# Rocky Enterprise Linux 9.2 Manual Pages on command 'ip.7'

*$ man ip.7*

IP(7)                              Linux Programmer's Manual                              IP(7)

NAME

   ip - Linux IPv4 protocol implementation

SYNOPSIS

   #include <sys/socket.h>

   #include <netinet/in.h>

   #include <netinet/ip.h> /* superset of previous */

   tcp_socket = socket(AF_INET, SOCK_STREAM, 0);

   udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

   raw_socket = socket(AF_INET, SOCK_RAW, protocol);

DESCRIPTION

   Linux  implements the Internet Protocol, version 4, described in RFC 791 and RFC 1122.  ip

   contains a level 2 multicasting implementation conforming to RFC 1112.  It  also  contains

   an IP router including a packet filter.

   The programming interface is BSD-sockets compatible.  For more information on sockets, see

   socket(7).

   An IP socket is created using socket(2):

      socket(AF_INET, socket_type, protocol);

   Valid socket types include SOCK_STREAM to open a stream socket, SOCK_DGRAM to open a data?

   gram socket, and SOCK_RAW to open a raw(7) socket to access the IP protocol directly.

   protocol  is  the  IP  protocol in the IP header to be received or sent.  Valid values for

   protocol include:

   ? 0 and IPPROTO_TCP for tcp(7) stream sockets;

? 0 and IPPROTO_UDP for udp(7) datagram sockets;

? IPPROTO_SCTP for sctp(7) stream sockets; and

? IPPROTO_UDPLITE for udplite(7) datagram sockets.

For SOCK_RAW you may specify a valid IANA IP protocol defined in  RFC 1700  assigned  num?
bers.

When  a  process  wants  to  receive new incoming packets or connections, it should bind a
socket to a local interface address using bind(2).  In this case, only one IP  socket  may
be  bound  to  any  given local (address, port) pair.  When INADDR_ANY is specified in the
bind call, the socket will be bound to all local interfaces.  When listen(2) is called  on
an  unbound socket, the socket is automatically bound to a random free port with the local
address set to INADDR_ANY.  When connect(2) is called on an unbound socket, the socket  is
automatically  bound  to  a random free port or to a usable shared port with the local ad?
dress set to INADDR_ANY.

A TCP local socket address that has been bound is unavailable for some time after closing,
unless  the  SO_REUSEADDR flag has been set.  Care should be taken when using this flag as
it makes TCP less reliable.

Address format

An IP socket address is defined as a combination of an IP interface address and  a  16-bit
port  number.  The basic IP protocol does not supply port numbers, they are implemented by
higher level protocols like udp(7) and tcp(7).  On raw sockets sin_port is set to  the  IP
protocol.

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};
/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

sin_family is always set to AF_INET.  This is required; in Linux 2.2 most networking func?
tions return EINVAL when this setting is missing.  sin_port contains the port  in  network
byte  order.   The  port numbers below 1024 are called privileged ports (or sometimes: re?

served ports).  Only a privileged process (on Linux: a process  that  has  the

CAP_NET_BIND_SERVICE capability in the user namespace governing its network namespace) may

bind(2) to these sockets.  Note that the raw IPv4 protocol as such has  no  concept  of  a

port, they are implemented only by higher protocols like tcp(7) and udp(7).

sin_addr  is  the  IP host address.  The s_addr member of struct in_addr contains the host

interface address in network byte order.  in_addr should be assigned one of  the  INADDR_*

values (e.g., INADDR_LOOPBACK) using htonl(3) or set using the inet_aton(3), inet_addr(3),

inet_makeaddr(3) library functions or directly with  the  name  resolver  (see  gethostby?

name(3)).

IPv4  addresses are divided into unicast, broadcast, and multicast addresses.  Unicast ad?

dresses specify a single interface of a host, broadcast addresses specify all hosts  on  a

network,  and  multicast  addresses  address all hosts in a multicast group.  Datagrams to

broadcast addresses can be sent or received only when the SO_BROADCAST socket flag is set.

In the current implementation, connection-oriented sockets are allowed to use only unicast

addresses.

Note that the address and the port are always stored in network byte order.   In  particu?

lar,  this  means that you need to call htons(3) on the number that is assigned to a port.

All address/port manipulation functions in the standard library work in network  byte  or?

der.

