no Collides were received. The address is then added to the ADDRESSES table and considered allocated. Similarly, when a Claim is received from another node, then, as per the Claim receipt process laid out in Algorithm 4, an allocation event is placed on the timer queue. Once this event times out, then the address is added to the ADDRESSES table.

Waiting for either a timeout or status change is accomplished via the `pthread_cond_timedwait(3)` [88] call.

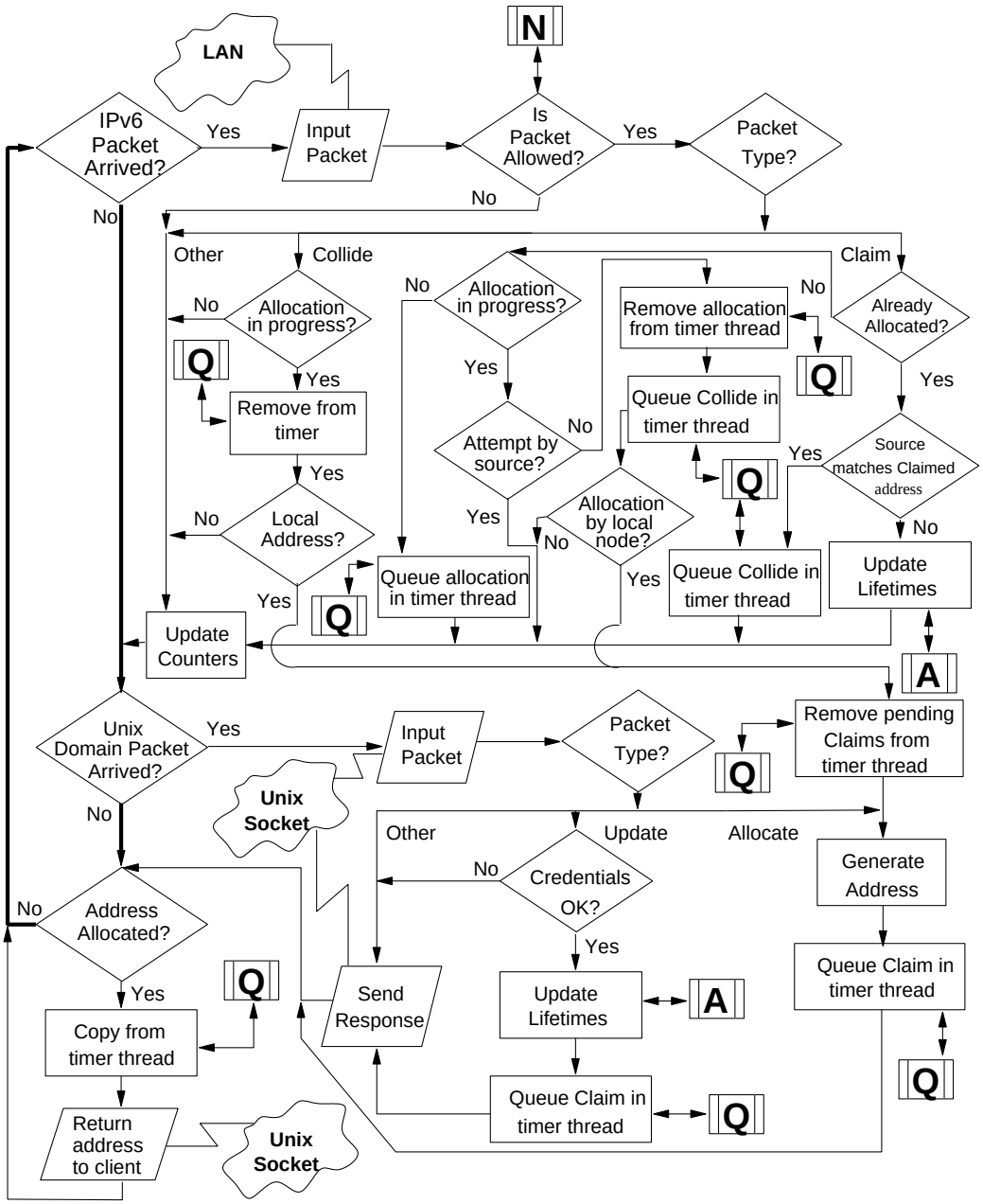


Figure 7.3: Main thread

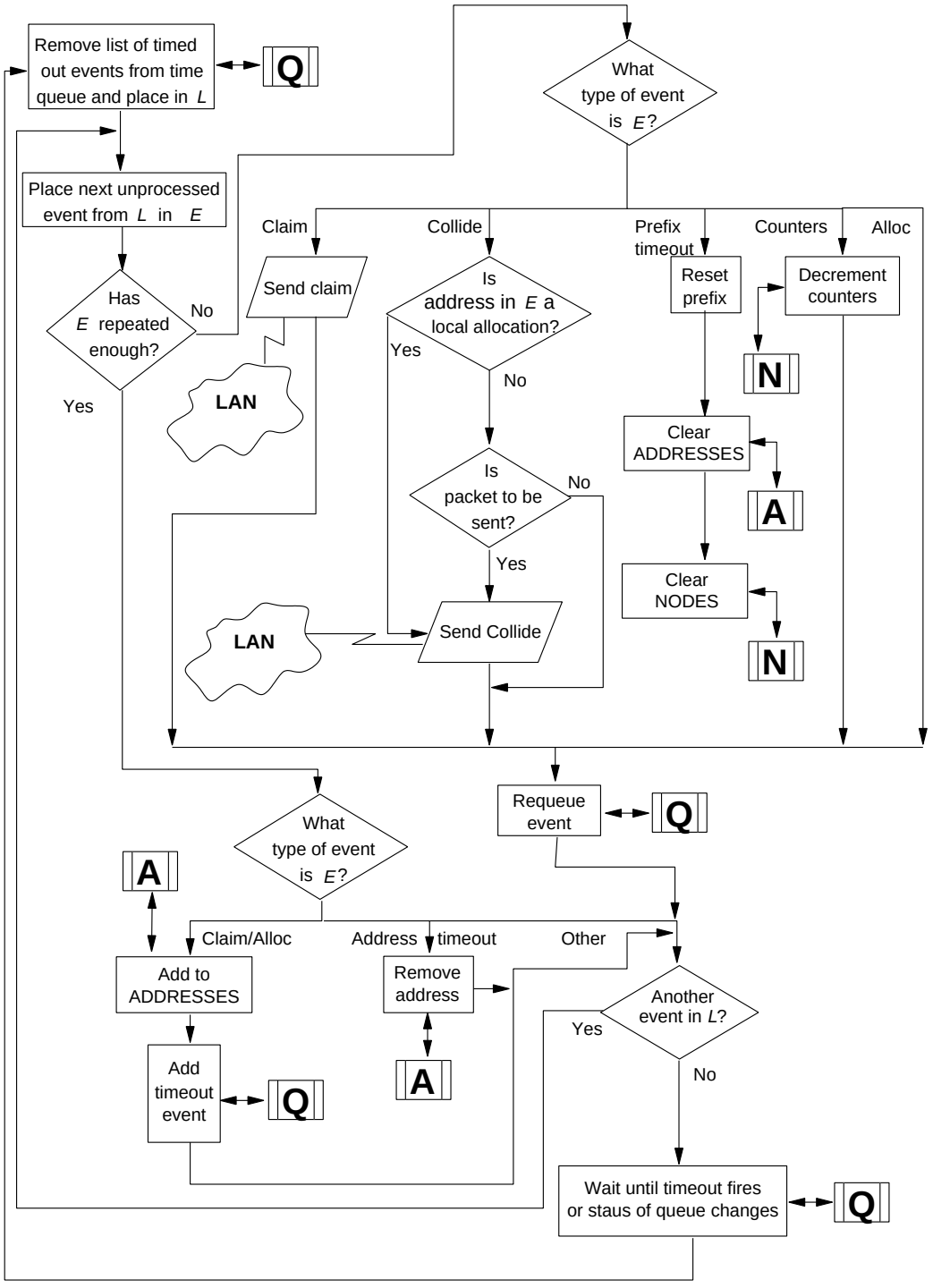


Figure 7.4: Timer thread

7.5 Benchmarking

The model and implementation have been presented to the reader, and so the viability and performance of these should be demonstrated. Such an exhibition forms a baseline which must be of sufficient standard, and against which future work will be judged.

