

NAME

ip-route – routing table management

SYNOPSIS

ip [*ip-OPTIONS*] **route** { *COMMAND* | **help** }

ip route { **show** | **flush** } *SELECTOR*

ip route save *SELECTOR*

ip route restore

ip route get *ROUTE_GET_FLAGS* *ADDRESS* [**fr om** *ADDRESS* **iif** *STRING*] [**oif** *STRING*] [**mark** *MARK*] [**tos** *TOS*] [**vrf** *NAME*] [**ipproto** *PROTOCOL*] [**sport** *NUMBER*] [**dport** *NUMBER*]

ip route { **add** | **del** | **change** | **append** | **replace** } *ROUTE*

SELECTOR := [**root** *PREFIX*] [**match** *PREFIX*] [**exact** *PREFIX*] [**table** *TABLE_ID*] [**vrf** *NAME*] [**proto** *RTPROTO*] [**type** *TYPE*] [**scope** *SCOPE*]

ROUTE := *NODE_SPEC* [*INFO_SPEC*]

NODE_SPEC := [*TYPE*] *PREFIX* [**tos** *TOS*] [**table** *TABLE_ID*] [**proto** *RTPROTO*] [**scope** *SCOPE*] [**metric** *METRIC*] [**ttl-propagate** { **enabled** | **disabled** }]

INFO_SPEC := { *NH* | **nhid** *ID* } *OPTIONS* *FLAGS* [**nexthop** *NH*] ...

NH := [**encap** *ENCAP*] [**via** [*FAMILY*] *ADDRESS*] [**dev** *STRING*] [**weight** *NUMBER*] *NHFLAGS*

FAMILY := [**inet** | **inet6** | **mpls** | **bridge** | **link**]

OPTIONS := *FLAGS* [**mtu** *NUMBER*] [**advms** *NUMBER*] [**as** [**to**] *ADDRESS*] **rtt** *TIME*] [**rttvar** *TIME*] [**reordering** *NUMBER*] [**window** *NUMBER*] [**cwnd** *NUMBER*] [**ssthresh** *NUMBER*] [**realms** *REALM*] [**rto_min** *TIME*] [**initcwnd** *NUMBER*] [**initrwnd** *NUMBER*] [**features** *FEATURES*] [**quickack** *BOOL*] [**congtl** *NAME*] [**pref** *PREF*] [**expires** *TIME*] [**fastopen_no_cookie** *BOOL*]

TYPE := [**unicast** | **local** | **broadcast** | **multicast** | **throw** | **unreachable** | **prohibit** | **blackhole** | **nat**]

TABLE_ID := [**local** | **main** | **default** | **all** | *NUMBER*]

SCOPE := [**host** | **link** | **global** | *NUMBER*]

NHFLAGS := [**onlink** | **pervasive**]

RTPROTO := [**kernel** | **boot** | **static** | *NUMBER*]

FEATURES := [**ecn** |]

PREF := [**low** | **medium** | **high**]

ENCAP := [*ENCAP_MPLS* | *ENCAP_IP* | *ENCAP_BPF* | *ENCAP_SEG6* | *ENCAP_SEG6LOCAL* | *ENCAP_IOAM6*]

```

ENCAP_MPLS := mpls [ LABEL ] [ ttl TTL ]

ENCAP_IP := ip id TUNNEL_ID dst REMOTE_IP [ src SRC ] [ tos TOS ] [ ttl TTL ]

ENCAP_BPF := bpf [ in PROG ] [ out PROG ] [ xmit PROG ] [ headroom SIZE ]

ENCAP_SEG6 := seg6 mode [ encap | inline | l2encap ] segs SEGMENTS [ hmac KEYID ]

ENCAP_SEG6LOCAL := seg6local action SEG6_ACTION [ SEG6_ACTION_PARAM ] [ count ]

ENCAP_IOAM6 := ioam6 trace prealloc type IOAM6_TRACE_TYPE ns IOAM6_NAMESPACE size
               IOAM6_TRACE_SIZE

ROUTE_GET_FLAGS := [ fibmatch ]

```

DESCRIPTION

ip route is used to manipulate entries in the kernel routing tables.

