

NAME

IP - Internet Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
```

DESCRIPTION

IP is the network-layer protocol used by the Internet protocol family. It encapsulates TCP and UDP messages into datagrams to be transmitted by the network interface. Normally, applications do not need to interface directly to IP. However, certain multicast socket options are controlled by passing options to the IPPROTO_IP protocol level through a UDP socket, and IP Type of Service is controlled by passing an option to the IPPROTO_IP protocol level through either a TCP or UDP socket. (See the *getsockopt(2)* manual page.)

The following socket options are defined in the include file `<netinet/in.h>`. The type of the variable pointed to by the *optval* parameter is indicated in parentheses. The data types `struct ip_mreq`, `struct ip_mreq_source`, `struct group_req`, `struct group_source_req`, and `struct in_addr` are defined in `<netinet/in.h>`.

IP_TOS	(unsigned int) Sets the IP Type of Service. Allowable values for <i>optval</i> are 4 for high reliability, 8 for high throughput, and 16 for low delay. Other values will not return an error, but may have unpredictable results. Default: zero.
IP_ADD_MEMBERSHIP	(struct ip_mreq) Requests that the system join a multicast group in "exclude" mode.
MCAST_JOIN_GROUP	(struct group_req) Requests that the system join a multicast group in "exclude" mode.
IP_DROP_MEMBERSHIP	(struct ip_mreq) Allows the system to leave a multicast group.
MCAST_LEAVE_GROUP	(struct group_req) Allows the system to leave a multicast group.
IP_BLOCK_SOURCE	(struct ip_mreq_source) Adds a source address to the list of blocked sources for a multicast group in "exclude" mode.
MCAST_BLOCK_SOURCE	(struct group_source_req) Adds a source address to the list of blocked sources for a multicast group in "exclude" mode.
IP_UNBLOCK_SOURCE	(struct ip_mreq_source) Removes a source address from the list of blocked sources for a multicast group in "exclude" mode.
MCAST_UNBLOCK_SOURCE	(struct group_source_req) Removes a source address from the list of blocked sources for a multicast group in "exclude" mode.
IP_ADD_SOURCE_MEMBERSHIP	(struct ip_mreq_source) Adds a source address to the list of allowed sources for a multicast group in "include" mode, joining the group in "include" mode if not already joined.
MCAST_JOIN_SOURCE_GROUP	(struct group_source_req) Adds a source address to the list of allowed sources for a multicast group in "include" mode, joining the group in "include" mode if not already joined.
IP_DROP_SOURCE_MEMBERSHIP	(struct ip_mreq_source) Removes a source address from the list of allowed sources for a multicast group in "include" mode, leaving the group when the last source is removed.
MCAST_LEAVE_SOURCE_GROUP	(struct group_source_req) Removes a source address from the list of allowed sources for a multicast group in "include" mode, leaving the group when the last source is removed.



<code>IP_MULTICAST_IF</code>	(<code>struct in_addr</code>) Specifies a network interface other than the default to be used when sending multicast datagrams through this socket. Default: multicast datagrams are sent from the interface associated with the specific multicast group, with the default multicast route or with the default route.
<code>IP_MULTICAST_LOOP</code>	(<code>unsigned char</code> ; boolean) Enables or disables loopback in the IP layer for multicast datagrams sent through this socket. The value of the variable pointed to by <i>optval</i> is zero (disable) or non-zero (enable). This option is provided for compatibility only. Normally, multicast datagrams are always looped back if the system has joined the group. See <i>DEPENDENCIES</i> below. Default: enabled.
<code>IP_MULTICAST_TTL</code>	(<code>unsigned char</code>) Specifies the time-to-live value for multicast datagrams sent through this socket. The value of the variable pointed to by <i>optval</i> can be zero through 255. Default: one.

An application joins a multicast group on a network interface in order to receive multicast datagrams sent on the network to which that interface connects. An application can join up to `IP_MAX_MEMBERSHIPS` multicast groups on each socket. `IP_MAX_MEMBERSHIPS` is defined in `<netinet/in.h>`. However, each network interface may impose a smaller system-wide limit because of interface resource limitations and because the system uses some link-layer multicast addresses.

The application must also bind to the destination port number in order to receive datagrams that are sent to that port number. If the application binds to the address `INADDR_ANY`, it may receive all datagrams that are sent to the port number. If the application binds to a multicast group address, it may receive only datagrams sent to that group and port number. It is not necessary to join a multicast group in order to send datagrams to it.

For each multicast group that an application joins on a given socket and network interface, there is an associated filter mode and source list. The filter mode can be "exclude" mode or "include" mode. The source list is a set of IP addresses which will be compared to the source addresses of received multicast datagrams.

An application uses "exclude" mode when it wants to block reception of multicast datagrams from a specific set of sources, while allowing multicast datagrams from all other sources. For groups in "exclude" mode, the source list may be empty, thus allowing reception of multicast datagrams from all sources.

An application uses "include" mode when it wants to allow reception of multicast datagrams from a specific set of sources, while blocking multicast datagrams from all other sources. For groups in "include" mode, the source list needs to contain at least one member in order for the application to stay joined to the group; removing the last address from an "include" mode source list will cause the application to leave the group.

`IP_ADD_MEMBERSHIP` and `MCAST_JOIN_GROUP` request that the system join a multicast group on the specified interface. The group is joined in "exclude" mode, with an empty source list. The `imr_interface` field holds the IPv4 address of a local interface, or `INADDR_ANY`. The `gr_interface` field holds the interface index of a local interface, or zero. If the interface address is `INADDR_ANY`, or if the interface index is zero, the system joins the specified group on the interface from which datagrams for that group would be sent, based on the routing configuration. For example:

```
struct ip_mreq mreq;
mreq.imr_multiaddr.s_addr = inet_addr("224.1.2.3");
mreq.imr_interface.s_addr = INADDR_ANY;
setsockopt(s, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));

struct group_req gr;
struct sockaddr_in *sin;
sin = (struct sockaddr_in *)&gr.gr_group;
bzero(sin, sizeof(*sin));
sin->sin_family = AF_INET;
sin->sin_addr.s_addr = inet_addr("224.1.2.3");
gr.gr_interface = 0;
setsockopt(s, IPPROTO_IP, MCAST_JOIN_GROUP, &gr, sizeof(gr));
```

`IP_DROP_MEMBERSHIP` and `MCAST_LEAVE_GROUP` allow the system to leave a multicast group. The `imr_interface` field holds the IPv4 address of a local interface, or `INADDR_ANY`. The

`gr_interface` field holds the interface index of a local interface, or zero. If the interface address is `INADDR_ANY`, or if the interface index is zero, the system chooses a multicast group by matching the multicast address only. For example:

```
struct ip_mreq mreq;
mreq.imr_multiaddr.s_addr = inet_addr("224.1.2.3");
mreq.imr_interface.s_addr = INADDR_ANY;
setsockopt(s, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq));

struct group_req gr;
struct sockaddr_in *sin;
sin = (struct sockaddr_in *)&gr.gr_group;
bzero(sin, sizeof(*sin));
sin->sin_family = AF_INET;
sin->sin_addr.s_addr = inet_addr("224.1.2.3");
gr.gr_interface = 0;
setsockopt(s, IPPROTO_IP, MCAST_LEAVE_GROUP, &gr, sizeof(gr));
```

`IP_BLOCK_SOURCE` and `MCAST_BLOCK_SOURCE` add a source address to the list of sources being blocked. `IP_UNBLOCK_SOURCE` and `MCAST_UNBLOCK_SOURCE` remove a source address from the list of sources being blocked. The group must already be joined, and must be in "exclude" mode. The `imr_interface` field holds the IPv4 address of a local interface, or `INADDR_ANY`. The `gsr_interface` field holds the interface index of a local interface, or zero. If the interface address is `INADDR_ANY`, or if the interface index is zero, the system chooses a multicast group by matching the multicast address only. For example:

```
struct ip_mreq_source imr;
imr.imr_multiaddr.s_addr = inet_addr("224.1.2.3");
imr.imr_sourceaddr.s_addr = inet_addr("10.4.5.6");
imr.imr_interface.s_addr = INADDR_ANY;
setsockopt(s, IPPROTO_IP, IP_BLOCK_SOURCE, &imr, sizeof(imr));
setsockopt(s, IPPROTO_IP, IP_UNBLOCK_SOURCE, &imr, sizeof(imr));

struct group_source_req gsr;
struct sockaddr_in *sin;
sin = (struct sockaddr_in *)&gsr.gsr_group;
bzero(sin, sizeof(*sin));
sin->sin_family = AF_INET;
sin->sin_addr.s_addr = inet_addr("224.1.2.3");
sin = (struct sockaddr_in *)&gsr.gsr_source;
bzero(sin, sizeof(*sin));
sin->sin_family = AF_INET;
sin->sin_addr.s_addr = inet_addr("10.4.5.6");
gsr.gsr_interface = 0;
setsockopt(s, IPPROTO_IP, MCAST_BLOCK_SOURCE, &gsr, sizeof(gsr));
setsockopt(s, IPPROTO_IP, MCAST_UNBLOCK_SOURCE, &gsr, sizeof(gsr));
```

`IP_ADD_SOURCE_MEMBERSHIP` and `MCAST_JOIN_SOURCE_GROUP` add a source address to the list of allowed sources for a multicast group in "include" mode, joining the group in "include" mode if not already joined. `IP_DROP_SOURCE_MEMBERSHIP` and `MCAST_LEAVE_SOURCE_GROUP` remove a source address from the list of allowed sources for a multicast group in "include" mode, leaving the group if the last source is being removed. The `imr_interface` field holds the IPv4 address of a local interface, or `INADDR_ANY`. The `gsr_interface` field holds the interface index of a local interface, or zero. If the interface address is `INADDR_ANY` or the interface index is zero, the behavior depends on whether the group is being joined. If the group is being joined, the system joins the specified group on the interface from which datagrams for that group would be sent, based on the routing configuration; otherwise, the system chooses a multicast group by matching the multicast address only. For example:

```
struct ip_mreq_source imr;
imr.imr_multiaddr.s_addr = inet_addr("224.1.2.3");
imr.imr_sourceaddr.s_addr = inet_addr("10.4.5.6");
imr.imr_interface.s_addr = INADDR_ANY;
setsockopt(s, IPPROTO_IP, IP_ADD_SOURCE_MEMBERSHIP, &imr, sizeof(imr));
setsockopt(s, IPPROTO_IP, IP_DROP_SOURCE_MEMBERSHIP, &imr, sizeof(imr));
```

```

struct group_source_req gsr;
struct sockaddr_in *sin;
sin = (struct sockaddr_in *)&gsr.gsr_group;
bzero(sin, sizeof(*sin));
sin->sin_family = AF_INET;
sin->sin_addr.s_addr = inet_addr("224.1.2.3");
sin = (struct sockaddr_in *)&gsr.gsr_source;
bzero(sin, sizeof(*sin));
sin->sin_family = AF_INET;
sin->sin_addr.s_addr = inet_addr("10.4.5.6");
gsr.gsr_interface = 0;
setsockopt(s, IPPROTO_IP, MCAST_JOIN_SOURCE_GROUP, &gsr, sizeof(gsr));
setsockopt(s, IPPROTO_IP, MCAST_LEAVE_SOURCE_GROUP, &gsr, sizeof(gsr));

```

IP_MULTICAST_IF specifies a local network interface to be used when sending multicast datagrams through this socket. For example:

```

#include <arpa/inet.h>
struct in_addr addr;
addr.s_addr = inet_addr("192.1.2.3");
setsockopt(s, IPPROTO_IP, IP_MULTICAST_IF, &addr, sizeof(addr));

```

Normally, applications do not need to specify the interface. By default, multicast datagrams are sent from the interface specified by the routing configuration, namely the interface associated with the specific multicast group, with the default multicast route or with the default route. If **addr** is set to the address **INADDR_ANY**, the default interface is selected. Otherwise, **addr** should be the IP address of a local interface.

IP_MULTICAST_LOOP enables or disables loopback for multicast datagrams sent through this socket. For example:

```

unsigned char loop = 1;
setsockopt(s, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));

```

Note that the type of the *optval* parameter is **unsigned char** instead of **int**, which is common for boolean socket options. This option is provided for compatibility only. Normally, if a multicast datagram is sent to a group that the system has joined, a copy of the datagram is always looped back and delivered to any applications that are bound to the destination port. See *DEPENDENCIES* below.

IP_MULTICAST_TTL controls the scope a multicast by setting the time-to-live value for multicast datagrams sent through this socket. For example:

```

unsigned char ttl = 64;
setsockopt(s, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));

```

Note that the type of *optval* parameter is **unsigned char** instead **int**, which is common for socket options. By default, the time-to-live field (TTL) is one, which limits the multicast to the local network. If the TTL is zero, the multicast is limited to the local system (loopback). If the TTL is two, the multicast can be forwarded through at most one gateway; and so forth. Multicast datagrams can be forwarded to other networks only if there are special multicast routers on the local and intermediate networks.

DEPENDENCIES

The behavior of **IP_MULTICAST_LOOP** depends on the network driver and interface card. Normally, loopback cannot be disabled, even if **IP_MULTICAST_LOOP** is set to zero, because it occurs in the driver or in the network interface. However, if the outbound interface is **lo0** (127.0.0.1), or if **IP_MULTICAST_TTL** is set to zero, setting **IP_MULTICAST_LOOP** to zero will disable loopback for multicast datagrams sent through the socket.

ERRORS

One of the following errors may be returned if a call to **setsockopt()** or **getsockopt()** fails.

[EADDRINUSE]	The specified multicast group has been joined already on socket.
[EADDRNOTAVAIL]	The specified IP address is not a local interface address; or there is no route for the specified multicast address; or the specified multicast group has not been joined.

[EINVAL]	The parameter <i>level</i> is not IPPROTO_IP ; or <i>optval</i> is the NULL address; or the specified multicast address is not valid; or the specified source address is not valid; or the specified interface index is not valid; or the option is not valid for the current filter mode.
[ENOBUFS]	Insufficient memory is available for internal system data structures; or the number of sources in a multicast source filter exceeds the maximum number of sources allowed, as determined by the ndd tunable parameters ip_ipc_mcast_maxsrc and ip_igmp_maxsrc .
[ENOPROTOOPT]	The parameter <i>optname</i> is not a valid socket option for the IPPROTO_IP level.
[EOPNOTSUPP]	The socket type is not SOCK_DGRAM .
[ETOOMANYREFS]	An attempt to join more than IP_MAX_MEMBERSHIPS multicast groups on a socket.

AUTHOR

The socket interfaces to IP were developed by the University of California, Berkeley. Multicast extensions were developed by the Stanford University.

SEE ALSO

ndd(1M), **bind(2)**, **getsockopt(2)**, **recv(2)**, **send(2)**, **socket(2)**, **getip4sourcefilter(3N)**, **getsourcefilter(3N)**, **if_nameindex(3N)**, **setip4sourcefilter(3N)**, **setsourcefilter(3N)**, **inet(7F)**.

RFC 3678 Socket Interface Extensions for Multicast Source Filters.



| i |