The results obtained sought to emulate a group of normal DAAs in a LAN, by measuring various properties whilst allocation and deallocation took place.

The LAN test was conducted on an isolated 10Mb network amongst 23 nodes with a 64 bit prefix length and 32 bit group ID. Each node ran a single instance of the `daa` application, as well as a single instance of a request simulator. This request simulator's purpose was to allocate addresses as fast as possible (remembering that the DAA enforces a 1 second pause between packets, so a single allocation causes the client to wait for at least 3 seconds before another address can be requested.) The simulator issued an update on an already allocated address roughly 20% of the time, and about 60% of the allocated addresses were explicitly released by the simulator. The remaining 40% would eventually timeout.

Thus the simulator provides a conjectured[2] series of allocation, update and release requests to the DAA.

Running the experiment over days proved invaluable in tracking scalability and stability bugs. Once the `daa` application was stable, shorter experiments could be conducted. The results presented here were gathered during a three hour run of the request simulator.

The first graph, Figure 7.5, shows the ratio of allocations originating from the network to allocations locally sourced. It is a function of the number of nodes in the network, and notes the direct interest a node has in network traffic. When fewer nodes are connected, on average each node issues a greater proportion of Claims.

Another method to determine node interest is to directly plot the number

[2]The ambiguity here is unavoidable, since this has never been deployed in the real world. One can only conjecture how 'normal' usage of such a system might appear.

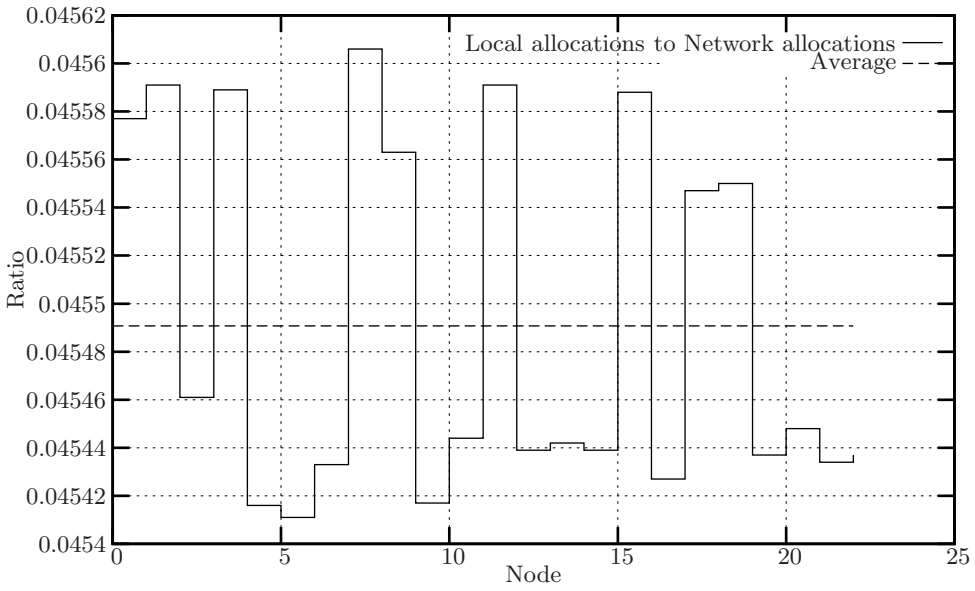


Figure 7.5: Allocation Ratio

of Claims sent versus the number of Claims received, as in Figure 7.6. Sent and received Collides were left out since in all cases those figures were zero.

The next statistic is important since it demonstrates the low likelihood of collisions. Figure 7.7 illustrates that *no* Collide messages were sent at all during the test. The significance could be somewhat mitigated in view of the fact that the test was only run over three hours, however the smallest possible group ID does provide the maximum probability of a collision. Note the figure displays *successful allocations*, not just packets sent.

Recall that addresses are generated by a deterministic function. The performance of the function is measured in Figure 7.8, where breaks from the sequence are recorded against continuations. As can be seen, *no* breaks were noted, implying that no collisions occurred. A longer simulation run would eventually produce such a clash, however for our purposes this was sufficient.