Route types:

unicast - the route entry describes real paths to the destinations covered by the route prefix.

unreachable - these destinations are unreachable. Packets are discarded and the ICMP message *host unreachable* is generated. The local senders get an *EHOSTUNREACH* error.

blackhole - these destinations are unreachable. Packets are discarded silently. The local senders get an *EINVAL* error.

prohibit - these destinations are unreachable. Packets are discarded and the ICMP message *communication administratively prohibited* is generated. The local senders get an *EACCES* error.

local - the destinations are assigned to this host. The packets are looped back and delivered locally.

broadcast - the destinations are broadcast addresses. The packets are sent as link broadcasts.

throw - a special control route used together with policy rules. If such a route is selected, lookup in this table is terminated pretending that no route was found. Without policy routing it is equivalent to the absence of the route in the routing table. The packets are dropped and the ICMP message *net unreachable* is generated. The local senders get an *ENETUNREACH* error.

nat - a special NAT route. Destinations covered by the prefix are considered to be

dummy (or external) addresses which require translation to real (or internal) ones before forwarding. The addresses to translate to are selected with the attribute **via**. **Warning:** Route NAT is no longer supported in Linux 2.6.

anycast - *not implemented* the destinations are *anycast* addresses assigned to this host. They are mainly equivalent to **local** with one difference: such addresses are invalid when used as the source address of any packet.

multicast - a special type used for multicast routing. It is not present in normal routing tables.

Route tables: Linux-2.x can pack routes into several routing tables identified by a number in the range from 1 to $2^{32}-1$ or by name from the file `/etc/iproute2/rt_tables`. By default all normal routes are inserted into the **main** table (ID 254) and the kernel only uses this table when calculating routes. Values (0, 253, 254, and 255) are reserved for built-in use.

Actually, one other table always exists, which is invisible but even more important. It is the **local** table (ID 255). This table consists of routes for local and broadcast addresses. The kernel maintains this table automatically and the administrator usually need not modify it or even look at it.

The multiple routing tables enter the game when *policy routing* is used.

`ip route add`
add new route

`ip route change`
change route

`ip route replace`
change or add new one

to TYPE PREFIX (default)

the destination prefix of the route. If *TYPE* is omitted, **ip** assumes type **unicast**. Other values of *TYPE* are listed above. *PREFIX* is an IP or IPv6 address optionally followed by a slash and the prefix length. If the length of the prefix is missing, **ip** assumes a full-length host route. There is also a special *PREFIX default* - which is equivalent to IP **0/0** or to IPv6 **::/0**.

tos TOS

dsfield TOS

the Type Of Service (TOS) key. This key has no associated mask and the longest match is understood as: First, compare the TOS of the route and of the packet. If they are not equal, then the packet may still match a route with a zero TOS. *TOS* is either an 8 bit hexadecimal number or an identifier from `/etc/iproute2/rt_dsfield`.

metric NUMBER

preference NUMBER

the preference value of the route. *NUMBER* is an arbitrary 32bit number, where routes with lower values are preferred.

table *TABLEID*

the table to add this route to. *TABLEID* may be a number or a string from the file */etc/iproute2/rt_tables*. If this parameter is omitted, **ip** assumes the **main** table, with the exception of **local**, **broadcast** and **nat** routes, which are put into the **local** table by default.

vrf *NAME*

the vrf name to add this route to. Implicitly means the table associated with the VRF.

dev *NAME*

the output device name.

via [*FAMILY*] *ADDRESS*

the address of the nexthop router, in the address family *FAMILY*. Actually, the sense of this field depends on the route type. For normal **unicast** routes it is either the true next hop router or, if it is a direct route installed in BSD compatibility mode, it can be a local address of the interface. For NAT routes it is the first address of the block of translated IP destinations.

src *ADDRESS*

the source address to prefer when sending to the destinations covered by the route prefix.

realm *REALMID*

the realm to which this route is assigned. *REALMID* may be a number or a string from the file */etc/iproute2/rt_realms*.

mtu *MTU***mtu lock** *MTU*

the MTU along the path to the destination. If the modifier **lock** is not used, the MTU may be updated by the kernel due to Path MTU Discovery. If the modifier **lock** is used, no path MTU discovery will be tried, all packets will be sent without the DF bit in IPv4 case or fragmented to MTU for IPv6.