There  are several special addresses: INADDR_LOOPBACK (127.0.0.1) always refers to the lo?

cal host via the loopback device; INADDR_ANY (0.0.0.0) means any address for binding;  IN?

ADDR_BROADCAST  (255.255.255.255)  means  any  host and has the same effect on bind as IN?

ADDR_ANY for historical reasons.

Socket options

IP supports some protocol-specific socket options that can be set with  setsockopt(2)  and

read with getsockopt(2).  The socket option level for IP is IPPROTO_IP.  A boolean integer

flag is zero when it is false, otherwise true.

When an invalid socket option is specified, getsockopt(2) and setsockopt(2) fail with  the

error ENOPROTOOPT.

IP_ADD_MEMBERSHIP (since Linux 1.2)

Join a multicast group.  Argument is an ip_mreqn structure.

struct ip_mreqn {

struct in_addr imr_multiaddr; /* IP multicast group

```
                          address */
    struct in_addr imr_address;   /* IP address of local
                          interface */
    int        imr_ifindex;   /* interface index */
};
```

imr_multiaddr contains the address of the multicast group the application wants to join or leave.  It must be a valid multicast address (or setsockopt(2) fails with the error  EIN? VAL).  imr_address is the address of the local interface with which the system should join the multicast group; if it is equal to INADDR_ANY, an appropriate interface is  chosen  by the  system.   imr_ifindex  is the interface index of the interface that should join/leave the imr_multiaddr group, or 0 to indicate any interface.

The ip_mreqn structure is available only since Linux 2.2.  For  compatibility,  the old ip_mreq structure (present since Linux 1.2) is still supported; it differs from ip_mreqn only by not including the imr_ifindex field.  (The kernel determines which structure is being passed based on the size passed in optlen.)

IP_ADD_MEMBERSHIP is valid only for setsockopt(2).

IP_ADD_SOURCE_MEMBERSHIP (since Linux 2.4.22 / 2.5.68)

Join  a multicast group and allow receiving data only from a specified source.  Ar? gument is an ip_mreq_source structure.

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

The ip_mreq_source structure is similar to  ip_mreqn  described  under  IP_ADD_MEMBERSHIP. The  imr_multiaddr field contains the address of the multicast group the application wants to join or leave.  The imr_interface field is the address  of  the  local  interface  with which  the system should join the multicast group.  Finally, the imr_sourceaddr field con? tains the address of the source the application wants to receive data from.

This option can be used multiple times to allow receiving data from more  than  one

source.

IP_BIND_ADDRESS_NO_PORT (since Linux 4.2)

Inform the kernel to not reserve an ephemeral port when using bind(2) with a port
number of 0.  The port will later be automatically chosen at connect(2) time, in  a
way that allows sharing a source port as long as the 4-tuple is unique.

IP_BLOCK_SOURCE (since Linux 2.4.22 / 2.5.68)

Stop  receiving  multicast  data  from a specific source in a given group.  This is
valid only after the application has subscribed to the multicast group using either
IP_ADD_MEMBERSHIP or IP_ADD_SOURCE_MEMBERSHIP.
Argument  is  an  ip_mreq_source structure as described under IP_ADD_SOURCE_MEMBER?
SHIP.

IP_DROP_MEMBERSHIP (since Linux 1.2)

Leave a multicast group.  Argument is an ip_mreqn or ip_mreq structure  similar  to
IP_ADD_MEMBERSHIP.

IP_DROP_SOURCE_MEMBERSHIP (since Linux 2.4.22 / 2.5.68)

Leave  a  source-specific group?that is, stop receiving data from a given multicast
group that come from a given source.  If the application has subscribed to multiple
sources within the same group, data from the remaining sources will still be deliv?
ered.  To stop receiving data from all sources at once, use IP_DROP_MEMBERSHIP.
Argument is an ip_mreq_source structure as  described  under  IP_ADD_SOURCE_MEMBER?
SHIP.

IP_FREEBIND (since Linux 2.4)

If enabled, this boolean option allows binding to an IP address that is nonlocal or
does not (yet) exist.  This permits listening on a socket,  without  requiring  the
underlying  network  interface  or the specified dynamic IP address to be up at the
time that the application is trying to bind to it.  This option is  the  per-socket
equivalent of the ip_nonlocal_bind /proc interface described below.