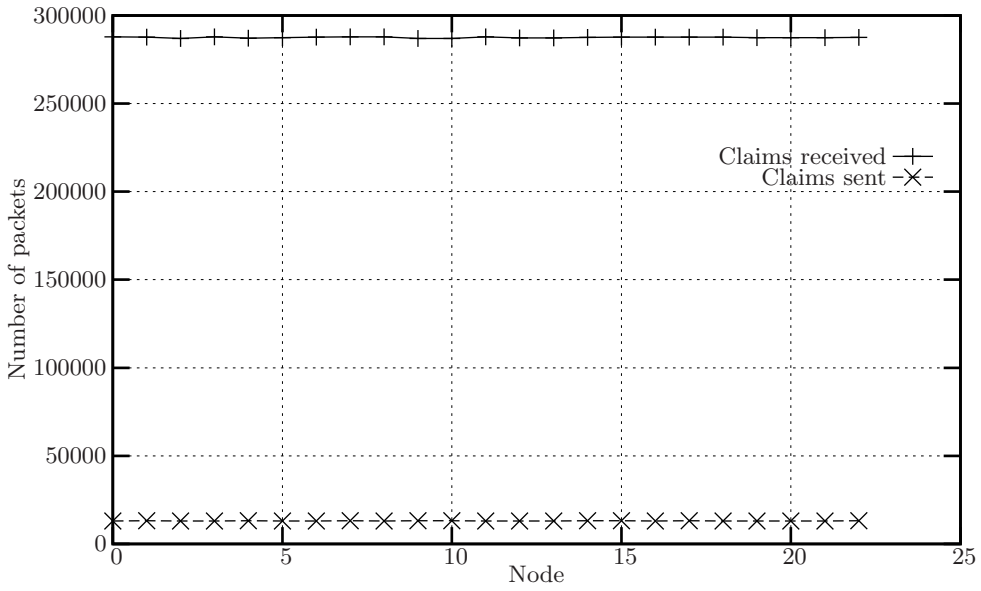


Figure 7.6: Packet origination

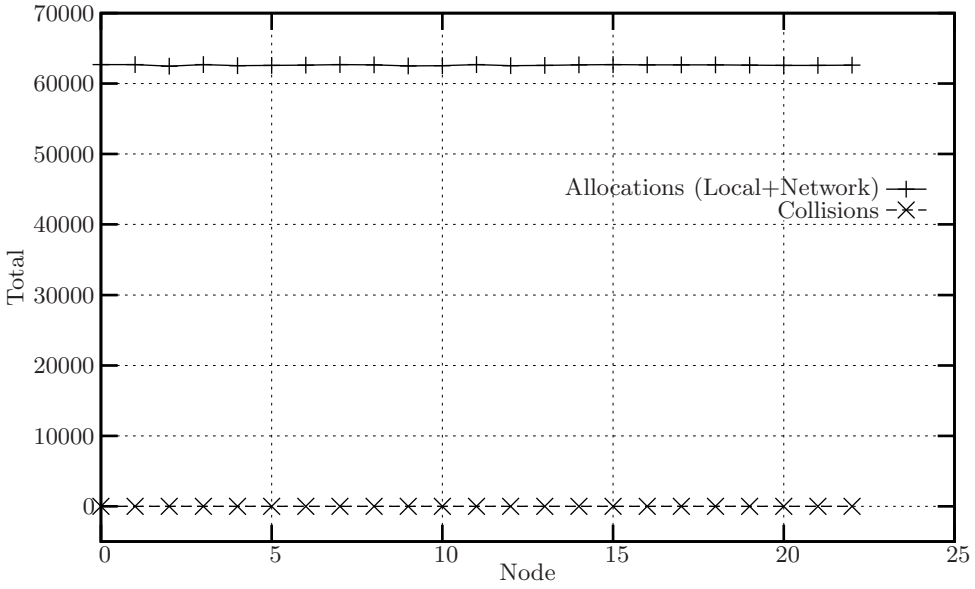


Figure 7.7: Number of allocations and collisions

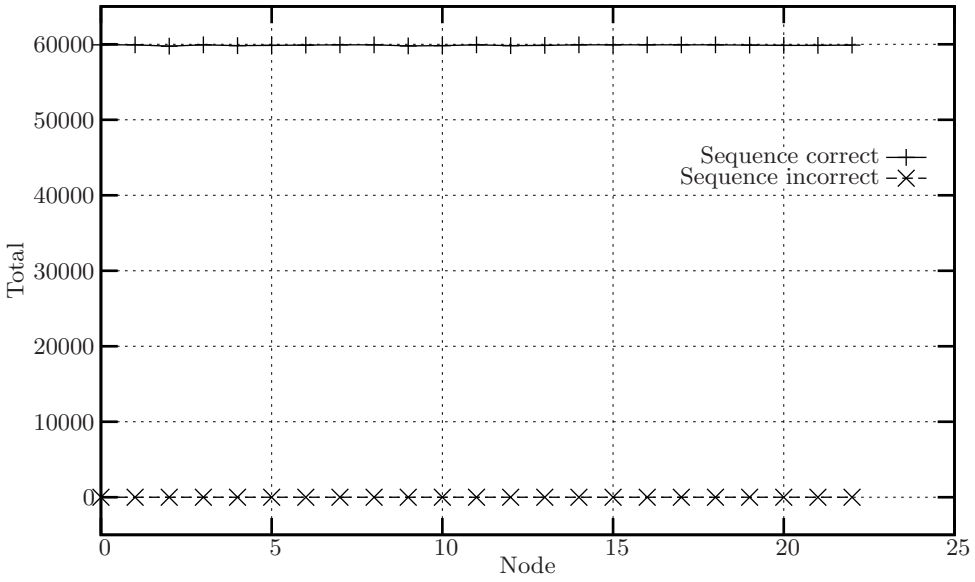


Figure 7.8: Deterministic function performance

7.6 Attack Analysis

The final results presented are traces of simulated attacks. These simulations will show how DAOMAP nodes behave in the presence of a malevolent node who has access to the group key, and is thus part of the group and can attempt to influence other members.

Six attack scenarios have been identified for analysis, and are depicted in Table 7.1. The scenarios were selected to represent all stages of DAOMAP; each scenario is the intersection between a packet type (Claim or Collide) and a stage in the allocation process (Unallocated, Allocating or Allocated, Section 6.3.2). The cells contain references to the simulation for their respective attack scenarios.

	Claim	Collide
Unallocated	Simulation 1	Simulation 2
Allocating	Simulation 3	Simulation 4
Allocated	Simulation 5	Simulation 2

Table 7.1: Attack Scenarios

Traces are presented in table form and reproduce various actions which may take place as well as the effect such actions have on the *ClaimCount*, *CollideCount* and *CollisionCount* for each node[3]. Actions may include packets being transmitted or timers decremented, and are placed in the first row. The result of an action, if any, is given below the action in the form of changes to the counts stored.

Before the simulation is explained, a notation is introduced.

1. a_i denotes an allocated address.
2. x_i denotes an unallocated address or an address in the allocating phase.
3. $x_i|a_i$ is either an allocated or unallocated address.
4. Nodes are represented by a slanted capital, such as A.
5. A set of nodes is denoted by {}.
6. The packet type is either $\mathrm{Claim_k(x_i|a_i)}$ or $\mathrm{Collide_k(x_i|a_i)}$ where k is the packet sequence number and the address $x_i|a_i$ is carried by the message.
7. The *node* $\rightarrow$ *packet* expression indicates that *node* sent *packet* to the group, where *node* is the node name and *packet* is a packet type. It is assumed that network errors are absent.
8. $\ominus$ indicates that a decrement event has passed, that is each count in the NODES table has been decremented by the correct amount.

The general scenario used in the simulations is this: the group has four nodes, A, B, C, and D. A assumes the role of the attacker in all scenarios. The traces show counts from the NODES table of D, a member of the group. Since the group size has been fixed at four, the calculated values for the parameters affecting the IGNOREMESSAGE function described in Section 6.9.5 are given in Table 7.2. These values have been rounded up to the nearest integer, to facilitate clearer observation, and remain constant throughout all simulations.

To understand the traces, the reader should first identify the scenario being played out as well as the nodes involved. Then, starting with the left-

[3]In the actual traces, shortened versions of these count names are used, respectively *Claims*, *Collides* and *Collisions*

most event, digest an event by determining what type of action occurred and what the consequences were. Consequences are placed directly below the action which caused them.

In three of the five simulations, some of the nodes are left out of the traces. The reason for this is that counts for those nodes do not change. Only when an action invokes a change in a value, is the new value printed. Once a count has exceeded its threshold, the node is then ignored and the trace halts.

RESPONSE_RATIO =	0.360
COLLIDEREDUCE =	1.443 or 2
CLAIMREDUCE =	2.164 or 3
COLLISIONREDUCE =	1.443 or 2
COLLIDETHRESHOLD =	4.328 or 5
CLAIMTHRESHOLD =	6.492 or 7
COLLISIONTHRESHOLD =	4.328 or 5

Table 7.2: Calculated Parameter Values

7.6.1 Flooding Attacks on Unallocated Addresses

The first two scenarios to be dealt with simulate trivial flooding attacks where the attacker, A, attempts to flood the group with many Claim messages for unallocated address, or Collide messages containing allocated or unallocated addresses.

Simulation 1 shows how a node issuing Claims is ignored since *Claims* for A rises above CLAIMTHRESHOLD given in Table 7.2. Note the decrement action and the effect on the *Claims* count for A.

Similarly, when A issues Collides for unallocated or allocated addresses, Simulation 2 shows how A is also ignored. This trace applies to both allocated and unallocated addresses, since according to the DAOMAP specification other nodes issue no responses to these Collides.

Claim/Unallocated

		$A \rightarrow \text{Claim}_1(x_1)$	$A \rightarrow \text{Claim}_1(x_2)$	$A \rightarrow \text{Claim}_1(x_3)$	$A \rightarrow \text{Claim}_1(x_4)$	$A \rightarrow \text{Claim}_1(x_5)$	$A \rightarrow \text{Claim}_1(x_6)$	⏀	$A \rightarrow \text{Claim}_1(x_7)$	$A \rightarrow \text{Claim}_1(x_8)$	$A \rightarrow \text{Claim}_1(x_9)$	$A \rightarrow \text{Claim}_1(x_{10})$	A **Ignored**
A	*Claims*	1	2	3	4	5	6	3	4	5	6	7	
	Collides												
	Collisions												

Simulation 1: A issues Claims for unallocated addresses, A is eventually ignored.

Collide/Unallocated or Allocated

		$A \rightarrow \text{Collide}_1(x_1 \mid a_1)$	$A \rightarrow \text{Collide}_1(x_2 \mid a_2)$	$A \rightarrow \text{Collide}_1(x_3 \mid a_3)$	$A \rightarrow \text{Collide}_1(x_4 \mid a_4)$	⏀	$A \rightarrow \text{Collide}_1(x_5 \mid a_5)$	$A \rightarrow \text{Collide}_1(x_6 \mid a_6)$	$A \rightarrow \text{Collide}_1(x_7 \mid a_7)$	A **Ignored**
A	*Claims*									
	Collides	1	2	3	4	2	3	4	5	
	Collisions									

Simulation 2: A issues Collides for addresses, A is eventually ignored.

7.6.2 Attacks in the Allocating Phase

Addresses which are in the process of being allocated are termed 'allocating' (Section 6.3.2). Attacks on allocating addresses can be launched with either Claim or Collide messages; each has a different effect.

