Network Working Group
Request for Comments: 2771
Category: Informational

R. Finlayson
LIVE.COM

February 2000

An Abstract API for Multicast Address Allocation

Status of this Memo

Copyright Notice

Abstract

This document describes the "abstract service interface" for the dynamic multicast address allocation service, as seen by applications. While it does not describe a concrete API (i.e., for a specific programming language), it describes - in abstract terms - the semantics of this service, including the guarantees that it makes to applications.

Additional documents (not necessarily products of the IETF) would describe concrete APIs for this service.

1. Introduction

Applications are the customers of a multicast address allocation service, so a definition of this service should include not only the inter-node network protocols that are used to implement it, but also the 'protocol' that applications use to access the service. While APIs ("application programming interfaces") for specific programming languages (or operating systems) are outside the domain of the IETF, it is appropriate for us to define - in abstract terms - the semantic interface that this service presents to applications. Specific APIs would then be based upon this abstract service interface.

Note that it is possible to implement the multicast address allocation service in at least two different ways. The first (and perhaps most common) way is for end nodes to allocate addresses by communicating with a separate "Address Allocation Server" node, using the "Host to Address Allocation Server" network protocol (MADCAP) [1][7]. Alternatively, an "Address Allocation Server" implementation

might be co-located (along with one or more applications) on an end node, in which case some other, internal, mechanism might be used to access the server. In either case, however, the abstract service interface (and, presumably, any specific APIs) would remain the same.

The remainder of this document describes the abstract interface.

Note that this interface is intended only for the allocation of dynamic multicast addresses, as used by the traditional multicast service model [2]. Future multicast service models might allocate or assign multicast addresses in other ways, but this is outside the scope of this document.

2. Abstract Data Types

The interface described below uses the following abstract data types:

- AddressFamily: e.g., IPv4 or IPv6

- MulticastAddress: An actual multicast address (i.e., that could subsequently be used as the destination of a datagram)

- MulticastAddressSet: A set of "MulticastAddress"es

- LanguageTag: The code for a (human) language, as defined in [4]

- Scope: An "administrative scope" [3] from which multicast addresses are to be allocated. Each scope is a "MulticastAddressSet", with an associated set of (character-string) names - indexed by "LanguageTag". (Each language tag has at most one corresponding name, per scope.) For each scope, a (language tag, name) pair may be defined to be the 'default' name for this scope. (See the section "Querying the name of a scope" below.)

  (An implementation of this abstract data type might also include other information, such as a default TTL for the scope.)

- Time: An (absolute) event time. This is used for specifying the "lifetime" of multicast addresses: the period of time during which allocated multicast addresses are guaranteed to be available. (It is also used to specify the desired start time for an "advance allocation".)

Note that a concrete API might prefer to specify some of these times as relative times (i.e., relative to the current time-of-day), rather than absolute time. (Relative times have the advantage of not requiring clock synchronization.)

- Lease: A compound data type that describes the result of a (successful) multicast address allocation. It consists of:
  - [MulticastAddressSet] The set of addresses that were allocated;
  - [AddressFamily] The address family of these addresses
  - [Time] The lifetime of these addresses (the same for each address)
  - [Time] The "start time" of the allocation. (See the discussion of "advance allocation" below.) (A concrete API would likely also include a MADCAP "Lease Identifier" [1].)
- NestingRelationship: A binary data type that describes whether or not two scopes nest. Two scopes nest if traffic sent sent to a multicast group within one scope could be seen by all hosts present within the other scope were they to join the multicast group within the first scope. This value would be "False" for overlapping scopes where only some (or none) of the hosts within the second scope could see traffic sent to an address due to the presence of an administratively scoped boundary. In cases where the first and second scopes are topologically identical this value would be "True."
- Status: A result code.

3. The Abstract Interface

3.1 Allocating multicast addresses:

```
alloc_multicast_addr(in AddressFamily family,
                     in Scope scope,
                     in Integer minDesiredAddresses,
                     in Integer maxDesiredAddresses,
                     in Time minDesiredStartTime,
                     in Time maxDesiredStartTime,
                     in Time minDesiredLifetime,
                     in Time maxDesiredLifetime,
                     out Lease multicastAddressSetLease,
                     out Status status)
```

This operation attempts to allocate a set of multicast addresses (the size of this set is in the range [minDesiredAddresses, maxDesiredAddresses]) within the given address family and scope, and within a given range of desired lifetimes. ("minDesiredStartTime" and "maxDesiredStartTime" are used to specify "advance allocation"; this is described in more detail below.)