window *NUMBER*

the maximal window for TCP to advertise to these destinations, measured in bytes. It limits maximal data bursts that our TCP peers are allowed to send to us.

rtt *TIME*

the initial RTT ('Round Trip Time') estimate. If no suffix is specified the units are raw values passed directly to the routing code to maintain compatibility with previous releases. Otherwise if a suffix of s, sec or secs is used to specify seconds and ms, msec or msecs to specify milliseconds.

rttvar *TIME* (Linux 2.3.15+ only)

the initial RTT variance estimate. Values are specified as with **rtt** above.

rto_min *TIME* (Linux 2.6.23+ only)

the minimum TCP Retransmission TimeOut to use when communicating with this destination. Values are specified as with **rtt** above.

ssthresh *NUMBER* (Linux 2.3.15+ only)

an estimate for the initial slow start threshold.

cwnd *NUMBER* (Linux 2.3.15+ only)

the clamp for congestion window. It is ignored if the **lock** flag is not used.

initcwnd *NUMBER* (Linux 2.5.70+ only)

the initial congestion window size for connections to this destination. Actual window size is this value multiplied by the MSS (“Maximal Segment Size”) for same connection. The default is zero, meaning to use the values specified in RFC2414.

initrwnd *NUMBER* (Linux 2.6.33+ only)

the initial receive window size for connections to this destination. Actual window size is this value multiplied by the MSS of the connection. The default value is zero, meaning to use Slow Start value.

features *FEATURES* (Linux 3.18+ only)

Enable or disable per-route features. Only available feature at this time is **ecn** to enable explicit congestion notification when initiating connections to the given destination network. When responding to a connection request from the given network, ecn will also be used even if the **net.ipv4.tcp_ecn** sysctl is set to 0.

quickack *BOOL* (Linux 3.11+ only)

Enable or disable quick ack for connections to this destination.

fastopen_no_cookie *BOOL* (Linux 4.15+ only)

Enable TCP Fastopen without a cookie for connections to this destination.

congtctl *NAME* (Linux 3.20+ only)**congtctl lock** *NAME* (Linux 3.20+ only)

Sets a specific TCP congestion control algorithm only for a given destination. If not specified, Linux keeps the current global default TCP congestion control algorithm, or the one set from the application. If the modifier **lock** is not used, an application may nevertheless overwrite the suggested congestion control algorithm for that destination. If the modifier **lock** is used, then an application is not allowed to overwrite the specified congestion control algorithm for that destination, thus it will be enforced/guaranteed to use the proposed algorithm.

advms *NUMBER* (Linux 2.3.15+ only)

the MSS (‘Maximal Segment Size’) to advertise to these destinations when establishing TCP connections. If it is not given, Linux uses a default value calculated from the first hop device MTU. (If the path to these destination is asymmetric, this guess may be wrong.)

reordering *NUMBER* (Linux 2.3.15+ only)

Maximal reordering on the path to this destination. If it is not given, Linux uses the value selected with **sysctl** variable **net/ipv4/tcp_reordering**.

nexthop *NEXTHOP*

the nexthop of a multipath route. *NEXTHOP* is a complex value with its own syntax similar to the top level argument lists:

via [*FAMILY*] *ADDRESS* - is the nexthop router.

dev *NAME* - is the output device.

weight *NUMBER* - is a weight for this element of a multipath route reflecting its relative bandwidth or quality.

The internal buffer used in `iproute2` limits the maximum number of nexthops that may be specified in one go. If only *ADDRESS* is given, the current buffer size allows for 144 IPv6 nexthops and 253 IPv4 ones. For IPv4, this effectively limits the number of nexthops possible per route. With IPv6, further nexthops may be appended to the same route via **ip route append** command.

scope *SCOPE_VAL*

the scope of the destinations covered by the route prefix. *SCOPE_VAL* may be a number or a string from the file `/etc/iproute2/rt_scopes`. If this parameter is omitted, **ip** assumes scope **global** for all gatewayed **unicast** routes, scope **link** for direct **unicast** and **broadcast** routes and scope **host** for **local** routes.

protocol *RTPROTO*

the routing protocol identifier of this route. *RTPROTO* may be a number or a string from the file `/etc/iproute2/rt_protos`. If the routing protocol ID is not given, **ip** assumes **protocol boot** (i.e. it assumes the route was added by someone who doesn't understand what they are doing). Several protocol values have a fixed interpretation. Namely:

redirect - the route was installed due to an ICMP redirect.

kernel - the route was installed by the kernel during autoconfiguration.

boot - the route was installed during the bootup sequence. If a routing daemon starts, it will purge all of them.

static - the route was installed by the administrator to override dynamic routing. Routing daemon will respect them and, probably, even advertise them to its peers.

ra - the route was installed by Router Discovery protocol.