In Simulation 3, B is trying to allocate an address. A responds with Claim messages for the same address. Without thresholds, A could continue this course of action indefinitely and deny B from ever completing an allocation.

When C and D detect duplicate claims, they issue Collides. Note the random transmission of Collides; whether or not a node emits a Collide is

governed by the RESPONSE_RATIO (Section 6.9.5). The result of this trace is that A is ignored since the COLLISIONTHRESHOLD is exceeded.

The Collide trace portrayed in Simulation 4 illustrates how A issues a Collide for every address B tries to allocate. Again, A is ignored, but this time the COLLIDETHRESHOLD is overstepped.

Claim/Allocating

		$B \rightarrow \text{Claim}_1(x_1)$	$A \rightarrow \text{Claim}_1(x_1)$	$C \rightarrow \text{Collide}_1(x_1)$ $B \rightarrow \text{Claim}_1(x_2)$	$A \rightarrow \text{Claim}_1(x_2)$	$B \rightarrow \text{Claim}_1(x_3)$	$A \rightarrow \text{Claim}_1(x_3)$	$D \rightarrow \text{Collide}_1(x_3)$ $B \rightarrow \text{Claim}_1(x_4)$	$A \rightarrow \text{Claim}_1(x_4)$	$D \rightarrow \text{Collide}_1(x_4)$ $B \rightarrow \text{Claim}_1(x_5)$	$A \rightarrow \text{Claim}_1(x_5)$	A **Ignored**
A	*Claims*											
	Collides											
	Collisions		1		2		3		4		5	
B	*Claims*	1		2		3		4		5		
	Collides											
	Collisions											
C	*Claims*											
	Collides			1								
	Collisions											
D	*Claims*											
	Collides							1		2		
	Collisions											

Simulation 3: B tries to allocate an address, A interferes with Claims. A is eventually ignored.

7.6.3 Attacks on Allocated Addresses

It has already been stated in Section 7.6.1 that the Collide attacks on allocated and unallocated addresses are identical, and so they are not discussed further.

The attack with Claim messages generates an interesting trace, shown in Simulation 5. A attempts to claim a_1 which is already allocated to another

Collide/Allocating

		$B \rightarrow \text{Claim}_1(x_1)$	$A \rightarrow \text{Collide}_1(x_1)$	$B \rightarrow \text{Claim}_1(x_2)$	$A \rightarrow \text{Collide}_1(x_2)$	$B \rightarrow \text{Claim}_1(x_3)$	$A \rightarrow \text{Collide}_1(x_3)$	$B \rightarrow \text{Claim}_1(x_4)$	$A \rightarrow \text{Collide}_1(x_4)$	$B \rightarrow \text{Claim}_1(x_5)$	$A \rightarrow \text{Collide}_1(x_5)$	A **Ignored**
A	*Claims*											
	Collides		1		2		3		4		5	
	Collisions											
B	*Claims*	1		2		3		4		5		
	Collides											
	Collisions											

Simulation 4: B tries to allocate an address, A interferes with Collides. A is eventually ignored.

node. B and C respond with Collides. A continues to issue Claims for allocated addresses until its *Claims* rises above ClaimThreshold.

7.6.4 Scenario Summary

After examining the six attack scenarios identified earlier by means of traced simulations, it appears that the thresholds and decrements defined previously are satisfactory. In every scenario, the attacker was ignored before any legitimate node; in fact no legitimate nodes were forced into isolation at all.

Further, isolation of the attacker occurred well before counts for legal nodes ever approached any of the thresholds. We thus conclude that the parameters work well and protect nodes from the identified attack scenarios.

Claim/Allocated	A			B			C			D		
	Claims	*Collides*	*Collisions*	*Claims*	*Collides*	*Collisions*	*Claims*	*Collides*	*Collisions*	*Claims*	*Collides*	*Collisions*
$A \rightarrow \text{Claim}_1(\text{a}_1)$	1											
$\{B, C\} \rightarrow \text{Collide}_1(\text{a}_1)$					1			1				
$A \rightarrow \text{Claim}_1(\text{a}_2)$	2											
$C \rightarrow \text{Collide}_1(\text{a}_2)$								2				
$A \rightarrow \text{Claim}_1(\text{a}_3)$	3											
$A \rightarrow \text{Claim}_1(\text{a}_4)$	4											
$D \rightarrow \text{Collide}_1(\text{a}_4)$											1	
$A \rightarrow \text{Claim}_1(\text{a}_5)$	5											
$D \rightarrow \text{Collide}_1(\text{a}_5)$					2							
$\oplus$	2				0			0			0	
$A \rightarrow \text{Claim}_1(\text{a}_6)$	3											
$C \rightarrow \text{Collide}_2(\text{a}_1)$								1				
$B \rightarrow \text{Collide}_1(\text{a}_6)$					1							
$A \rightarrow \text{Claim}_1(\text{a}_7)$	4										1	
$C \rightarrow \text{Collide}_2(\text{a}_2)$								2				
$\{B, D\} \rightarrow \text{Collide}_1(\text{a}_7)$					2						2	
$A \rightarrow \text{Claim}_1(\text{a}_8)$	5											
$D \rightarrow \text{Collide}_1(\text{a}_8)$											3	
$B \rightarrow \text{Collide}_2(\text{a}_3)$					3							
$A \rightarrow \text{Claim}_1(\text{a}_9)$	6											
$C \rightarrow \text{Collide}_2(\text{a}_5)$								3				
$A \rightarrow \text{Claim}_1(\text{a}_{10})$	7											
A **Ignored**												

Simulation 5: A tries to allocate addresses which are already allocated. The other nodes reply with Collides. A is eventually ignored.

7.7 Conclusion

The DAOMAP implementation described in the chapter, `daa`, combined with its client-side library `libdaomap` provide applications with globally unique multicast addresses.

The platform on which the suite runs was described, followed by a brief listing of requirements in terms of other software. Each internal component (Data Store, Deterministic Function, Network Module, Client API) was analysed in terms of their functioning and interaction. A quick recap on each component's most salient point is helpful: the Data Store uses the Berkeley DB, the Deterministic Function is provided by OpenSSL's MD4() function, the Network Module is almost platform-independent and the Unix domain sockets used by the Client API support credential passing allowing for secured address lifetime manipulation.

Flowcharts illustrating various processes were provided, with the two major charts showing execution of the main and timer threads.

Finally the results of test runs and simulations were presented, where legal traffic and attacks were examined. Amongst the legal traffic, the deterministic function appeared to operate well since no sequence breaks or Collides were seen. However, lengthier tests may be required in order to fully validate this. After conducting the attack simulations, it was concluded that the rate limiting protection provided by the IGNOREMESSAGE() function works successfully.

Chapter 8

Conclusion

The work is now complete; what remains is to summarise the contents of this dissertation, and to point to possible future research which may influence multicast address allocation.

8.1 Summary

In the opening chapter it was stated that although multicasting has promised much, the delivered services at this point are still lacking. Numerous impediments exist; and the address allocation problem, which was selected as the topic for this discourse, was only one. That introduction to the subject was brief, the purpose only to whet the reader's appetite.

A major portion of the background was presented in Chapter 2, Internet Protocol Version 6. Since the developed protocol relies heavily on IPv6, a firm grounding in the underlying principles of IPv6 was needed in order to fully understand why DAOMAP is different from other allocation mechanisms. The format and construction of IPv6 addresses was dealt with extensively; as was the composition of IPv6 packets and the various transition options available.

The User Datagram Protocol was explained in Chapter 3. Its connectionless mode of operation made it ideal for use in multicast traffic, however the drawback is one of reliability.

Chapter 4 followed, and dealt with the basics behind multicasting. Three possibilities exist for multicasting across network boundaries: Router-dependent Multicast, Application-layer Multicast, and Hybrids. The first requires support in devices which connect networks; such support allows these devices to correctly transmit multicast packets to only those networks or hosts which have expressed interest in that multicast traffic. The most prominent example of this type of multicast is IP Multicast, which consists of two components: a reporting protocol which routers use to determine which groups the listeners are interested in, and an internetwork protocol for passing packets between networks.