IP_HDRINCL (since Linux 2.0)

If  enabled,  the user supplies an IP header in front of the user data.  Valid only
for SOCK_RAW sockets; see raw(7) for more information.  When this flag is  enabled,
the values set by IP_OPTIONS, IP_TTL, and IP_TOS are ignored.

IP_MSFILTER (since Linux 2.4.22 / 2.5.68)

This  option provides access to the advanced full-state filtering API.  Argument is

an ip_msfilter structure.

```
struct ip_msfilter {
    struct in_addr imsf_multiaddr; /* IP multicast group
                            address */
    struct in_addr imsf_interface; /* IP address of local
                            interface */
    uint32_t      imsf_fmode;    /* Filter-mode */
    uint32_t      imsf_numsrc;   /* Number of sources in
                            the following array */
    struct in_addr imsf_slist[1];  /* Array of source
                            addresses */
};
```

There are two macros, MCAST_INCLUDE and MCAST_EXCLUDE, which can be used to specify the filtering mode. Additionally, the IP_MSFILTER_SIZE(n) macro exists to determine how much memory is needed to store ip_msfilter structure with n sources in the source list.

For the full description of multicast source filtering refer to RFC 3376.

IP_MTU (since Linux 2.2)

Retrieve the current known path MTU of the current socket. Returns an integer. IP_MTU is valid only for getsockopt(2) and can be employed only when the socket has been connected.

IP_MTU_DISCOVER (since Linux 2.2)

Set or receive the Path MTU Discovery setting for a socket. When enabled, Linux will perform Path MTU Discovery as defined in RFC 1191 on SOCK_STREAM sockets. For non-SOCK_STREAM sockets, IP_PMTUDISC_DO forces the don't-fragment flag to be set on all outgoing packets. It is the user's responsibility to packetize the data in MTU-sized chunks and to do the retransmits if necessary. The kernel will reject (with EMSGSIZE) datagrams that are bigger than the known path MTU. IP_PMTUD? ISC_WANT will fragment a datagram if needed according to the path MTU, or will set the don't-fragment flag otherwise.

The system-wide default can be toggled between IP_PMTUDISC_WANT and IP_PMTUD? ISC_DONT by writing (respectively, zero and nonzero values) to the /proc/sys/net/ipv4/ip_no_pmtu_disc file.

Path MTU discovery value   Meaning

IP_PMTUDISC_WANT        Use per-route settings.

IP_PMTUDISC_DONT        Never do Path MTU Discovery.

IP_PMTUDISC_DO        Always do Path MTU Discovery.

IP_PMTUDISC_PROBE        Set DF but ignore Path MTU.

When PMTU discovery is enabled, the kernel automatically keeps track of the path MTU per destination host. When it is connected to a specific peer with connect(2), the currently known path MTU can be retrieved conveniently using the IP_MTU socket option (e.g., after an EMSGSIZE error occurred). The path MTU may change over time. For connectionless sockets with many destinations, the new MTU for a given destination can also be accessed using the error queue (see IP_RECVERR). A new er‐ ror will be queued for every incoming MTU update.

While MTU discovery is in progress, initial packets from datagram sockets may be dropped. Applications using UDP should be aware of this and not take it into ac‐ count for their packet retransmit strategy.

To bootstrap the path MTU discovery process on unconnected sockets, it is possible to start with a big datagram size (headers up to 64 kilobytes long) and let it shrink by updates of the path MTU.

To get an initial estimate of the path MTU, connect a datagram socket to the desti‐ nation address using connect(2) and retrieve the MTU by calling getsockopt(2) with the IP_MTU option.

It is possible to implement RFC 4821 MTU probing with SOCK_DGRAM or SOCK_RAW sock‐ ets by setting a value of IP_PMTUDISC_PROBE (available since Linux 2.6.22). This is also particularly useful for diagnostic tools such as tracepath(8) that wish to deliberately send probe packets larger than the observed Path MTU.

IP_MULTICAST_ALL (since Linux 2.6.31)

This option can be used to modify the delivery policy of multicast messages to sockets bound to the wildcard INADDR_ANY address. The argument is a boolean inte‐ ger (defaults to 1). If set to 1, the socket will receive messages from all the groups that have been joined globally on the whole system. Otherwise, it will de‐ liver messages only from the groups that have been explicitly joined (for example via the IP_ADD_MEMBERSHIP option) on this particular socket.