The rest of the values are not reserved and the administrator is free to assign (or not to assign) protocol tags.

onlink pretend that the nexthop is directly attached to this link, even if it does not match any interface prefix.

pref *PREF*

the IPv6 route preference. *PREF* is a string specifying the route preference as defined in RFC4191 for Router Discovery messages. Namely:

low - the route has a lowest priority

medium - the route has a default priority

high - the route has a highest priority

nhid *ID*

use nexthop object with given id as nexthop specification.

encap *ENCAPTYPE ENCAPHDR*

attach tunnel encapsulation attributes to this route.

ENCAPTYPE is a string specifying the supported encapsulation type. Namely:

mpls - encapsulation type MPLS

ip - IP encapsulation (Geneve, GRE, VXLAN, ...)

bpf - Execution of BPF program

seg6 - encapsulation type IPv6 Segment Routing

seg6local - local SRv6 segment processing

ioam6 - encapsulation type IPv6 IOAM

ENCAPHDR is a set of encapsulation attributes specific to the *ENCAPTYPE*.

mpls

MPLSLABEL - mpls label stack with labels separated by /

ttl *TTL* - TTL to use for MPLS header or 0 to inherit from IP header

ip

id *TUNNEL_ID* **dst** *REMOTE_IP* [**src** *SRC*] [**tos** *TOS*] [**ttl** *TTL*] [**key**] [**csum**] [**seq**]

bpf

in *PROG* - BPF program to execute for incoming packets

out *PROG* - BPF program to execute for outgoing packets

xmit *PROG* - BPF program to execute for transmitted packets

headroom *SIZE* - Size of header BPF program will attach (xmit)

seg6

mode inline - Directly insert Segment Routing Header after IPv6 header

mode encap - Encapsulate packet in an outer IPv6 header with SRH

mode l2encap - Encapsulate ingress L2 frame within an outer IPv6 header and SRH

SEGMENTS - List of comma-separated IPv6 addresses

KEYID - Numerical value in decimal representation. See **ip-sr(8)**.

seg6local

SEG6_ACTION [*SEG6_ACTION_PARAM*] [**count**] - Operation to perform on matching packets. The optional **count** attribute is used to collect statistics on the processing of actions. Three counters are implemented: 1) packets correctly processed; 2) bytes correctly processed; 3) packets that cause a processing error (i.e., missing SID List, wrong SID List, etc). To retrieve the counters related to an action use the **-s** flag in the **show** command. The following actions are currently supported (**Linux 4.14+ only**).

End - Regular SRv6 processing as intermediate segment endpoint. This action only accepts packets with a non-zero Segments Left value. Other matching packets are dropped.

End.X nh6 NEXTHOP - Regular SRv6 processing as intermediate segment endpoint. Additionally, forward processed packets to given next-hop. This action only accepts packets with a non-zero Segments Left value. Other matching packets are dropped.

End.DX6 nh6 NEXTHOP - Decapsulate inner IPv6 packet and forward it to the specified next-hop. If the argument is set to ::, then the next-hop is selected according to the local

selection rules. This action only accepts packets with either a zero Segments Left value or no SRH at all, and an inner IPv6 packet. Other matching packets are dropped.

End.DT6 { table | vrf table } TABLEID - Decapsulate the inner IPv6 packet and forward it according to the specified lookup table. *TABLEID* is either a number or a string from the file */etc/iproute2/rt_tables*. If *vrf table* is used, the argument must be a VRF device associated with the table id. Moreover, the VRF table associated with the table id must be configured with the VRF strict mode turned on (*net.vrf.strict_mode=1*). This action only accepts packets with either a zero Segments Left value or no SRH at all, and an inner IPv6 packet. Other matching packets are dropped.