The remaining two categories were touched on only momentarily as they do not directly influence this work; references were provided for interested parties.

The problem which the author attempted to solve was discussed at length in Chapter 5. Drawbacks and failings of current address allocation proposal were illuminated. Specifically, the 3-tier MASC-AAS-MADCAP architecture was examined and found deficient. Literature in this area revealed that prefix based address allocation was the likely direction in which address allocation would proceed. Indeed, the Unicast prefix based multicast address depicted in Chapter 2 exploits this property.

The most significant contribution of Chapter 5 was the listing and justification of eight requirements which an allocation scheme would have to satisfy. They were:

dynamic allocation The model is able to assign and reclaim addresses continually.

distributed structure A major failing of current solutions is the need for centralised services. A distributed architecture would be easier to deploy.

integrable The final product must create as few deployment problems as possible, requiring little in terms of human and machine resources.

lifetime limitation Each address must be limited in the time it can be assigned to a particular application.

security Current allocation schemes simply view security from an authentication point of view, however various attacks exist which use a non-authentication vector.

fair-use enforcement The address space is limited, and must therefore be shared out amongst the group members, without a few members devouring the address space.

robustness The model needs to handle error and failure gracefully.

address collision limitation In order to reduce network traffic and streamline the allocation process, addresses need to be chosen such that they have a high probability of being unallocated.

DAOMAP (Distributed Allocation Of Multicast Addresses Protocol) was the model presented in Chapter 6. The design goals were to produce a structure for address allocation which could assign globally unique addresses in a scalable manner. The system only assigns single addresses at a time; and it was assumed that applications will not require many addresses per second.

The model consisted of four components, namely the data store, address generation module, network and client communications modules. Each component was documented and its interactions with the environment and other components were defined by means of six algorithms:

- A basic as well as refined address generation algorithms were presented.
- A general algorithm was given covering the allocation process.
- Two algorithms related to the network communications module detailed prescribed behaviour on the receipt of Claim and Collide packets.
- The steps taken to determine when a node should be ignored were listed.

The data store and address generation modules are interesting as they allow for collision prediction; by extension it is then easier to avoid address collisions. The MD4 function was selected for use in the address generation module, after testing four separate functions.

Since security was a focus for this study and the IgnoreMessage procedure was a vital link in the security chain, factors influencing that function

were examined in-depth and relations between three thresholds were defined. Indeed, multicast security was deemed more than simply keeping attackers out the group; strategies were devised to keep control within the group distributed and to ensure that internal attackers are forced from the group before placid members.

Lastly, the foregoing chapter chronicled the prototype that was developed. Flowcharts represented the internal apparatus of the program, and the environment in which the prototype will run was described. The chapter was concluded with a series of tests and attack simulations to gauge how well the model withstands normal as well as abnormal use. This tests results demonstrated the viability of the approach followed by DAOMAP.

8.2 Limitations

Concerning the design phase of this work, the network protocol has not been formally analysed. In order for such analysis to occur, the protocol would first have to be expressed in a specification language, and analysis tools applied. Such analysis is important in ironing out small defects which may remain hidden.

The testing of the implementation was limited in both time and resources. A larger experiment would only be possible on a large IPv6 network, which was not available at test time, and ideally tests would run over weeks in order to prove stability.

8.3 Future Work

Much work still remains to be done in the area of multicasting. The topic covers diverse research subjects such as transport protocols, distributed security amongst group members, quality of service concerns and general network efficiency. With regards to address allocation, a contribution has been made in this dissertation by introducing the idea of collision avoidance, as well as in defining various parameters and relations which affect security in the group.

Possibilities of a higher level secure protocol for general application multicasting become possible since such applications could request addresses in an spontaneous manner; further the applications can have a reasonable degree of confidence that the addresses they are assigned will be reserved for their use.

Looking slightly farther afield, the standardisation of Group Secure Association Key Management Protocol (GSAKMP) by the Multicast Security group at the IETF will allow for increased levels of security. This protocol is similar to IPSec in that it provides a framework for securing communications. Importantly it allows group members to determine the source of packets in an unambiguous fashion, which, as was noted in Chapter 2, would allow for greater security within the group. However at the moment it is still in draft form [89].

At the beginning, many difficulties encountered by multicast researchers were heaped up; while the mountain may seem insurmountable to one person, chipping away one problem at a time eventually reduces the pile to ground level. Hopefully this dissertation is a reasonably sized chip from that pile.

Appendix A

The Berkeley DB

The nuts and bolts of the implementation has been delayed until now, so that the reader did not become bogged down in technicalities whilst attempting to conceive the system in its entirety. This appendix, and the subsequent two, provide tutorials on their respective subject matter; in this case the Berkeley DB.

The aim of each tutorial is to demonstrate how certain functionality may be provided, as well as render a reference for those conversant in the broader aspects of each subject, and who are looking for a specific feature. These tutorials assume a knowledge of C, and ignore error checking as an aid to brevity. This tutorial is in no way a complete reference to Berkeley DB, the interested reader is directed to [90] if that is what is desired.

Before the actual database code is examined, here are the two structures used in the ADDRESSES (`address_record_t`) and NODES (`node_record_t`) tables:

```
typedef struct {
  ip6_address_t mcast_address;
  ip6_address_t unicast_address;
  uint32_t lifetime;
  uint8_t local;
  pid_t pid;
} address_record_t;

typedef struct {
```

```
    ip6_address_t unicast_address;
    uint32_t address_count;
    uint32_t claim_count;
    uint32_t collide_count;
    uint32_t collision_count;
    ip6_address_t last_allocated_address;
    ip6_address_t next_allocated_address;
    uint32_t message_id;
    uint8_t ignore;
} node_record_t;
```

Listing A.1: DB structures

In each example below certain structures are explicitly zeroed-out. This is in line with the Berkeley DB documentation which strongly recommends such actions. The reason given is that the internal formation of the structures is not guaranteed to remain as-is, and additional fields could cause instabilities in non-cleared structures.

A.1 Database creation

Listing A.2 shows how two databases, one with an index allowing lookups on secondary keys, are created with the Berkeley DB API. Note that these API calls are applicable to version 4 of Berkeley DB.

Initially a database environment is constructed which supports locking for multiple access from within a single threaded program. Each database is then created within the environment, using a sorted, balanced tree for storage. Lastly, an index is created for NODES to allow lookups on a secondary key. The user-supplied function `get_node_address()` is used by the DB to extract a secondary key from data.

```
DB_ENV *db_env;
DB *addr_db, *node_db, *node_db_index;

db_env = NULL;
addr_db = node_db = NULL;
```

```
db_env_create(&db_env, 0);
db_env->open(db_env, NULL, DB_CREATE | DB_INIT_LOCK |
   DB_PRIVATE |  DB_INIT_MPOOL | DB_THREAD, 0);
db_create(&addr_db, db_env, 0);
addr_db->open(addr_db, NULL, NULL, NULL, DB_BTREE, DB_CREATE,
    0);
db_create(&node_db, db_env, 0);
node_db->open(node_db, NULL, NULL, NULL, DB_BTREE, DB_CREATE,
    0);
db_create(&node_db_index, db_env, 0);
node_db_index->open(node_db_index, NULL, NULL, NULL, DB_BTREE
   , DB_CREATE, 0);
node_db_index->associate(node_db, NULL, node_db_index,
   get_node_address, 0);
```

Listing A.2: DB setup

A.2 Lookups

Reading from a database requires the use of `DBT` structures, which hold information about keys and their values in the database, as well as where the information should be copied. A code snippet which reads from the ADDRESSES table is given in Listing A.3.

```
DBT key, data;

memset(&key, '\0', sizeof(DBT));
memset(&data, '\0', sizeof(DBT));

key.data = key_data;
key.size = sizeof(key_data);

data.data = dst;
data.ulen = sizeof(table_struct);
data.flags |= DB_DBT_USERMEM;

db_handle->get(db_handle, NULL, &key, &data, 0)
```

Listing A.3: DB Reading

In the above snippet, *key_data* is a pointer to the key, *dst* points to a portion of memory where the retrieved entry can be written to, *table_struct* is the structure used for a single entry in the database (either `address_record_t` or `node_record_t`) and *db_handle* is one of `addr_db` or `node_db`. The flag DB_DBT_USERMEM specifies that the memory pointed to by `data.data` has already been allocated.