IP_MULTICAST_IF (since Linux 1.2)

Set the local device for a multicast socket. The argument for setsockopt(2) is an

ip_mreqn or (since Linux 3.5) ip_mreq structure similar to IP_ADD_MEMBERSHIP, or an
in_addr structure.  (The kernel determines which structure is being passed based on
the  size  passed in optlen.)  For getsockopt(2), the argument is an in_addr struc‐
ture.

IP_MULTICAST_LOOP (since Linux 1.2)

Set or read a boolean integer argument that determines whether sent multicast pack‐
ets should be looped back to the local sockets.

IP_MULTICAST_TTL (since Linux 1.2)

Set  or  read the time-to-live value of outgoing multicast packets for this socket.
It is very important for multicast packets to set the smallest TTL  possible.   The
default  is  1 which means that multicast packets don't leave the local network un‐
less the user program explicitly requests it.  Argument is an integer.

IP_NODEFRAG (since Linux 2.6.36)

If enabled (argument is nonzero), the reassembly of outgoing packets is disabled in
the netfilter layer.  The argument is an integer.

This option is valid only for SOCK_RAW sockets.

IP_OPTIONS (since Linux 2.0)

Set or get the IP options to be sent with every packet from this socket.  The argu‐
ments are a pointer to a memory  buffer  containing  the  options  and  the  option
length.   The setsockopt(2) call sets the IP options associated with a socket.  The
maximum option size for IPv4 is 40 bytes.  See RFC 791  for  the  allowed  options.
When the initial connection request packet for a SOCK_STREAM socket contains IP op‐
tions, the IP options will be set automatically to the  options  from  the  initial
packet  with  routing headers reversed.  Incoming packets are not allowed to change
options after the connection is established.  The processing of all incoming source
routing  options  is  disabled  by  default  and  can  be  enabled by using the ac‐
cept_source_route /proc interface.  Other options like timestamps  are  still  han‐
dled.  For datagram sockets, IP options can be set only by the local user.  Calling
getsockopt(2) with IP_OPTIONS puts the current IP options used for sending into the
supplied buffer.

IP_PASSSEC (since Linux 2.6.17)

If labeled IPSEC or NetLabel is configured on the sending and receiving hosts, this
option enables receiving of the security context of the peer socket in an ancillary

message of type SCM_SECURITY retrieved using recvmsg(2). This option is supported only for UDP sockets; for TCP or SCTP sockets, see the description of the SO_PEER? SEC option below.

The value given as an argument to setsockopt(2) and returned as the result of get? sockopt(2) is an integer boolean flag.

The security context returned in the SCM_SECURITY ancillary message is of the same format as the one described under the SO_PEERSEC option below.

Note: the reuse of the SCM_SECURITY message type for the IP_PASSSEC socket option was likely a mistake, since other IP control messages use their own numbering scheme in the IP namespace and often use the socket option value as the message type. There is no conflict currently since the IP option with the same value as SCM_SECURITY is IP_HDRINCL and this is never used for a control message type.

IP_PKTINFO (since Linux 2.2)

Pass an IP_PKTINFO ancillary message that contains a pktinfo structure that sup? plies some information about the incoming packet. This works only for datagram oriented sockets. The argument is a flag that tells the socket whether the IP_PKT? INFO message should be passed or not. The message itself can be sent/retrieved only as a control message with a packet using recvmsg(2) or sendmsg(2).

```
struct in_pktinfo {
    unsigned int   ipi_ifindex;  /* Interface index */
    struct in_addr ipi_spec_dst; /* Local address */
    struct in_addr ipi_addr;     /* Header Destination
                          address */
};
```

ipi_ifindex is the unique index of the interface the packet was received on. ipi_spec_dst is the local address of the packet and ipi_addr is the destination ad? dress in the packet header. If IP_PKTINFO is passed to sendmsg(2) and ipi_spec_dst is not zero, then it is used as the local source address for the routing table lookup and for setting up IP source route options. When ipi_ifindex is not zero, the primary local address of the interface specified by the index overwrites ipi_spec_dst for the routing table lookup.

