

IDMR Working Group
INTERNET-DRAFT
Expires December 2002
Type: Informational

Dave Thaler
Microsoft
Bill Fenner
AT&T Research
Bob Quinn
Stardust.com
29 June 2002

Socket Interface Extensions for Multicast Source Filters
<[draft-ietf-idmr-msf-api-03.txt](#)>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of [Section 10 of RFC2026](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

Copyright Notice

Draft

Multicast Source Filter API

June 2002

Copyright (C) The Internet Society (2002). All Rights Reserved.

1. Abstract

IGMPv3 for IPv4 adds the capability for applications to express source filters on multicast group memberships, which allows receiver applications to determine the set of senders (sources) from which to accept multicast traffic. This capability also simplifies support of one-to-many type multicast applications. It is expected that in the future, the same capability will be available in IPv6 as well.

This document specifies new socket options and ioctl commands to manage source filters for IP Multicast group memberships. It also defines the socket structures to provide input and output arguments to these new APIs. These extensions are designed to provide access to the source filtering features, while introducing a minimum of change into the system and providing complete compatibility for existing multicast applications.

2. Introduction

The de facto standard application program interface (API) for TCP/IP applications is the "sockets" interface. Although this API was developed for Unix in the early 1980s it has also been implemented on a wide variety of non-Unix systems. TCP/IP applications written using the sockets API have in the past enjoyed a high degree of portability and we would like the same portability with applications that employ multicast source filters. Changes are required to the sockets API to support such filtering and this memo describes these changes.

This document specifies new socket options and ioctl commands to manage source filters for IP Multicast group memberships. It also defines the socket structures to provide input and output arguments to these new APIs. These extensions are designed to provide access to the source filtering features required by applications, while introducing a minimum of change into the system and providing complete compatibility for existing multicast applications.

Furthermore, [RFC 2553](#) [1] defines socket interface extensions for

IPv6, including protocol-independent functions for most operations. However, while it defines join and leave functions

Expires December 2002

[Page 2]

Draft

Multicast Source Filter API

June 2002

for IPv6, it does not provide protocol-independent versions of these operations. Such functions will be described in this document.

[3.](#) Design Considerations

There are a number of important considerations in designing changes to this well-worn API:

- o The API changes should provide both source and binary compatibility for programs written to the original API. That is, existing program binaries should continue to operate when run on a system supporting the new API. In addition, existing applications that are re-compiled and run on a system supporting the new API should continue to operate. Simply put, the API changes for multicast receivers that specify source filters should not break existing programs.
- o The changes to the API should be as small as possible in order to simplify the task of converting existing multicast receiver applications to use source filters.
- o Applications should be able to detect when the new source filter APIs are unavailable (e.g., calls fail with the ENOTSUPP error) and react gracefully (e.g., revert to old non-source-filter API or display a meaningful error message to the user).

[3.1.](#) What Needs to be Added

The current IP Multicast APIs allow a receiver application to specify the group address (destination) and (optionally) the local interface. These existing APIs need not change (and cannot, to retain binary compatibility). Hence, what is needed are new source filter APIs that provide the same functionality and also

allow receiver multicast applications to:

- o Specify zero or more unicast (source) address(es) in a source filter.
- o Determine whether the source filter describes an inclusive or exclusive list of sources.

Expires December 2002

[Page 3]

Draft

Multicast Source Filter API

June 2002

The new API design must enable this functionality for both IPv4 and IPv6.

[3.2.](#) Data Types

The data types of the structure elements given in this memo are intended to be examples, not absolute requirements. Whenever possible, data types from Draft 6.6 (March 1997) of POSIX 1003.1g are used: `uintN_t` means an unsigned integer of exactly N bits (e.g., `uint32_t`). We also assume the argument data types from 1003.1g when possible (e.g., the final argument to `setsockopt()` is a `size_t` value).

[3.3.](#) Headers

When function prototypes and structures are shown, we show the headers that must be `#included` to cause that item to be defined.

[3.4.](#) Structures

When structures are described, the members shown are the ones that must appear in an implementation. Additional, nonstandard members may also be defined by an implementation. As an additional precaution, nonstandard members could be verified by Feature Test Macros as described in IEEE Std 1003.1. (Such Feature Test Macros are not defined by this RFC.) The ordering shown for the members of a structure is the recommended ordering, given alignment considerations of multibyte members, but an implementation may

order the members differently.

[4.](#) Overview of APIs

There are a number of different APIs described in this document, that are appropriate for a number of different application types and IP versions. Before providing detailed descriptions, this section provides a "taxonomy" with a brief description of each.