Reading from the secondary index is slightly different. Instead of one key there are two; one each for the primary and secondary indices. In the example below, `pkey` is a structure holding the primary key and likewise `skey` stores the secondary key. New in this snippet is the DB_DBT_MALLOC flag which asks the database to allocate memory for that destination. Also note that the function called is `pget()` rather than `get()`.

```
DBT skey, pkey, data;

memset(&skey, '\0', sizeof(DBT));
memset(&pkey, '\0', sizeof(DBT));
memset(&data, '\0', sizeof(DBT));

skey.data = key_data;
skey.size = sizeof(key_data);

data.data = dst;
data.ulen = sizeof(table_struct);
data.flags |= DB_DBT_USERMEM;

pkey.flags |= DB_DBT_MALLOC;

db_handle->pget(index_handle, NULL, &skey, &pkey, &data, 0);
```

Listing A.4: Secondary Index Reading

A.3 Database Writing

Writing to a database is accomplished by:

```
DBT key, data;
```

```
memset(&key, '\0', sizeof(DBT));
memset(&data, '\0', sizeof(DBT));

key.data = key_data;
key.size = sizeof(key_data);

data.data = src;
data.size = sizeof(src);
db_handle->put(db_handle, NULL, &key, &data, flags)
```

Listing A.5: DB writing

where *src* points to a structure of the appropriate type and *flags* is either DB_NOOVERWRITE (don't allow duplicates) or 0 (no flags set.) Writing to a database with a secondary index automatically updates both indices, so additional steps are not required.

A.4 Entry deletion

Deleting records from the database is demonstrated in Listing A.6.

```
DBT key;

memset(&key, '\0', sizeof(DBT));

key.data = key_data;
key.size = sizeof(key_data);

db_handle->del(db_handle, NULL, &key, 0)
```

Listing A.6: Entry Deletion

A.5 Entry looping

Berkeley DB provides the programmer with a 'cursor' type for iterating through the database. Listing A.7 illustrates how successive records in the database can be processed.

```
DBC *cursor;
DBT key, data;

memset(&key, '\0', sizeof(DBT));
memset(&data, '\0', sizeof(DBT));

db_handle->cursor(db_handle, NULL, &cursor, 0);

while (cursor->c_get(cursor, &key, &data, DB_NEXT) == 0){
  .
  .
  .
}
cursor->c_close(cursor);
```

Listing A.7: DB Cursors

At each iteration, the next available key and its data from *db_handle* are placed in their respective structures.

Appendix B

IPv6 code snippets

The IPv6 specific code, as has already been mentioned, was written with portability in mind. To this end, data is converted to and from network byte order, and standard BSD-socket calls are used. The sole exception to this occurs in the code where the prefix length of an address is determined.

The 'correct' method under Linux to discover such information is to access the routing subsystem via Netlink sockets. We have taken a shortcut and simply read the information from the `/proc` filesystem, which is a virtual filesystem providing OS data.

The code example provided in Listing B.1 explains how to construct an IPv6 UDP socket which listens on address *listen_address*.

```
1  struct sockaddr_in6 sock_addr;
2  struct ipv6_mreq group_addr;
3  int hops = 1, on = 1, off = 0, if_index, group_sockfd;
4
5  group_sockfd = socket(AF_INET6, SOCK_DGRAM, 0);
6  setsockopt(group_sockfd, SOL_SOCKET, SO_REUSEADDR, &on,
       sizeof(on));
7
8  if_index = if_nametoindex(interface);
9
10 sock_addr.sin6_family = AF_INET6;
11 inet_pton(AF_INET6, listen_addr, &sock_addr.sin6_addr);
12 sock_addr.sin6_port = htons(port);
13 sock_addr.sin6_scope_id = if_index;
```

```
14
15 bind(group_sockfd, (struct sockaddr *) &sock_addr, sizeof(
       sock_addr));
16
17 inet_pton(AF_INET6, listen_addr, &group_addr.ipv6mr_multiaddr);
18 group_addr.ipv6mr_interface = if_index;
19
20 setsockopt(group_sockfd, SOL_IPV6, IPV6_MULTICAST_HOPS, (int
       *) &hops, sizeof(hops))
21 setsockopt(group_sockfd, SOL_IPV6, IPV6_MULTICAST_LOOP, (
       unsigned int *) &loop, sizeof(loop));
22 setsockopt(group_sockfd, IPPROTO_IPV6, IPV6_JOIN_GROUP, (
       struct ipv6_mreq *) &group_addr, sizeof(group_addr));
```

Listing B.1: Creating an IPv6 Socket

Lines 5 – 15 create a socket and bind it to the group address. Importantly, a scope id *must* be assigned; the interface index is used in this case[1]. The remaining lines fill in a structure containing information about the multicast address the application will listen on, set the maximum number of hops the packet will traverse, turn off local delivery of packets which originate locally and finally instruct the kernel to join the multicast group.

Transmitting and receiving on the socket then occurs as per usual.

[1]The interface index is simply a unique number assigned to each network interface.

Appendix C

Unix Domain Sockets

Probably the most interesting portion of the Unix socket code involves the appending of the end-point processes' credentials to the messages as they travel over the socket. The example listings first demonstrate how to enable such credential passing, and then proceed to use the credentials.

```
1  #define UNIX_SOCK_PATH "/tmp/daa"
2
3  struct sockaddr_un unix_d_server;
4  int unix_sockfd;
5  const int toggle_on = 1;
6
7  unix_sockfd = socket(AF_LOCAL, SOCK_STREAM, 0);
8
9  unlink(UNIX_SOCK_PATH);
10
11 memset(&unix_d_server, '\0', sizeof(unix_d_server));
12
13 unix_d_server.sun_family = AF_LOCAL;
14 strncpy(unix_d_server.sun_path, UNIX_SOCK_PATH, strlen(
       UNIX_SOCK_PATH));
15
16 bind(unix_sockfd, (struct sockaddr *) &unix_d_server, sizeof(
       struct sockaddr_un));
17
18 setsockopt(unix_sockfd, SOL_SOCKET , SO_PASSCRED, &toggle_on,
        sizeof(toggle_on));
```

```
19
20 listen(unix_sockfd, LISTENQ);
```

Listing C.1: Creating an Unix Domain Socket with Credential Passing

A Unix domain socket's end-point address is a file; in Listing C.1, the file is defined as `/tmp/daa`. Before the socket can be created this file must not exist, and hence the attempt to delete it with `unlink(2)`.

Once the socket has been constructed, clients connect has usual. The credential passing is completely transparent to them, as this data is filled in by the kernel. On the server side, credentials are retrieved as follows:

```
1  char buf[100];
2  struct ucred creds;
3  struct msghdr msg_hdr;
4  struct cmsghdr *cmsg_hdr;
5  struct iovec iov[1];
6
7  union {
8    struct cmsghdr cm;
9    char control[CMSG_SPACE(sizeof(struct ucred))];
10 } control_un;
11
12 msg_hdr.msg_control = control_un.control;
13 msg_hdr.msg_controllen = sizeof(control_un.control);
14 msg_hdr.msg_name = NULL;
15 msg_hdr.msg_namelen = 0;
16
17 iov[0].iov_base = buf;
18 iov[0].iov_len = sizeof(buf);
19 msg_hdr.msg_iov = iov;
20 msg_hdr.msg_iovlen = 1;
21
22 recvmsg(sockfd, &msg_hdr, 0);
23
24 if (msg_hdr.msg_controllen > sizeof(struct cmsghdr)){
25   cmsg_hdr = CMSG_FIRSTHDR(&msg_hdr);
26   if (cmsg_hdr->cmsg_len == CMSG_LEN(sizeof(struct ucred)) &&
        cmsg_hdr->cmsg_level == SOL_SOCKET && cmsg_hdr->
       cmsg_type == SCM_CREDENTIALS)
```

```
27        memcpy(&creds, CMSG_DATA(cmsg_hdr), sizeof(struct ucred
              ));
28 }
```

