NAME

ip6 - Internet Protocol version 6 (IPv6) network layer

SYNOPSIS

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

int
socket(*AF_INET6*, *SOCK_RAW*, *proto*);

DESCRIPTION

The IPv6 network layer is used by the IPv6 protocol family for transporting data. IPv6 packets contain an IPv6 header that is not provided as part of the payload contents when passed to an application. IPv6 header options affect the behavior of this protocol and may be used by high-level protocols (such as the tcp(4) and udp(4) protocols) as well as directly by "raw sockets", which process IPv6 messages at a lower-level and may be useful for developing new protocols and special-purpose applications.

Header

All IPv6 packets begin with an IPv6 header. When data received by the kernel are passed to the application, this header is not included in buffer, even when raw sockets are being used. Likewise, when data are sent to the kernel for transmit from the application, the buffer is not examined for an IPv6 header: the kernel always constructs the header. To directly access IPv6 headers from received packets and specify them as part of the buffer passed to the kernel, link-level access (bpf(4), for example) must instead be utilized.

The header has the following definition:

```
struct ip6_hdr {
     union {
          struct ip6_hdrctl {
               uint32_t ip6_un1_flow;      /* 20 bits of flow ID */
               uint16_t ip6_un1_plen;      /* payload length */
               uint8_t  ip6_un1_nxt;       /* next header */
               uint8_t  ip6_un1_hlim;      /* hop limit */
          } ip6_un1;
          uint8_t ip6_un2_vfc;    /* version and class */
     } ip6_ctlun;
     struct in6_addr ip6_src;     /* source address */
     struct in6_addr ip6_dst;     /* destination address */
} __packed;
```

```
#define ip6_vfc         ip6_ctlun.ip6_un2_vfc
#define ip6_flow    ip6_ctlun.ip6_un1.ip6_un1_flow
#define ip6_plen    ip6_ctlun.ip6_un1.ip6_un1_plen
#define ip6_nxt         ip6_ctlun.ip6_un1.ip6_un1_nxt
#define ip6_hlim    ip6_ctlun.ip6_un1.ip6_un1_hlim
#define ip6_hops    ip6_ctlun.ip6_un1.ip6_un1_hlim
```

All fields are in network-byte order. Any options specified (see *Options* below) must also be specified in network-byte order.

ip6_flow specifies the flow ID. *ip6_plen* specifies the payload length. *ip6_nxt* specifies the type of the next header. *ip6_hlim* specifies the hop limit.

The top 4 bits of *ip6_vfc* specify the class and the bottom 4 bits specify the version.

ip6_src and *ip6_dst* specify the source and destination addresses.

The IPv6 header may be followed by any number of extension headers that start with the following generic definition:

```
struct ip6_ext {
    uint8_t ip6e_nxt;
    uint8_t ip6e_len;
} __packed;
```

Options

IPv6 allows header options on packets to manipulate the behavior of the protocol. These options and other control requests are accessed with the getsockopt(2) and setsockopt(2) system calls at level IPPROTO_IPV6 and by using ancillary data in recvmsg(2) and sendmsg(2). They can be used to access most of the fields in the IPv6 header and extension headers.

The following socket options are supported:

IPV6_UNICAST_HOPS *int **
: Get or set the default hop limit header field for outgoing unicast datagrams sent on this socket.

IPV6_MULTICAST_IF *u_int **
: Get or set the interface from which multicast packets will be sent. For hosts with multiple interfaces, each multicast transmission is sent from the primary network interface. The interface is specified as its index as provided by if_nametoindex(3). A value of zero specifies

the default interface.

IPV6_MULTICAST_HOPS *int **

Get or set the default hop limit header field for outgoing multicast datagrams sent on this socket. This option controls the scope of multicast datagram transmissions.

Datagrams with a hop limit of 1 are not forwarded beyond the local network. Multicast datagrams with a hop limit of zero will not be transmitted on any network but may be delivered locally if the sending host belongs to the destination group and if multicast loopback (see below) has not been disabled on the sending socket. Multicast datagrams with a hop limit greater than 1 may be forwarded to the other networks if a multicast router (such as mrouted(8) (*ports/net/mrouted*)) is attached to the local network.

IPV6_MULTICAST_LOOP *u_int **

Get or set the status of whether multicast datagrams will be looped back for local delivery when a multicast datagram is sent to a group to which the sending host belongs.