End.DT4 vrf table TABLEID - Decapsulate the inner IPv4 packet and forward it according to the specified lookup table. *TABLEID* is either a number or a string from the file */etc/iproute2/rt_tables*. The argument must be a VRF device associated with the table id. Moreover, the VRF table associated with the table id must be configured with the VRF strict mode turned on (*net.vrf.strict_mode=1*). This action only accepts packets with either a zero Segments Left value or no SRH at all, and an inner IPv4 packet. Other matching packets are dropped.

End.DT46 vrf table TABLEID - Decapsulate the inner IPv4 or IPv6 packet and forward it according to the specified lookup table. *TABLEID* is either a number or a string from the file */etc/iproute2/rt_tables*. The argument must be a VRF device associated with the table id. Moreover, the VRF table associated with the table id must be configured with the VRF strict mode turned on (*net.vrf.strict_mode=1*). This action only accepts packets with either a zero Segments Left value or no SRH at all, and an inner IPv4 or IPv6 packet. Other matching packets are dropped.

End.B6 srh segs SEGMENTS [hmac KEYID] - Insert the specified SRH immediately after the IPv6 header, update the DA with the first segment of the newly inserted SRH, then forward the resulting packet. The original SRH is not modified. This action only accepts packets with a non-zero Segments Left value. Other matching packets are dropped.

End.B6.Encaps srh segs SEGMENTS [hmac KEYID] - Regular SRv6 processing as intermediate segment endpoint. Additionally, encapsulate the matching packet within an outer IPv6 header followed by the specified SRH. The destination address of the outer IPv6 header is set to the first segment of the new SRH. The source address is set as described in *ip-sr(8)*.

ioam6

IOAM6_TRACE_TYPE - List of IOAM data required in the trace, represented by a bitfield (24 bits).

IOAM6_NAMESPACE - Numerical value to represent an IOAM namespace. See **ip-ioam**(8).

IOAM6_TRACE_SIZE - Size, in octets, of the pre-allocated trace data block.

expires *TIME* (**Linux 4.4+ only**)

the route will be deleted after the expires time. **Only** support IPv6 at present.

ttl-propagate { **enabled** | **disabled** }

Control whether TTL should be propagated from any encap into the un-encapsulated packet, overriding any global configuration. Only supported for MPLS at present.

ip route delete

delete route

ip route del has the same arguments as **ip route add**, but their semantics are a bit different.

Key values (**to**, **tos**, **preference** and **table**) select the route to delete. If optional attributes are present, **ip** verifies that they coincide with the attributes of the route to delete. If no route with the given key and attributes was found, **ip route del** fails.

ip route show

list routes

the command displays the contents of the routing tables or the route(s) selected by some criteria.

to *SELECTOR* (**default**)

only select routes from the given range of destinations. *SELECTOR* consists of an optional modifier (**root**, **match** or **exact**) and a prefix. **root** *PREFIX* selects routes with prefixes not shorter than *PREFIX*. F.e. **root 0/0** selects the entire routing table. **match** *PREFIX* selects routes with prefixes not longer than *PREFIX*. F.e. **match 10.0/16** selects *10.0/16*, *10/8* and *0/0*, but it does not select *10.1/16* and *10.0.0/24*. And **exact** *PREFIX* (or just *PREFIX*) selects routes with this exact prefix. If neither of these options are present, **ip** assumes **root 0/0** i.e. it lists the entire table.

tos *TOS***dsfield** *TOS*

only select routes with the given TOS.

table *TABLEID*

show the routes from this table(s). The default setting is to show table **main**. *TABLEID* may either be the ID of a real table or one of the special values:

all - list all of the tables.

cache - dump the routing cache.

vrf *NAME*

show the routes for the table associated with the vrf name

cloned

cached list cloned routes i.e. routes which were dynamically forked from other routes because some route attribute (f.e. MTU) was updated. Actually, it is equivalent to **table cache**.

from *SELECTOR*

the same syntax as for **to**, but it binds the source address range rather than destinations. Note that the **from** option only works with cloned routes.

protocol *RTPROTO*

only list routes of this protocol.

scope *SCOPE_VAL*

only list routes with this scope.

type *TYPE*

only list routes of this type.

dev *NAME*

only list routes going via this device.

via [*FAMILY*] *PREFIX*

only list routes going via the nexthop routers selected by *PREFIX*.

src *PREFIX*

only list routes with preferred source addresses selected by *PREFIX*.

realm *REALMID*

realms *FROMREALM/TOREALM*

only list routes with these realms.

ip route flush

flush routing tables

this command flushes routes selected by some criteria.