Listing C.2: Receiving Unix Domain Message with Credential Passing

From the `unix(7)` manpage, the structure `ucred` has these fields:

```
1 struct ucred {
2     pid_t  pid;  /* process id of the sending process */
3     uid_t  uid;  /* user id of the sending process */
4     gid_t  gid;  /* group id of the sending process */
5 };
```

Glossary of Abbreviations

AAS	Address Allocation Server
API	Application Program Interface
ASM	Any Source Multicast
BGMP	Border Gateway Multicast Protocol
BSD	Berkeley Software Distribution
BTP	Banana Tree Protocol
CATNIP	Common Architecture for the Internet
CBT	Core Based Trees
CIDR	Classless Inter-domain Routing
DAA	Distributed Address Allocator
DAOMAP	Distributed Allocation Of Multicast Addresses Protocol
DHCP	Dynamic Host Configuration Protocol
DM	Designated Member
DNS	Domain Name System
DS	Differentiated Service
DVMRP	Distance Vector Multicast Routing Protocol
ECN	Explicit Congestion Notice
GNU	GNU's Not UNIX
GSAKMP	Group Secure Association Key Management Protocol
HMTP	Host Multicast Tree Protocol

IANA	Internet Assigned Numbers Authority
ICMPv6	Internet Control Message Protocol for IPv6
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISATAP	Intra-Site Automatic Tunnel Addressing Protocol
ISP	Internet Sevice Provider
LAN	Local Area Network
MAC	Media Access Control
MADCAP	Multicast Address Dynamic Client Allocation Protocol
MASC	Multicast Address Set Claim
MD4	Message Digest 4
MD5	Message Digest 5
MBONE	Multicast Backbone
MLDv2	Multicast Listener Discovery version 2
OSPF	Open Shortest Path First
PIP	P Internet Protocol
POSIX	Portable Operating System Interface
RFC	Request For Comment
RP	Rendezvous Point
RPB	Reverse Path Broadcasting
RPF	Reverse Path Flooding
RPM	Reverse Path Multicasting
SFM	Source Filtered Multicast

SIP	Simple Internet Protocol
SIPP	Simple Internet Protocol Plus
SSM	Source Specific Multicast
TCP	Transmission Control Protocol
TENET	Tertiary Education Network
TUBA	TCP and UDP with Bigger Addresses
UDP	User Datagram Protocol
YOID	Your Own Internet Distribution

Bibliography

[1] S Deering. Host Extensions for IP Multicasting. RFC 1112, Internet Engineering Task Force, August 1989. 1, 30, 34, 35, 37

[2] T D C Little, G Ahanger, R J Folz, J F Gibbon, F W Reeve, D H Schelleng, and D Venkatesh. A digital on-demand video service supporting content-based queries. In *Proceedings of the first ACM international conference on Multimedia*, pages 427–436. ACM Press, 1993. 1

[3] L H Ngoh and T P Hopkins. Multicast communication facilities for distributed multimedia information systems. In *Second IEE National Conference on Telecommunications*, 1989. 1

[4] IP multicasting - the forgotten holy grail? http://dataweek.co.za/Article.ASP?pklArticleID=2807&pklIssueID=441. Last accessed 12 October 2004. 2

[5] D Zappala, V Lo, and C GauthierDickey. The multicast address allocation problem: theory and practice. *Computer Networks*, 45(1):55–73, 2004. 2, 54, 62, 63, 66, 69, 72

[6] J Postel. Internet Protocol. RFC 791, Internet Engineering Task Force, September 1981. 6

[7] D Cohen. ON HOLY WARS AND A PLEA FOR PEACE. IEN 137, Internet Engineering Task Force, April 1980. 6, 90

[8] C Huitema. *Routing in the Internet*. Prentice Hall PTR, 1995. 8, 9, 10

[9] T Dixon. Comparison of Proposals for Next Version of IP. RFC 1454, Internet Engineering Task Force, May 1993. 8, 9, 10

[10] R Carlson and D Ficarella. Six Virtual Inches to the Left: The Problem with IPng. RFC 1705, Internet Engineering Task Force, October 1994. 9

[11] S Bradner and A Mankin. The Recommendation for the IP Next Generation Protocol. RFC 1752, Internet Engineering Task Force, January 1995. 9

[12] M McGovern and R Ullmann. CATNIP: Common Architecture for the Internet. RFC 1707, Internet Engineering Task Force, October 1994. 9

[13] R Hinden. Simple Internet Protocol Plus White Paper. RFC 1710, Internet Engineering Task Force, October 1994. 9

[14] P Francis. Pip Near-term Architecture. RFC 1621, Internet Engineering Task Force, May 1994. 10

[15] R Callon. TCP and UDP with Bigger Addresses (TUBA), A Simple Proposal for Internet Addressing and Routing. RFC 1347, Internet Engineering Task Force, June 1992. 10

[16] D Marlow. Host Group Extensions for CLNP Multicasting. RFC 1768, Internet Engineering Task Force, March 1995. 10

[17] S Deering and R Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, Internet Engineering Task Force, December 1998. 11, 20, 22, 29, 35

[18] S Hagen. *IPv6 Essentials*. O'Reilly & Associates, Inc., 2002. 11, 13

[19] R Hinden and S Deering. Internet Protocol Version 6 (IPv6) Addressing Architecture. RFC 3513, Internet Engineering Task Force, April 2003. 11, 12, 13, 14, 15, 17, 23

[20] B Haberman and D Thaler. Unicast-Prefix-based IPv6 Multicast Addresses. RFC 3306, Internet Engineering Task Force, August 2002. 16, 17

[21] S Thomson and T Narten. IPv6 Stateless Address Autoconfiguration. RFC 2462, Internet Engineering Task Force, December 1998. 18

[22] R Droms (Ed.). Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 3315, Internet Engineering Task Force, July 2003. 18

[23] S Blake, D Black, M Carlson, E Davies, Z Wang, and W Weiss. An Architecture for Differentiated Services. RFC 2475, Internet Engineering Task Force, December 1998. 21

[24] K Nichols, S Blake, F Baker, and D Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, Internet Engineering Task Force, December 1998. 21

[25] K Ramakrishnan, S Floyd, and D Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, Internet Engineering Task Force, September 2001. 21

[26] J Rajahalme, A Conta, B Carpenter, and S Deering. IPv6 Flow Label Specification. RFC 3697, Internet Engineering Task Force, March 2004. 21

[27] TENET Home Page. http://www.tenet.ac.za. Last accessed 17 October 2004. 23

[28] ARIN Home Page. http://www.arin.net. Last accessed 17 October 2004. 23

[29] R Gilligan and E Nordmark. Transition Mechanisms for IPv6 Hosts and Routers. RFC 2893, Internet Engineering Task Force, August 2000. 24

[30] B Carpenter and K Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056, Internet Engineering Task Force, February 2001. 25

[31] B Carpenter and C Jung. Transmission of IPv6 over IPv4 Domains without Explicit Tunnels. RFC 2529, Internet Engineering Task Force, March 1999. 25

[32] F Templin, T Gleeson, M Talwar, and D Thaler. Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). Ietf draft, Internet Engineering Task Force, September 2003. 26

[33] D Comer. *Computer Networks and Internets*. Prentice Hall, 2 edition, 1999. 27, 48

[34] J Postel. User Datagram Protocol. RFC 768, Internet Engineering Task Force, August 1980. 28, 29

[35] W Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, 1994. ISBN 0 201 63346 9. 30

[36] S Deering. Multicast routing in internetworks and extended lans. In *Symposium proceedings on Communications architectures and protocols*. ACM Press, 1988. 33, 35, 45, 46

[37] B Zhang, S Jamin, and L Zhang. Host multicast: A framework for delivering multicast to end users. In *IEEE Infocom*, June 2002. 34, 52, 55, 56, 57