IPv4 Multicast Source Filter APIs:

- o Basic (Delta-based): Use `setsockopt()` and reference a single source and group address pair to make incremental changes

Expires December 2002

[Page 4]

Draft

Multicast Source Filter API

June 2002

- + Any-Source: Data accepted from any source by default, but source filter control is available
- + Controlled-Source: A source filter is required
- o Advanced (Full-state): Use `ioctl()` and reference the entire set of sources with the group address to affect membership changes

Protocol-Independent Multicast Source Filter APIs:

- o Basic (Delta-based): Use `setsockopt()` and reference a single source and group address pair to make incremental changes
- + Any-Source: Data accepted from any source by default, but source filter control is available
- + Controlled-Source: A source filter is required
- o Advanced (Full-state): Use `ioctl()` and reference the entire set of sources

One might ask why the protocol-independent APIs cannot accomodate IPv4 applications as well as IPv6. Since any IPv4 application requires modification to use multicast source filters anyway, it might seem like a good opportunity to create IPv6-compatible

source code.

The primary reasons for extending an IPv4-specific API are:

- o To minimize changes needed in existing IPv4 multicast application source code to add source filter support
- o To avoid overloading APIs to accomodate the differences between IPv4 interface addresses (e.g., in the `ip_mreq` structure), and interface indices.

[5.](#) IPv4 Multicast Source Filter APIs

Version 3 of the Internet Group Management Protocol (IGMPv3) [[2](#)] provides the ability to communicate source filter information to the router and hence avoid pulling down data from unwanted sources onto the local link. However, source filters may be implemented by the operating system regardless of whether the routers support IGMPv3, so when the source-filter API is available, applications

Expires December 2002

[Page 5]

Draft

Multicast Source Filter API

June 2002

can always benefit from using it.

There are two categories of the IPv4 source-filter APIs, both of which are designed to allow multicast receiver applications to designate the unicast address(es) of sender(s) along with the multicast group (destination address) to receive.

- o The "Basic" (Delta-based) API is the simpler of the two and allows an application to reference a single source address in each operation.
- o The "Advanced" (Full-state) API allows an application to define a source-filter comprised of zero or more source addresses.

[5.1.](#) Basic (Delta-based) API for IPv4

Some applications desire the simplicity of a delta-based API in which each function call references a single source address along with the multicast group address on which to listen. Such applications typically fall into either of two categories:

Any-source:

By default, all sources are accepted. Individual sources may be turned off and back on as needed over time.

Controlled-source:

Only sources in a given list are allowed. The list may change over time.

[5.1.1.](#) Any-Source IPv4 Applications

The following socket options are defined in <netinet/in.h> for applications in the any-source category:

Socket option	Argument type
IP_ADD_MEMBERSHIP	struct ip_mreq
IP_BLOCK_SOURCE	struct ip_mreq_source
IP_UNBLOCK_SOURCE	struct ip_mreq_source
IP_DROP_MEMBERSHIP	struct ip_mreq

IP_ADD_MEMBERSHIP and IP_DROP_MEMBERSHIP are already implemented

on most operating systems, and are used to join and leave an any-source group.

IP_BLOCK_SOURCE can be used to block data from a given source to a given group (e.g., if the user "mutes" that source), and IP_UNBLOCK_SOURCE can be used to undo this (e.g., if the user then "unmutes" the source).

The argument types of these options are defined as a result of including the <netinet/in.h> header.

```
struct ip_mreq {  
    struct in_addr imr_multiaddr; /* IP multicast address of group */
```

```

        struct in_addr imr_interface; /* local IP address of interface */
};

struct ip_mreq_source {
    struct in_addr imr_multiaddr; /* IP multicast address of group */
    struct in_addr imr_sourceaddr; /* IP address of source */
    struct in_addr imr_interface; /* local IP address of interface */
};

```

[5.1.2.](#) Controlled-Source IPv4 Applications

The following socket options are available for applications in the Controlled-source category:

Socket option	Argument type
IP_ADD_SOURCE_MEMBERSHIP	struct ip_mreq_source
IP_DROP_SOURCE_MEMBERSHIP	struct ip_mreq_source
IP_DROP_MEMBERSHIP	struct ip_mreq

These options would be used, for example, by "single-source" style applications such as audio/video broadcasting. They can also be used for logical multi-source sessions where each source independently allocates its own source-specific group address.

IP_DROP_MEMBERSHIP can be supported, as a convenience, to drop all sources which have been joined. The operations are the same as if the socket had been closed.

[5.1.3.](#) Error Codes

When the option would be legal on the group, but an address is invalid (e.g., when trying to block a source that is already blocked by the socket, or when trying to drop an unjoined group) the error generated is EADDRNOTAVAIL.