The arguments have the same syntax and semantics as the arguments of **ip route show**, but routing tables are not listed but purged. The only difference is the default action: **show** dumps all the IP main routing table but **flush** prints the helper page.

With the **-statistics** option, the command becomes verbose. It prints out the number of deleted routes and the number of rounds made to flush the routing table. If the option is given twice, **ip route flush** also dumps all the deleted routes in the format described in the previous subsection.

ip route get

get a single route

this command gets a single route to a destination and prints its contents exactly as the kernel sees it.

fibmatch

Return full fib lookup matched route. Default is to return the resolved dst entry

to ADDRESS (default)

the destination address.

from ADDRESS

the source address.

tos TOS

dsfield TOS

the Type Of Service.

iif NAME

the device from which this packet is expected to arrive.

oif NAME

force the output device on which this packet will be routed.

mark MARK

the firewall mark (**fwmark**)

vrf NAME

force the vrf device on which this packet will be routed.

ipproto PROTOCOL

ip protocol as seen by the route lookup

sport NUMBER

source port as seen by the route lookup

dport NUMBER

destination port as seen by the route lookup

connected

if no source address (option **from**) was given, relookup the route with the source set to the preferred address received from the first lookup. If policy routing is used, it may be a different route.

Note that this operation is not equivalent to **ip route show**. **show** shows existing routes. **get** resolves them and creates new clones if necessary. Essentially, **get** is equivalent to sending a packet along this path. If the **iif** argument is not given, the kernel creates a route to output packets towards the requested destination. This is equivalent to pinging the destination with a subsequent **ip route ls cache**, however, no packets are actually sent. With the **iif** argument, the kernel pretends

that a packet arrived from this interface and searches for a path to forward the packet.

ip route save

save routing table information to stdout

This command behaves like **ip route show** except that the output is raw data suitable for passing to **ip route restore**.

ip route restore

restore routing table information from stdin

This command expects to read a data stream as returned from **ip route save**. It will attempt to restore the routing table information exactly as it was at the time of the save, so any translation of information in the stream (such as device indexes) must be done first. Any existing routes are left unchanged. Any routes specified in the data stream that already exist in the table will be ignored.

NOTES

Starting with Linux kernel version 3.6, there is no routing cache for IPv4 anymore. Hence **ip route show cached** will never print any entries on systems with this or newer kernel versions.

EXAMPLES

ip ro

Show all route entries in the kernel.

ip route add default via 192.168.1.1 dev eth0

Adds a default route (for all addresses) via the local gateway 192.168.1.1 that can be reached on device eth0.

ip route add 10.1.1.0/30 encap mpls 200/300 via 10.1.1.1 dev eth0

Adds an ipv4 route with mpls encapsulation attributes attached to it.

ip -6 route add 2001:db8:1::/64 encap seg6 mode encap segs 2001:db8:42::1,2001:db8:ffff::2 dev eth0

Adds an IPv6 route with SRv6 encapsulation and two segments attached.

ip -6 route add 2001:db8:1::/64 encap seg6local action End.DT46 vrf table 100 dev vrf100

Adds an IPv6 route with SRv6 decapsulation and forward with lookup in VRF table.

ip -6 route add 2001:db8:1::/64 encap ioam6 trace prealloc type 0x800000 ns 1 size 12 dev eth0

Adds an IPv6 route with an IOAM Pre-allocated Trace encapsulation that only includes the hop limit and the node id, configured for the IOAM namespace 1 and a pre-allocated data block of 12 octets.

ip route add 10.1.1.0/30 nhid 10

Adds an ipv4 route using nexthop object with id 10.

SEE ALSO

ip(8)

AUTHOR

Original Manpage by Michail Litvak <mci@owl.openwall.com>