This option improves performance for applications that may have no more than one instance on a single host (such as a router daemon) by eliminating the overhead of receiving their own transmissions. It should generally not be used by applications for which there may be more than one instance on a single host (such as a conferencing program) or for which the sender does not belong to the destination group (such as a time-querying program).

A multicast datagram sent with an initial hop limit greater than 1 may be delivered to the sending host on a different interface from that on which it was sent if the host belongs to the destination group on that other interface. The multicast loopback control option has no effect on such delivery.

IPV6_JOIN_GROUP *struct ipv6_mreq **

Join a multicast group. A host must become a member of a multicast group before it can receive datagrams sent to the group.

```
struct ipv6_mreq {
        struct in6_addr     ipv6mr_multiaddr;
        unsigned int        ipv6mr_interface;
};
```

ipv6mr_interface may be set to zeroes to choose the default multicast interface or to the index of a particular multicast-capable interface if the host is multihomed. Membership is associated with a single interface; programs running on multihomed hosts may need to join the same group on more than one interface.

If the multicast address is unspecified (i.e., all zeroes), messages from all multicast addresses will be accepted by this group. Note that setting to this value requires superuser privileges.

IPV6_LEAVE_GROUP *struct ipv6_mreq* *
Drop membership from the associated multicast group. Memberships are automatically dropped when the socket is closed or when the process exits.

IPV6_ORIGDSTADDR *int* *
Get or set whether a datagram's original destination address and port are returned as ancillary data along with the payload in subsequent recvmsg(2) calls. The information is stored in the ancillary data as a sockaddr_in6 structure.

IPV6_PORTRANGE *int* *
Get or set the allocation policy of ephemeral ports for when the kernel automatically binds a local address to this socket. The following values are available:

IPV6_PORTRANGE_DEFAULT	Use the regular range of non-reserved ports (varies, see ip(4)).
IPV6_PORTRANGE_HIGH	Use a high range (varies, see ip(4)).
IPV6_PORTRANGE_LOW	Use a low, reserved range (600-1023, see ip(4)).

IPV6_PKTINFO *int* *
Get or set whether additional information about subsequent packets will be provided as ancillary data along with the payload in subsequent recvmsg(2) calls. The information is stored in the following structure in the ancillary data returned:

```
struct in6_pktinfo {
        struct in6_addr ipi6_addr;    /* src/dst IPv6 address */
        unsigned int    ipi6_ifindex; /* send/recv if index */
};
```

IPV6_HOPLIMIT *int* *
Get or set whether the hop limit header field from subsequent packets will be provided as ancillary data along with the payload in subsequent recvmsg(2) calls. The value is stored as an *int* in the ancillary data returned.

IPV6_HOPOPTS *int* *
Get or set whether the hop-by-hop options from subsequent packets will be provided as

ancillary data along with the payload in subsequent recvmsg(2) calls. The option is stored in the following structure in the ancillary data returned:

```
struct ip6_hbh {
        uint8_t ip6h_nxt;    /* next header */
        uint8_t ip6h_len;    /* length in units of 8 octets */
/* followed by options */
} __packed;
```

The **inet6_opt_init**() routine and family of routines may be used to manipulate this data.

This option requires superuser privileges.

IPV6_DSTOPTS *int* *

Get or set whether the destination options from subsequent packets will be provided as ancillary data along with the payload in subsequent recvmsg(2) calls. The option is stored in the following structure in the ancillary data returned:

```
struct ip6_dest {
        uint8_t ip6d_nxt;    /* next header */
        uint8_t ip6d_len;    /* length in units of 8 octets */
/* followed by options */
} __packed;
```

The **inet6_opt_init**() routine and family of routines may be used to manipulate this data.

This option requires superuser privileges.

IPV6_TCLASS *int* *

Get or set the value of the traffic class field used for outgoing datagrams on this socket. The value must be between -1 and 255. A value of -1 resets to the default value.

IPV6_RECVTCLASS *int* *

Get or set the status of whether the traffic class header field will be provided as ancillary data along with the payload in subsequent recvmsg(2) calls. The header field is stored as a single value of type *int*.

IPV6_RTHDR *int* *

Get or set whether the routing header from subsequent packets will be provided as ancillary data along with the payload in subsequent recvmsg(2) calls. The header is stored in the following structure in the ancillary data returned:

```
struct ip6_rthdr {
        uint8_t ip6r_nxt;     /* next header */
        uint8_t ip6r_len;     /* length in units of 8 octets */
        uint8_t ip6r_type;    /* routing type */
        uint8_t ip6r_segleft;         /* segments left */
/* followed by routing-type-specific data */
} __packed;
```

The **inet6_opt_init**() routine and family of routines may be used to manipulate this data.