If the address allocation succeeds, the result is returned in "multicastAddressSetLease" (with "status" = OK).

During the lifetime of this lease, the allocation service will make a "best-effort" attempt to not allocate any of these addresses to others. (However, once the lease's lifetime has expired, any of its addresses can be allocated to others.)

Multicast addresses are allocated for a limited lifetime. An application may attempt to extend this lifetime, but this operation may fail. Therefore, an application must be prepared for the possibility it will not be able to use the same addresses for as long as it desires. In particular, the application must be prepared to either quit early (because its original multicast address assignments have expired), or, alternatively, to occasionally 'renumber' its multicast addresses (in some application or higher-level-protocol dependent way), by making a new allocation. However, if an application needs to consider 'renumbering', it will always know this in advance, at the time it acquired its current address(es) - by checking the lifetime in the returned lease. An application will never need to be notified asynchronously of the need to 'renumber'.

Possible errors:

- bad address family
- bad scope
- bad desired number of addresses (e.g., max < min)
- bad desired lifetimes (e.g., max < min)
- errors with the two "start time" parameters (see "Advance allocation" below)
- no addresses can be allocated (for the requested parameters)

An allocation attempt can also fail with a result "status" code of TRY_LATER, indicating that the requested allocation cannot be made at this time, but that it might succeed if the caller retries the attempt at some future time. (This future time is returned in the "start time" field of the

"multicastAddressSetLease";
the other parts of this lease are undefined.)

Note that a concrete (i.e., programming language-specific) API for multicast address allocation will probably include additional, specialized variants of this general allocation operation. For instance, it may include separate operations for:

- allocating only a single address (i.e., minDesiredAddresses = maxDesiredAddresses = 1);
- (attempting to) allocate an address with a single, fixed lifetime (i.e., minDesiredLifetime = maxDesiredLifetime);
- (attempting to) allocate an address for immediate use (i.e., minDesiredStartTime = maxDesiredStartTime = 'now')

3.2 Changing multicast addresses' lifetime:

```
change_multicast_addr_lifetime(in Lease multicastAddressSetLease,
                               in Time minDesiredLifetime,
                               in Time maxDesiredLifetime,
                               out Time lifetime)
```

This operation attempts to change the lifetime of previously allocated multicast addresses. Unless an error occurs, it returns the new lifetime (which might remain unchanged).

Possible errors:

- bad address family
- bad durations (e.g., max < min)

- the addresses' lifetime could not be changed
  (and the existing lifetime was not in the requested range
  [minDesiredLifetime,maxDesiredLifetime])
- the addresses were not ones that we had allocated
  (see section 5.9) - or they have already expired

3.3 Deallocating multicast addresses:

deallocate_multicast_addr(in Lease multicastAddressSetLease) This operation attempts to deallocate previously allocated multicast addresses.

Possible errors:

- bad address family
- the addresses were not ones that we had allocated
  (or they have already expired)

3.4 Querying the set of usable multicast address scopes:

```
get_multicast_addr_scopes(in AddressFamily family,
                          out "set of" Scope)
```

This operation returns the set of administrative multicast address scopes that are defined for this node.

Possible errors:

- bad address family

3.5 Querying the name of a scope:

```
get_scope_name(in Scope scope,
               in LanguageTag language,
               out String name,
               out LanguageTag languageForName)
```

This operation returns a character-string name for a given scope. If the scope has a name in the specified "language", then this name (and language) is returned. Otherwise, the scope's default (language, name) pair is returned.

Possible errors:

- bad scope.

3.6 Querying the nesting state of known usable multicast address scopes:

```
get_scope_nesting_state(in "set of" Scope,
                        out "matrix of" NestingRelationship)
```

Possible errors:

- bad scope.
- nesting state undetermined at this time.