IP_RECVERR (since Linux 2.2)

Enable extended reliable error message passing. When enabled on a datagram socket,

all generated errors will be queued in a per-socket error queue.  When the user re‐

ceives  an  error  from  a  socket operation, the errors can be received by calling

recvmsg(2) with the MSG_ERRQUEUE flag set.   The  sock_extended_err  structure  de‐

scribing  the error will be passed in an ancillary message with the type IP_RECVERR

and the level IPPROTO_IP.  This is useful for reliable  error  handling  on  uncon‐

nected  sockets.   The  received data portion of the error queue contains the error

packet.

The IP_RECVERR control message contains a sock_extended_err structure:

    #define SO_EE_ORIGIN_NONE    0

    #define SO_EE_ORIGIN_LOCAL   1

    #define SO_EE_ORIGIN_ICMP    2

    #define SO_EE_ORIGIN_ICMP6   3

    struct sock_extended_err {

       uint32_t ee_errno;   /* error number */

       uint8_t  ee_origin;  /* where the error originated */

       uint8_t  ee_type;    /* type */

       uint8_t  ee_code;    /* code */

       uint8_t  ee_pad;

       uint32_t ee_info;    /* additional information */

       uint32_t ee_data;    /* other data */

       /* More data may follow */

    };

    struct sockaddr *SO_EE_OFFENDER(struct sock_extended_err *);

ee_errno contains the errno number of the queued error.  ee_origin  is  the  origin

code  of  where the error originated.  The other fields are protocol-specific.  The

macro SO_EE_OFFENDER returns a pointer to the address of the network  object  where

the  error  originated  from given a pointer to the ancillary message.  If this ad‐

dress is not known, the sa_family member of the sockaddr contains AF_UNSPEC and the

other fields of the sockaddr are undefined.

IP  uses the sock_extended_err structure as follows: ee_origin is set to SO_EE_ORI‐

GIN_ICMP for errors received as an ICMP packet, or SO_EE_ORIGIN_LOCAL  for  locally

generated  errors.   Unknown values should be ignored.  ee_type and ee_code are set

from the type and code fields of the ICMP header.  ee_info contains the  discovered

MTU for EMSGSIZE errors. The message also contains the sockaddr_in of the node caused the error, which can be accessed with the SO_EE_OFFENDER macro. The sin_family field of the SO_EE_OFFENDER address is AF_UNSPEC when the source was un? known. When the error originated from the network, all IP options (IP_OPTIONS, IP_TTL, etc.) enabled on the socket and contained in the error packet are passed as control messages. The payload of the packet causing the error is returned as nor? mal payload. Note that TCP has no error queue; MSG_ERRQUEUE is not permitted on SOCK_STREAM sockets. IP_RECVERR is valid for TCP, but all errors are returned by socket function return or SO_ERROR only.

For raw sockets, IP_RECVERR enables passing of all received ICMP errors to the ap? plication, otherwise errors are reported only on connected sockets

It sets or retrieves an integer boolean flag. IP_RECVERR defaults to off.

IP_RECVOPTS (since Linux 2.2)

Pass all incoming IP options to the user in a IP_OPTIONS control message. The routing header and other options are already filled in for the local host. Not supported for SOCK_STREAM sockets.

IP_RECVORIGDSTADDR (since Linux 2.6.29)

This boolean option enables the IP_ORIGDSTADDR ancillary message in recvmsg(2), in which the kernel returns the original destination address of the datagram being re? ceived. The ancillary message contains a struct sockaddr_in.

IP_RECVTOS (since Linux 2.2)

If enabled, the IP_TOS ancillary message is passed with incoming packets. It con? tains a byte which specifies the Type of Service/Precedence field of the packet header. Expects a boolean integer flag.

IP_RECVTTL (since Linux 2.2)

When this flag is set, pass a IP_TTL control message with the time-to-live field of the received packet as a 32 bit integer. Not supported for SOCK_STREAM sockets.

IP_RETOPTS (since Linux 2.2)

Identical to IP_RECVOPTS, but returns raw unprocessed options with timestamp and route record options not filled in for this hop.

IP_ROUTER_ALERT (since Linux 2.2)

Pass all to-be forwarded packets with the IP Router Alert option set to this socket. Valid only for raw sockets. This is useful, for instance, for user-space

RSVP daemons.  The tapped packets are not forwarded by the kernel; it is the user's responsibility to send them out again.  Socket binding is ignored, such packets are filtered only by protocol.  Expects an integer flag.