This option requires superuser privileges.

IPV6_PKTOPTIONS *struct cmsghdr **

Get or set all header options and extension headers at one time on the last packet sent or received on the socket. All options must fit within the size of an mbuf (see mbuf(9)). Options are specified as a series of *cmsghdr* structures followed by corresponding values. *cmsg_level* is set to IPPROTO_IPV6, *cmsg_type* to one of the other values in this list, and trailing data to the option value. When setting options, if the length *optlen* to setsockopt(2) is zero, all header options will be reset to their default values. Otherwise, the length should specify the size the series of control messages consumes.

Instead of using sendmsg(2) to specify option values, the ancillary data used in these calls that correspond to the desired header options may be directly specified as the control message in the series of control messages provided as the argument to setsockopt(2).

IPV6_CHECKSUM *int **

Get or set the byte offset into a packet where the 16-bit checksum is located. When set, this byte offset is where incoming packets will be expected to have checksums of their data stored and where outgoing packets will have checksums of their data computed and stored by the kernel. A value of -1 specifies that no checksums will be checked on incoming packets and that no checksums will be computed or stored on outgoing packets. The offset of the checksum for ICMPv6 sockets cannot be relocated or turned off.

IPV6_V6ONLY *int **

Get or set whether only IPv6 connections can be made to this socket. For wildcard sockets, this can restrict connections to IPv6 only.

IPV6_USE_MIN_MTU *int **

Get or set whether the minimal IPv6 maximum transmission unit (MTU) size will be used to avoid fragmentation from occurring for subsequent outgoing datagrams.

IPV6_AUTH_LEVEL *int **
: Get or set the ipsec(4) authentication level.

IPV6_ESP_TRANS_LEVEL *int **
: Get or set the ESP transport level.

IPV6_ESP_NETWORK_LEVEL *int **
: Get or set the ESP encapsulation level.

IPV6_IPCOMP_LEVEL *int **
: Get or set the ipcomp(4) level.

The IPV6_PKTINFO, IPV6_HOPLIMIT, IPV6_HOPOPTS, IPV6_DSTOPTS, IPV6_RTHDR, and IPV6_ORIGDSTADDR options will return ancillary data along with payload contents in subsequent recvmsg(2) calls with *cmsg_level* set to IPPROTO_IPV6 and *cmsg_type* set to respective option name value (e.g., IPV6_HOPTLIMIT). Some of these options may also be used directly as ancillary *cmsg_type* values in sendmsg(2) to set options on the packet being transmitted by the call. The *cmsg_level* value must be IPPROTO_IPV6. For these options, the ancillary data object value format is the same as the value returned as explained for each when received with recvmsg(2).

Note that using sendmsg(2) to specify options on particular packets works only on UDP and raw sockets. To manipulate header options for packets on TCP sockets, only the socket options may be used.

In some cases, there are multiple APIs defined for manipulating an IPv6 header field. A good example is the outgoing interface for multicast datagrams, which can be set by the IPV6_MULTICAST_IF socket option, through the IPV6_PKTINFO option, and through the *sin6_scope_id* field of the socket address passed to the sendto(2) system call.

Resolving these conflicts is implementation dependent. This implementation determines the value in the following way: options specified by using ancillary data (i.e., sendmsg(2)) are considered first, options specified by using IPV6_PKTOPTIONS to set "sticky" options are considered second, options specified by using the individual, basic, and direct socket options (e.g., IPV6_UNICAST_HOPS) are considered third, and options specified in the socket address supplied to sendto(2) are the last choice.

Multicasting

IPv6 multicasting is supported only on AF_INET6 sockets of type SOCK_DGRAM and SOCK_RAW, and only on networks where the interface driver supports multicasting. Socket options (see above) that manipulate membership of multicast groups and other multicast options include IPV6_MULTICAST_IF, IPV6_MULTICAST_HOPS, IPV6_MULTICAST_LOOP, IPV6_LEAVE_GROUP, and IPV6_JOIN_GROUP.

Raw Sockets

Raw IPv6 sockets are connectionless and are normally used with the sendto(2) and recvfrom(2) calls, although the connect(2) call may be used to fix the destination address for future outgoing packets so that send(2) may instead be used and the bind(2) call may be used to fix the source address for future outgoing packets instead of having the kernel choose a source address.