[38] B Cain, S Deering, I Kouvelas, B Fenner, and A Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, Internet Engineering Task Force, October 2002. 34

[39] D Waitzman, C Partridge, and S Deering. Distance Vector Multicast Routing Protocol. RFC 1075, Internet Engineering Task Force, November 1988. 34, 48

[40] D Estrin, D Farinacci, A Helmy, D Thaler, S Deering, M Handley, V Jacobson, C Liu, P Sharma, and L Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. RFC 2362, Internet Engineering Task Force, June 1998. 34, 52

[41] J Moy. Multicast Extensions to OSPF. RFC 1584, Internet Engineering Task Force, March 1994. 34

[42] S Ratnasamy, M Handley, R Karp, and S Shenker. Application-level multicast using content-addressable networks. *Lecture Notes in Computer Science*, 2233:14–??, 2001. 34, 55

[43] D Helder and S Jamin. Banana Tree Protocol, an End-host Multicast Protocol. Technical Report TR-429-00, University of Michigan, July 2000. 34, 52, 55, 56

[44] Y Chu, S Rao, and H Zhang. A Case for End System Multicast. In *ACM SIGMETRICS 2000*. ACM, June 2000. 34, 52, 55

[45] R Vida and L Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810, Internet Engineering Task Force, June 2004. 36, 41

[46] H Holbrook and B Cain. Source-Specific Multicast for IP. Ietf draft, Internet Engineering Task Force, October 2003. 36, 53

[47] M Crawford. Transmission of IPv6 Packets over Ethernet Networks. RFC 2464, Internet Engineering Task Force, December 1998. 38

[48] A Conta and S Deering. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 2463, Internet Engineering Task Force, December 1998. 42

[49] P A Laplante, editor. *Dictionary of Computer Science, Engineering and Technology*. Lewis Publishers Inc., 2000. 44

[50] Mohammad Banikazemi. IP Multicasting: Concepts, Algorithms, and Protocols. Last accessed 11 August 2004. 46, 47

[51] B Forouzan. *Data Communications and Networking*. McGraw Hill, 2 edition, 2001. 47, 48

[52] K Savetz, N Randall, and Y Lepage. *MBONE: Multicasting Tomorrow's Internet*. IDG, 1996. 48

[53] T Ballardie, P Francis, and J Crowcroft. Core based trees (CBT). In *SIGCOMM'93*, pages 85–95. ACM Press, 1993. 49, 50

[54] S Deering, D Estrin, D Farinacci, V Jacobson, C Liu, and L Wei. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, 1996. 50, 51, 52

[55] 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*, pages 93–104. ACM Press, 1998. 52, 54, 60, 66, 68, 69

[56] S Banerjee, B Bhattacharjee, and C Kommareddy. Scalable Application Layer Multicast. In *SIGCOMM'02*. ACM Press, 2002. 52, 55

[57] C Diot, B Levine, B Lyles, H Kassem, and D Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, / 2000. 52

[58] M Handley. Session Directories and Scalable Internet Multicast Address Allocation. In *SIGCOMM'98*, pages 105–116. ACM Press, 1998. 54, 62, 64, 66, 67, 68

[59] P Francis and P Radoslavov and R Govindan and B Lidell. YOID: Your Own Internet Distribution. WORK IN PROGRESS. 55, 56

[60] ADSL access - FAQ's. http://www.telkom.co.za. Last accessed 17 August 2004. 60

[61] D Thaler, M Handley, and D Estrin. The Internet Multicast Address Allocation Architecture. RFC 2908, Internet Engineering Task Force, September 2000. 60, 66, 67, 68, 69

[62] P Radoslavov, D Estrin, R Govindan, M Handley, S Kumar, and D Thaler. The Multicast Address-Set Claim (MASC) Protocol. RFC 2909, Internet Engineering Task Force, September 2000. 60, 66, 68

[63] S Hanna, B Patel, and M Shah. Multicast Address Dynamic Client Allocation Protocol (MADCAP). RFC 2730, Internet Engineering Task Force, December 1999. 60, 66, 68, 69

[64] M Handley and S Hanna. Multicast Address Allocation Protocol (AAP). Technical Report draft-ietf-malloc-aap-04.txt, Internet Engineering Task Force, June 2000. Expired December 2000. 62

[65] Internet Protocol Version 6 Multicast Addresses. http://www.iana.org/assignments/ipv6-multicast-addresses. Last accessed 22 July 2004. 67

[66] O Catrina, D Thaler, B Aboba, and E Guttman. Zeroconf Multicast Address Allocation Protocol (ZMAAP). Technical Report draft-ietf-zeroconf-zmaap-02.txt, Internet Engineering Task Force, October 2002. Expired April 2003. 69

[67] RAND(3) Linux Programmer's Manual. Applicable to Glibc 2.3.2, dated 15 November 2003. 81

[68] R Rivest. The MD4 Message-Digest Algorithm. RFC 1320, Internet Engineering Task Force, April 1992. 81, 82

[69] R Rivest. The MD5 Message-Digest Algorithm. RFC 1321, Internet Engineering Task Force, April 1992. 81, 82

[70] B Schneier. *Applied Cryptography*. Wiley, 1996. 82

[71] Port Numbers. http://www.iana.org/assignments/port-numbers. Last accessed 10 August 2004. 90

[72] P Kruus. A SURVEY OF MULTICAST SECURITY ISSUES AND ARCHITECTURES. 1998. 98

[73] R Canetti, J Garay, G Itkis, D Micciancio, M Naor, and B Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In *INFOCOMM'99*, 1999. 98

[74] S Kent and R Atkinson. Security Architecture for the Internet Protocol. RFC 2401, Internet Engineering Task Force, November 1998. 99

[75] S Kent and R Atkinson. IP Encapsulating Security Payload (ESP). RFC 2406, Internet Engineering Task Force, November 1998. 99

[76] D Harkins and D Carrel. The Internet Key Exchange (IKE). RFC 2409, Internet Engineering Task Force, November 1998. 99

[77] A Perrig, R Canetti, D Song, and D Tygar. Efficient and Secure Source Authentication for Multicast. In *NDSS'01*, 2001. 100

[78] Baugher and Carrara. The Use of TESLA in SRTP. Technical Report draft-ietf-msec-srtp-tesla-01.txt, Internet Engineering Task Force, July 2004. Expires January 2005. 100

[79] LBNL's Network Research Group. http://www-nrg.ee.lbl.gov. Last accessed 17 August 2004. 102

[80] ANSI X3J11. American National Standard for Information Systems — Programming Language — C. Technical Report X3.159-1989, 1989. Also ISO/IEC 9899. 113

[81] GNU Operating System - Free Software Foundation. http://www.gnu.org. Last accessed 23 August 2004. 113

[82] Berkeley DB. http://www.sleepycat.com/products/db.shtml. Last accessed 23 August 2004. 114

[83] OpenSSL: The Open Source toolkit for SSL/TLS. http://www.openssl.org. Last accessed 24 August 2004. 115

[84] W Richard Stevens. *UNIX Network Programming - Networking APIs: Sockets and XTI*, volume 1. Prentice-Hall, 2 edition, 1998. ISBN 0 13 490012 X. 115

[85] UNIX(7) Linux Programmer's Manual. Applicable to Glibc 2.3.2, dated 2 December 2002. 116

[86] The IEEE and The Open Group. The Open Group Base Specifications Issue 6. Technical Report POSIX.1c, 2004. Also ISO/IEC 9945-1:1996. 116

[87] GNU C Library - GNU Project - Free Software Foundation (FSF). http://www.gnu.org/software/libc/libc.html. Last accessed 8 October 2004. 116

[88] PTHREAD_COND(3) Manual. Applicable to Glibc 2.3.2. 119

[89] H Harney, U Meth, A Colegrove, and G Gross. GSAKMP. Technical Report draft-ietf-msec-gsakmp-sec-06.txt, Internet Engineering Task Force, June 2004. Expires December 2004. 137

[90] Berkeley DB Reference Guide, Version 4.2.52. http://www.sleepycat.com/docs/ref/toc.html. Last accessed 2 September 2004. 138