IP_TOS (since Linux 1.0)

Set or receive the Type-Of-Service (TOS) field that is sent with  every  IP  packet originating  from  this  socket.   It is used to prioritize packets on the network. TOS is a byte.  There are some standard TOS flags defined: IPTOS_LOWDELAY to  mini‐ mize  delays  for interactive traffic, IPTOS_THROUGHPUT to optimize throughput, IP‐ TOS_RELIABILITY to optimize for  reliability,  IPTOS_MINCOST  should  be  used  for "filler  data"  where  slow  transmission doesn't matter.  At most one of these TOS values can be specified.  Other bits are invalid and shall be cleared.  Linux sends IPTOS_LOWDELAY  datagrams  first  by default, but the exact behavior depends on the configured queueing discipline.  Some high-priority levels  may  require  superuser privileges (the CAP_NET_ADMIN capability).

IP_TRANSPARENT (since Linux 2.6.24)

Setting  this  boolean  option  enables  transparent proxying on this socket.  This socket option allows the calling application to bind to a nonlocal IP  address  and operate  both  as  a client and a server with the foreign address as the local end‐ point.  NOTE: this requires that routing be set up in a way that packets  going  to the foreign address are routed through the TProxy box (i.e., the system hosting the application that employs the IP_TRANSPARENT socket option).  Enabling  this  socket option requires superuser privileges (the CAP_NET_ADMIN capability). TProxy  redirection  with the iptables TPROXY target also requires that this option be set on the redirected socket.

IP_TTL (since Linux 1.0)

Set or retrieve the current time-to-live field that is used in  every  packet  sent from this socket.

IP_UNBLOCK_SOURCE (since Linux 2.4.22 / 2.5.68)

Unblock  previously  blocked  multicast  source.   Returns EADDRNOTAVAIL when given source is not being blocked. Argument is an ip_mreq_source structure as  described  under  IP_ADD_SOURCE_MEMBER‐ SHIP.

SO_PEERSEC (since Linux 2.6.17)

If labeled IPSEC or NetLabel is configured on both the sending and receiving hosts, this read-only socket option returns the security context of the peer  socket  con‐ nected  to  this socket.  By default, this will be the same as the security context of the process that created the peer socket unless overridden by the policy or by a process with the required permissions.

The  argument  to getsockopt(2) is a pointer to a buffer of the specified length in bytes into which the security context string will be copied.  If the buffer  length is  less than the length of the security context string, then getsockopt(2) returns -1, sets errno to ERANGE, and returns the required length via optlen.   The  caller should  allocate at least NAME_MAX bytes for the buffer initially, although this is not guaranteed to be sufficient.  Resizing the buffer to the  returned  length  and retrying may be necessary.

The  security  context  string  may include a terminating null character in the re‐ turned length, but is not guaranteed to do so: a security context  "foo"  might  be represented  as either {'f','o','o'} of length 3 or {'f','o','o','\0'} of length 4, which are considered to be interchangeable.  The string is printable, does not con‐ tain non-terminating null characters, and is in an unspecified encoding (in partic‐ ular, it is not guaranteed to be ASCII or UTF-8).

The use of this option for sockets in the AF_INET address family is supported since Linux 2.6.17 for TCP sockets, and since Linux 4.17 for SCTP sockets.

For  SELinux,  NetLabel conveys only the MLS portion of the security context of the peer across the wire, defaulting the rest of the security context to the values de‐ fined  in  the  policy  for the netmsg initial security identifier (SID).  However, NetLabel can be configured to pass full security contexts over  loopback.   Labeled IPSEC always passes full security contexts as part of establishing the security as‐ sociation (SA) and looks them up based on the association for each packet.

/proc interfaces

The IP protocol supports a set of /proc interfaces to configure  some  global  parameters. The  parameters  can  be  accessed  by  reading  or  writing  files  in  the  directory /proc/sys/net/ipv4/.  Interfaces described as Boolean take an integer value, with  a  non‐ zero  value  ("true")  meaning  that the corresponding option is enabled, and a zero value ("false") meaning that the option is disabled.

ip_always_defrag (Boolean; since Linux 2.2.13)

[New with kernel 2.2.13; in earlier kernel versions this feature was controlled at compile time by the CONFIG_IP_ALWAYS_DEFRAG option; this option is not present in 2.4.x and later]

When this boolean flag is enabled (not equal 0), incoming fragments (parts of IP packets that arose when some host between origin and destination decided that the packets were too large and cut them into pieces) will be reassembled (defragmented) before being processed, even if they are about to be forwarded.