This operation would return a matrix that shows the current nesting relationships between the supplied set of scopes which would have previously been supplied via the get_multicast_addr_scopes(...) function.

3.7 Querying the set of scopes that a given scope is known to nest inside:

```
get_larger_scopes(in Scope,
                  out "set of" Scope)
```

This operation returns the set of administrative multicast address scopes that are known to encompass the supplied Scope.

Possible errors:

- bad scope.
- nesting state undetermined at this time.

3.8 Querying the set of scopes that are known to nest inside a given scope:

```
get_smaller_scopes(in Scope,
                   out "set of" Scope)
```

This operation returns the set of administrative multicast address scopes that are known to nest inside the supplied Scope (NB this would include those scopes that are topologically identical to the supplied scope).

Possible errors:

- bad scope.
- nesting state undetermined at this time.

3.9 Note: The decision as to who is allowed to deallocate (or change the lifetime of) a previously allocated multicast address set lease is implementation-specific, and depends upon the security policy of the host system. Thus it is not specified in this abstract API. One possible starting point, however, is the following:

A previously allocated multicast address can be deallocated (or have its lifetime queried or changed) by the same "principal", and on the same node, as that which originally allocated it. ("principal" might, for example, be a "user" in the host operating system.)

3.10 Advance allocation

By specifying "minDesiredStartTime = maxDesiredStartTime = 'now'", the address allocation operation - "alloc_multicast_addr" - described above can be used to request a set of multicast addresses that can be used *immediately* (and until their lifetime expires). During this whole time, the addresses are not available for allocation to others.

It is also possible - using the "minDesiredStartTime" and "maxDesiredStartTime" parameters - to allocate multicast addresses *in advance* - i.e., so that they have a future "start time" as well as an expiration time. Before the start time, the multicast addresses may be allocated to others.

Advance allocation is convenient for allocating addresses for events that begin far in the future - e.g., several weeks or months away. Without advance allocation, it would be necessary to allocate addresses for a long period of time - even when it will not be used. Such a request would not only be a wasteful use of the multicast address space, but it may also be difficult to implement (especially since address allocations are expected to remain valid in spite of topology changes).

Advance allocation requests can produce the following errors (in addition to those defined earlier):

- bad start time durations (e.g., max < min)
- requested start times conflict with requested lifetimes (i.e., min start time > max lifetime)

The following operation is also defined:

```
change_multicast_addr_start_time(in Lease multicastAddressSetLease,
                                 in Time minDesiredStartTime,
                                 in Time maxDesiredStartTime,
                                 out Time startTime)
```

This operation attempts to change the start time of previously allocated multicast addresses. Unless an error occurs, it returns the new start time (which might remain unchanged).

Possible errors: the same as "change_multicast_addr_lifetime"

4. Security Considerations

As noted in section 5.9 above, each implementation of this abstract API should define a security policy that specifies when (and by whom) previously allocated multicast addresses can be deallocated (or queried, or have their lifetime changed).

Because multicast addresses are a finite resource, there is a potential for a "denial of service" attack by allocating a large number of multicast addresses without deallocating them. Preventing such an attack, however, is not the role of the API, but rather by the underlying MAAS ("Multicast Address Allocation Server(s)" [6]).

5. Acknowledgements

Many thanks to other participants in the "MALLOC" working group - in particular Steve Hanna, Dave Thaler, Roger Kermode, and Pavlin Radoslavov - for their valuable comments.

6. References

[1] Hanna, S., Patel, B. and M. Shah, "Multicast Address Dynamic Client Allocation Protocol (MADCAP)", RFC 2730, December 1999.

[2] Deering, S., "Host Extensions for IP Multicasting", STD 5, RFC 1112, August 1989.

[3] Meyer, D., "Administratively Scoped IP Multicast", BCP 23, RFC 2365, July, 1998.

[4] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

[5] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[6] Estrin, D., Handley, M. and D. Thaler, "The Internet Multicast Address Allocation Architecture", Work in Progress.

[7] Kermode, R., "MADCAP Multicast Scope Nesting State Option", Work in Progress.

7. Author's Address

Ross Finlayson,
Live Networks, Inc. (LIVE.COM)

EMail: finlayson@live.com
WWW: http://www.live.com/

8. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.