By using connect(2) or bind(2), raw socket input is constrained to only packets with their source address matching the socket destination address if connect(2) was used and to packets with their destination address matching the socket source address if bind(2) was used.

If the *proto* argument to socket(2) is zero, the default protocol (IPPROTO_RAW) is used for outgoing packets. For incoming packets, protocols recognized by kernel are **not** passed to the application socket (e.g., tcp(4) and udp(4)) except for some ICMPv6 messages. The ICMPv6 messages not passed to raw sockets include echo, timestamp, and address mask requests. If *proto* is non-zero, only packets with this protocol will be passed to the socket.

IPv6 fragments are also not passed to application sockets until they have been reassembled. If reception of all packets is desired, link-level access (such as bpf(4)) must be used instead.

Outgoing packets automatically have an IPv6 header prepended to them (based on the destination address and the protocol number the socket was created with). Incoming packets are received by an application without the IPv6 header or any extension headers.

Outgoing packets will be fragmented automatically by the kernel if they are too large. Incoming packets will be reassembled before being sent to the raw socket, so packet fragments or fragment headers will never be seen on a raw socket.

EXAMPLES

The following determines the hop limit on the next packet received:

```
struct iovec iov[2];
u_char buf[BUFSIZ];
struct cmsghdr *cm;
struct msghdr m;
int optval;
```

```
bool found;
u_char data[2048];

/* Create socket. */

(void)memset(&m, 0, sizeof(m));
(void)memset(&iov, 0, sizeof(iov));

iov[0].iov_base = data;             /* buffer for packet payload */
iov[0].iov_len = sizeof(data); /* expected packet length */

m.msg_name = &from;                 /* sockaddr_in6 of peer */
m.msg_namelen = sizeof(from);
m.msg_iov = iov;
m.msg_iovlen = 1;
m.msg_control = (caddr_t)buf;       /* buffer for control messages */
m.msg_controllen = sizeof(buf);

/*
 * Enable the hop limit value from received packets to be
 * returned along with the payload.
 */
optval = 1;
if (setsockopt(s, IPPROTO_IPV6, IPV6_HOPLIMIT, &optval,
    sizeof(optval)) == -1)
        err(1, "setsockopt");

found = false;
do {
        if (recvmsg(s, &m, 0) == -1)
                err(1, "recvmsg");
        for (cm = CMSG_FIRSTHDR(&m); cm != NULL;
            cm = CMSG_NXTHDR(&m, cm)) {
                if (cm->cmsg_level == IPPROTO_IPV6 &&
                    cm->cmsg_type == IPV6_HOPLIMIT &&
                    cm->cmsg_len == CMSG_LEN(sizeof(int))) {
                        found = true;
                        (void)printf("hop limit: %d\n",
                            *(int *)CMSG_DATA(cm));
                        break;
                }
        }
} while (!found);
```

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one or when trying to send a datagram with the destination address specified and the socket is already connected.
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket has not been connected.
[ENOBUFS]	when the system runs out of memory for an internal data structure.
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.
[EACCES]	when an attempt is made to create a raw IPv6 socket by a non-privileged process.

The following errors specific to IPv6 may occur when setting or getting header options:

[EINVAL]	An unknown socket option name was given.
[EINVAL]	An ancillary data object was improperly formed.

SEE ALSO

getsockopt(2), recv(2), send(2), setsockopt(2), socket(2), CMSG_DATA(3), if_nametoindex(3), inet6_opt_init(3), bpf(4), icmp6(4), inet6(4), ip(4), netintro(4), tcp(4), udp(4)

W. Stevens and M. Thomas, *Advanced Sockets API for IPv6*, RFC 2292, February 1998.

S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 2460, December 1998.

R. Gilligan, S. Thomson, J. Bound, and W. Stevens, *Basic Socket Interface Extensions for IPv6*, RFC 2553, March 1999.

R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens, *Basic Socket Interface Extensions for IPv6*, RFC 3493, February 2003.

W. Stevens, M. Thomas, E. Nordmark, and T. Jinmei, *Advanced Sockets Application Program Interface (API) for IPv6*, RFC 3542, May 2003.

S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 8200, July 2017.

W. Stevens, B. Fenner, and A. Rudoff, *UNIX Network Programming, 3rd Edition*, *Addison-Wesley Professional*, November 2003.

STANDARDS

Most of the socket options are defined in RFC 2292 / 3542 or RFC 2553 / 3493. The IPV6_PORTRANGE socket option and the conflict resolution rule are not defined in the RFCs and should be considered implementation dependent.