When the option itself is not legal on the group (i.e., when trying a Controlled-Source option on a group after doing `IP_ADD_MEMBERSHIP`, or when trying an Any-Source option without doing `IP_ADD_MEMBERSHIP`) the error generated is `EINVAL`.

When any of these options are used with `getsockopt()`, the error generated is `EOPNOTSUPP`.

Finally, if the implementation imposes a limit on the maximum number of sources in a source filter, `ENOBUFS` is generated when an operation would exceed the maximum.

[5.2.](#) Advanced (Full-state) API for IPv4

Applications which require the ability to switch between filter modes without leaving a group must use a full-state API (i.e., to change the semantics of the source filter from inclusive to exclusive, or vice versa). Applications which use a large source list for a given group address should also use the full-state API, since filter changes can be done atomically in a single operation.

For this purpose the following are defined in `<sys/sockio.h>`:

- o `ioctl()` `SIOCGIPMSFILTER`: to retrieve the list of source addresses that comprise the source filter along with the current filter mode.
- o `ioctl()` `SIOCSIPMSFILTER`: to set or modify the source filter content (e.g. unicast source address list) or mode (exclude or include).

`SIOCGIPMSFILTER` could not be done with `getsockopt()`, since the group and interface must be passed down in order to retrieve the correct filter. This can, however, be done with an `ioctl()`, and hence for symmetry, both gets and sets are done with an `ioctl`.

[5.2.1.](#) Set Source Filter

Ioctl option	Argument type
SIOCSIPMSFILTER	struct ip_msfilter

The argument type of this option is defined as a result of including the <netinet/in.h> header.

```
struct ip_msfilter {
    struct in_addr imsf_multiaddr; /* IP multicast address of group */
    struct in_addr imsf_interface; /* local IP address of interface */
    uint32_t      imsf_fmode;      /* filter mode */
    uint32_t      imsf_numsrc;     /* number of sources in src_list */
    struct in_addr imsf_slist[1];  /* start of source list */
};

#define IP_MSFILTER_SIZE(numsrc) \
    (sizeof(struct ip_msfilter) - sizeof(struct in_addr) \
    + (numsrc) * sizeof(struct in_addr))
```

The imsf_fmode mode is a 32-bit integer that identifies the filter mode. The value of this field must be either MCAST_INCLUDE or MCAST_EXCLUDE, which are likewise defined in <netinet/in.h>.

If the implementation imposes a limit on the maximum number of sources in a source filter, ENOBUFS is generated when the operation would exceed the maximum.

[5.2.2.](#) Get Source Filter

SIOCGIPMSFILTER cannot be done with getsockopt(), since the group and interface must be passed down in order to retrieve the correct filter. This can, however, be done with an ioctl():

Ioctl option	Argument type
SIOCGIPMSFILTER	struct ip_msfilter

The structure length pointed to must be at least IP_MSFILTER_SIZE(0) bytes long, and the imsf_numsrc parameter should be set so that IP_MSFILTER_SIZE(imsf_numsrc) indicates the buffer length. The result of this call will be that the imsf_multiaddr and imsf_interface fields will be unchanged, while imsf_fmode, imsf_numsrc, and as many source addresses as fit will be filled into the application's buffer.

If the application does not know the size of the source list beforehand, it can make a reasonable guess (e.g., 0), and if upon completion, the `imsf_numsrc` field holds a larger value, the operation can be repeated with a large enough buffer.

[6.](#) Protocol-Independent Multicast Source Filter APIs

Protocol-independent functions are provided for join and leave operations so that an application may pass a `sockaddr_storage` structure obtained from calls such as `getaddrinfo()` [\[1\]](#) as the group to join. For example, an application can resolve a DNS name (e.g., `NTP.MCAST.NET`) to a multicast address which may be either IPv4 or IPv6, and may easily join and leave the group.

While the Multicast Listener Discovery (MLD) protocol [\[3\]](#) for IPv6 does not currently support source-filters, the operating system may provide filtering services with this API. A future version of MLD will support source-filters on routers, providing functionality equivalent to IGMPv3 for IPv4.

[6.1.](#) Basic (Delta-based) API

The reception of multicast packets is controlled by the `setsockopt()` options summarized below. An error of `EOPNOTSUPP` is returned if these options are used with `getsockopt()`.