Enable only if running either a firewall that is the sole link to your network or a transparent proxy; never ever use it for a normal router or host. Otherwise, frag? mented communication can be disturbed if the fragments travel over different links. Defragmentation also has a large memory and CPU time cost.

This is automagically turned on when masquerading or transparent proxying are con? figured.

ip_autoconfig (since Linux 2.2 to 2.6.17)

Not documented.

ip_default_ttl (integer; default: 64; since Linux 2.2)

Set the default time-to-live value of outgoing packets. This can be changed per socket with the IP_TTL option.

ip_dynaddr (Boolean; default: disabled; since Linux 2.0.31)

Enable dynamic socket address and masquerading entry rewriting on interface address change. This is useful for dialup interface with changing IP addresses. 0 means no rewriting, 1 turns it on and 2 enables verbose mode.

ip_forward (Boolean; default: disabled; since Linux 1.2)

Enable IP forwarding with a boolean flag. IP forwarding can be also set on a per-interface basis.

ip_local_port_range (since Linux 2.2)

This file contains two integers that define the default local port range allocated to sockets that are not explicitly bound to a port number?that is, the range used for ephemeral ports. An ephemeral port is allocated to a socket in the following circumstances:

*  the port number in a socket address is specified as 0 when calling bind(2);

*  listen(2) is called on a stream socket that was not previously bound;

*  connect(2) was called on a socket that was not previously bound;

* sendto(2) is called on a datagram socket that was not previously bound.

Allocation of ephemeral ports starts with the first number  in  ip_local_port_range and  ends  with  the  second number.  If the range of ephemeral ports is exhausted, then the relevant system call returns an error (but see BUGS).

Note that the port range in ip_local_port_range should not conflict with the  ports used  by  masquerading (although the case is handled).  Also, arbitrary choices may cause problems with some firewall packet filters that make  assumptions  about  the local ports in use.  The first number should be at least greater than 1024, or bet? ter, greater than 4096, to avoid clashes with well  known  ports  and  to  minimize firewall problems.

ip_no_pmtu_disc (Boolean; default: disabled; since Linux 2.2)

If  enabled, don't do Path MTU Discovery for TCP sockets by default.  Path MTU dis? covery may fail if misconfigured firewalls (that drop all ICMP packets) or  miscon? figured  interfaces (e.g., a point-to-point link where the both ends don't agree on the MTU) are on the path.  It is better to fix the broken routers on the path  than to turn off Path MTU Discovery globally, because not doing it incurs a high cost to the network.

ip_nonlocal_bind (Boolean; default: disabled; since Linux 2.4)

If set, allows processes to bind(2) to nonlocal IP addresses, which  can  be  quite useful, but may break some applications.

ip6frag_time (integer; default: 30)

Time in seconds to keep an IPv6 fragment in memory.

ip6frag_secret_interval (integer; default: 600)

Regeneration interval (in seconds) of the hash secret (or lifetime for the hash se? cret) for IPv6 fragments.

ipfrag_high_thresh (integer), ipfrag_low_thresh (integer)

If the amount of queued IP  fragments  reaches  ipfrag_high_thresh,  the  queue  is pruned down to ipfrag_low_thresh.  Contains an integer with the number of bytes.

neigh/*

See arp(7).

Ioctls

All ioctls described in socket(7) apply to ip.

Ioctls to configure generic device parameters are described in netdevice(7).

## ERRORS

**EACCES** The user tried to execute an operation without the necessary permissions. These include: sending a packet to a broadcast address without having the SO_BROADCAST flag set; sending a packet via a prohibit route; modifying firewall settings with‐ out superuser privileges (the CAP_NET_ADMIN capability); binding to a privileged port without superuser privileges (the CAP_NET_BIND_SERVICE capability).

**EADDRINUSE**

Tried to bind to an address already in use.

**EADDRNOTAVAIL**

A nonexistent interface was requested or the requested source address was not lo‐ cal.