Socket option	Argument type
<code>MCAST_JOIN_GROUP</code>	<code>struct group_req</code>
<code>MCAST_BLOCK_SOURCE</code>	<code>struct group_source_req</code>
<code>MCAST_UNBLOCK_SOURCE</code>	<code>struct group_source_req</code>
<code>MCAST_LEAVE_GROUP</code>	<code>struct group_req</code>
<code>MCAST_JOIN_SOURCE_GROUP</code>	<code>struct group_source_req</code>
<code>MCAST_LEAVE_SOURCE_GROUP</code>	<code>struct group_source_req</code>

The argument types of these options are defined as a result of including the `<netinet/in.h>` header.

```
struct group_req {
    uint32_t          gr_interface; /* interface index */
    struct sockaddr_storage gr_group; /* group address */
};
```

```
struct group_source_req {
```

Draft

Multicast Source Filter API

June 2002

```
uint32_t          gsr_interface; /* interface index */
struct sockaddr_storage gsr_group; /* group address */
struct sockaddr_storage gsr_source; /* source address */
};
```

The `sockaddr_storage` structure is defined in [RFC 2553](#) [1] to be large enough to hold either IPv4 or IPv6 address information.

The rules for generating errors are the same as those given in [Section 5.1.3](#).

[6.2](#). Advanced (Full-state) API

For the full-state API, the following `ioctl()` options are defined in `<sys/socket.h>`. An `ioctl()` is required for obtaining the filter on a group, since it requires both in and out parameter fields, which cannot be done with `getsockopt`. For symmetry, we use `ioctl()` for both get and set operations.

Ioctl option	Argument type
<code>SIOCGMSFILTER</code>	<code>struct group_filter</code>
<code>SIOCSMSFILTER</code>	<code>struct group_filter</code>

The argument types of these options are defined as a result of including the `<netinet/in.h>` header.

```
struct group_filter {
    uint32_t          gf_interface; /* interface index */
    struct sockaddr_storage gf_group; /* multicast address */
    uint32_t          gf_fmode; /* filter mode */
    uint32_t          gf_numsrc; /* number of sources */
    struct sockaddr_storage gf_slist[1]; /* source address */
};

#define GROUP_FILTER_SIZE(numsrc) \
    (sizeof(struct group_filter) - sizeof(struct sockaddr_storage) \
     + (numsrc) * sizeof(struct sockaddr_storage))
```

The `imf_numsrc` field is used in the same way as described for `imsf_numsrc` in [section 5.2.2](#).

Draft

Multicast Source Filter API

June 2002

7. Security Considerations

Although source filtering can help to combat denial-of-service attacks, source filtering alone is not a complete solution, since it does not provide protection against spoofing the source address to be an allowed source. Multicast routing protocols which use reverse-path forwarding based on the source address, however, do provide some natural protection against spoofing the source address, since if a router receives a packet on an interface other than the one towards the "real" source, it will drop the packet. However, this still does not provide any guarantee of protection.

8. Acknowledgements

This draft was updated based on feedback from the IETF's Internet-Draft Multicast Remnants (IDMR) Working Group. Wilbert de Graaf also provided many helpful comments.

9. Authors' Addresses

Dave Thaler
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
Phone: +1 425 703 8835
EMail: dthaler@microsoft.com

Bill Fenner
75 Willow Road
Menlo Park, CA 94025
Phone: +1 650 867 6073
EMail: fenner@research.att.com

Bob Quinn

IP Multicast Initiative (IPMI)
Stardust.com
1901 S. Bascom Ave. #333
Campbell, CA 95008
Phone: +1 408 879 8080
EMail: rcq@ipmulticast.com

Expires December 2002

[Page 12]

Draft

Multicast Source Filter API

June 2002

10. Normative References

- [1] Gilligan, R., Thomson, S., Bound, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", [RFC 2553](#), March 1999.

11. Non-normative References

- [2] Cain, B., Deering, S., Fenner, B., Kouvelas, I., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", [draft-ietf-idmr-igmp-v3-11.txt](#), Work in progress, May 2002.
- [3] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", [RFC 2710](#), October 1999.

12. Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed

for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Expires December 2002

[Page 13]

Draft

Multicast Source Filter API

June 2002

Table of Contents

1: Abstract	2
2: Introduction	2
3: Design Considerations	3
3.1: What Needs to be Added	3
3.2: Data Types	4
3.3: Headers	4
3.4: Structures	4
4: Overview of APIs	4
5: IPv4 Multicast Source Filter APIs	5
5.1: Basic (Delta-based) API for IPv4	6
5.1.1: Any-Source IPv4 Applications	6
5.1.2: Controlled-Source IPv4 Applications	7
5.1.3: Error Codes	8
5.2: Advanced (Full-state) API for IPv4	8
5.2.1: Set Source Filter	9
5.2.2: Get Source Filter	9
6: Protocol-Independent Multicast Source Filter APIs	10
6.1: Basic (Delta-based) API	10
6.2: Advanced (Full-state) API	11

7: Security Considerations	12
8: Acknowledgements	12
9: Authors' Addresses	12
10: Normative References	13
11: Non-normative References	13
12: Full Copyright Statement	13