**EAGAIN** Operation on a nonblocking socket would block.

**EALREADY**

A connection operation on a nonblocking socket is already in progress.

**ECONNABORTED**

A connection was closed during an accept(2).

**EHOSTUNREACH**

No valid routing table entry matches the destination address. This error can be caused by an ICMP message from a remote router or for the local routing table.

**EINVAL** Invalid argument passed. For send operations this can be caused by sending to a blackhole route.

**EISCONN**

connect(2) was called on an already connected socket.

**EMSGSIZE**

Datagram is bigger than an MTU on the path and it cannot be fragmented.

**ENOBUFS, ENOMEM**

Not enough free memory. This often means that the memory allocation is limited by the socket buffer limits, not by the system memory, but this is not 100% consis‐ tent.

**ENOENT** SIOCGSTAMP was called on a socket where no packet arrived.

**ENOPKG** A kernel subsystem was not configured.

**ENOPROTOOPT and EOPNOTSUPP**

Invalid socket option passed.

ENOTCONN

    The operation is defined only on a connected socket, but  the  socket  wasn't  con?

    nected.

EPERM  User  doesn't  have  permission to set high priority, change configuration, or send

    signals to the requested process or group.

EPIPE  The connection was unexpectedly closed or shut down by the other end.

ESOCKTNOSUPPORT

    The socket is not configured or an unknown socket type was requested.

Other errors may be generated by the overlaying protocols; see tcp(7), raw(7), udp(7), and

socket(7).

# NOTES

IP_FREEBIND, IP_MSFILTER, IP_MTU, IP_MTU_DISCOVER, IP_RECVORIGDSTADDR, IP_PASSSEC, IP_PKT?

INFO, IP_RECVERR, IP_ROUTER_ALERT, and IP_TRANSPARENT are Linux-specific.

Be very careful with the SO_BROADCAST option - it is not privileged in Linux.  It is  easy

to  overload  the  network  with careless broadcasts.  For new application protocols it is

better to use a multicast group instead of broadcasting.  Broadcasting is discouraged.

Some other BSD sockets implementations provide IP_RCVDSTADDR and IP_RECVIF socket  options

to  get  the  destination  address and the interface of received datagrams.  Linux has the

more general IP_PKTINFO for the same task.

Some BSD sockets implementations also provide an IP_RECVTTL option, but an ancillary  mes?

sage  with type IP_RECVTTL is passed with the incoming packet.  This is different from the

IP_TTL option used in Linux.

Using the SOL_IP socket options level isn't portable; BSD-based stacks use the  IPPROTO_IP

level.

INADDR_ANY (0.0.0.0) and INADDR_BROADCAST (255.255.255.255) are byte-order-neutral.

 This means htonl(3) has no effect on them.

## Compatibility

For compatibility with Linux 2.0, the obsolete socket(AF_INET, SOCK_PACKET, protocol) syn?

tax is still supported to open a packet(7) socket.  This is deprecated and should  be  re?

placed  by  socket(AF_PACKET, SOCK_RAW, protocol) instead.  The main difference is the new

sockaddr_ll address structure for generic link layer information instead of the old  sock?

addr_pkt.

# BUGS

There are too many inconsistent error values.

The error used to diagnose exhaustion of the ephemeral port range differs across the vari?

ous system calls (connect(2), bind(2), listen(2), sendto(2))  that  can  assign  ephemeral

ports.

The ioctls to configure IP-specific interface options and ARP tables are not described.

Receiving  the  original  destination  address with MSG_ERRQUEUE in msg_name by recvmsg(2)

does not work in some 2.2 kernels.

SEE ALSO

recvmsg(2), sendmsg(2), byteorder(3),  capabilities(7),  icmp(7),  ipv6(7),  netdevice(7),

netlink(7), raw(7), socket(7), tcp(7), udp(7), ip(8)

The kernel source file Documentation/networking/ip-sysctl.txt.

RFC 791  for  the  original  IP  specification.   RFC 1122 for the IPv4 host requirements.

RFC 1812 for the IPv4 router requirements.

COLOPHON

This page is part of release 5.10 of the Linux man-pages project.  A  description  of  the

project,  information  about  reporting  bugs, and the latest version of this page, can be

found at https://www.kernel.org/doc/man-pages/.

Linux                              2020-11-01                              IP(7)