NAME

ip − show / manipulate routing, devices, policy routing and tunnels

SYNOPSIS

ip [*OPTIONS*] *OBJECT* { *COMMAND* | **help** }

OBJECT := { **link** | **addr** | **route** | **rule** | **neigh** | **tunnel** | **maddr** | **mroute** | **monitor** }

OPTIONS := { −**V**[*ersion*] | −**s**[*tatistics*] | −**r**[*esolve*] | −**f**[*amily*] { **inet** | **inet6** | **ipx** | **dnet** | **link** } | −**o**[*neline*] }

ip link set *DEVICE* { **up** | **down** | **arp** { **on** | **off** } |
promisc { **on** | **off** } |
allmulti { **on** | **off** } |
dynamic { **on** | **off** } |
multicast { **on** | **off** } |
txqueuelen *PACKETS* |
name *NEWNAME* |
address *LLADDR* | **broadcast** *LLADDR* |
mtu *MTU* }

ip link show [*DEVICE*]

ip addr { **add** | **del** } *IFADDR* **dev** *STRING*

ip addr { **show** | **flush** } [**dev** *STRING*] [**scope** *SCOPE-ID*] [**to** *PREFIX*] [*FLAG-LIST*] [**label** *PATTERN*]

IFADDR := *PREFIX* | *ADDR* **peer** *PREFIX* [**broadcast** *ADDR*] [**anycast** *ADDR*] [**label** *STRING*] [**scope** *SCOPE-ID*]

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

FLAG-LIST := [*FLAG-LIST*] *FLAG*

FLAG := [**permanent** | **dynamic** | **secondary** | **primary** | **tentative** | **deprecated**]

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

ip route get *ADDRESS* [**from** *ADDRESS* **iif** *STRING*] [**oif** *STRING*] [**tos** *TOS*]

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

SELECTOR := [**root** *PREFIX*] [**match** *PREFIX*] [**exact** *PREFIX*] [**table** *TABLE_ID*] [**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*]

INFO_SPEC := *NH OPTIONS FLAGS* [**nexthop** *NH*] ...

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

OPTIONS := *FLAGS* [**mtu** *NUMBER*] [**advmss** *NUMBER*] [**rtt** *NUMBER*] [**rttvar** *NUMBER*] [**window** *NUMBER*] [**cwnd** *NUMBER*] [**ssthresh** *REALM*] [**realms** *REALM*]

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

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

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

FLAGS := [**equalize**]

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

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

ip rule [**list** | **add** | **del**] *SELECTOR ACTION*

SELECTOR := [**from** *PREFIX*] [**to** *PREFIX*] [**tos** *TOS*] [**fwmark** *FWMARK*] [**dev** *STRING*] [**pref** *NUMBER*]

ACTION := [**table** *TABLE_ID*] [**nat** *ADDRESS*] [**prohibit** | **reject** | **unreachable**] [**realms** [*SRCREALM*/]*DSTREALM*]

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

ip neigh { **add** | **del** | **change** | **replace** } { *ADDR* [**lladdr** *LLADDR*] [**nud** { **permanent** | **noarp** | **stale** | **reachable** }] | **proxy** *ADDR* } [**dev** *DEV*]

ip neigh { **show** | **flush** } [**to** *PREFIX*] [**dev** *DEV*] [**nud** *STATE*]

ip tunnel { **add** | **change** | **del** | **show** } [*NAME*]
[**mode** { **ipip** | **gre** | **sit** }]
[**remote** *ADDR*] [**local** *ADDR*]
[[**i**|**o**]**seq**] [[**i**|**o**]**key** *KEY*] [[**i**|**o**]**csum**]]
[**ttl** *TTL*] [**tos** *TOS*] [[**no**]**pmtudisc**]
[**dev** *PHYS_DEV*]

ADDR := { *IP_ADDRESS* | **any** }

TOS := { *NUMBER* | **inherit** }

TTL := { *1..255* | **inherit** }

KEY := { *DOTTED_QUAD* | *NUMBER* }

ip maddr [**add** | **del**] *MULTIADDR* **dev** *STRING*

ip maddr show [**dev** *STRING*]

ip mroute show [*PREFIX*] [**from** *PREFIX*] [**iif** *DEVICE*]

ip monitor [**all** | *LISTofOBJECTS*]

OPTIONS

−V, **-Version**
print the version of the **ip** utility and exit.

−s, **−stats**, **−statistics**
output more information. If the option appears twice or more, the amount of information increases. As a rule, the information is statistics or some time values.

−f, **−family**
followed by protocol family identifier: **inet**, **inet6** or **link** ,enforce the protocol family to use. If the option is not present, the protocol family is guessed from other arguments. If the rest of the command line does not give enough information to guess the family, **ip** falls back to the default one, usually **inet** or **any**. **link** is a special family identifier meaning that no networking protocol is involved.

−4 shortcut for **-family inet**.

−6 shortcut for −**family inet6**.

−0 shortcut for −**family link**.

−o, **−oneline**
output each record on a single line, replacing line feeds with the **'´** character. This is convenient when you want to count records with **wc**(1)
or to **grep**(1) the output.

−r, **−resolve**
use the system's name resolver to print DNS names instead of host addresses.

IP - COMMAND SYNTAX

OBJECT

link - network device.

address
- protocol (IP or IPv6) address on a device.

neighbour
- ARP or NDISC cache entry.

route - routing table entry.

rule - rule in routing policy database.

maddress
- multicast address.

mroute

- multicast routing cache entry.

tunnel

- tunnel over IP.

The names of all objects may be written in full or abbreviated form, f.e. **address** is abbreviated as **addr** or just **a.**

COMMAND

Specifies the action to perform on the object. The set of possible actions depends on the object type. As a rule, it is possible to **add**, **delete** and **show** (or **list**) objects, but some objects do not allow all of these operations or have some additional commands. The **help** command is available for all objects. It prints out a list of available commands and argument syntax conventions.

If no command is given, some default command is assumed. Usually it is **list** or, if the objects of this class cannot be listed, **help**.

ip link - network device configuration

link is a network device and the corresponding commands display and change the state of devices.

ip link set - change device attributes

dev *NAME* **(default)**

NAME specifies network device to operate on.

up and **down**

change the state of the device to **UP** or **DOWN**.

arp on or **arp off**

change the **NOARP** flag on the device.

multicast on or **multicast off**

change the **MULTICAST** flag on the device.

dynamic on or **dynamic off**

change the **DYNAMIC** flag on the device.

name *NAME*

change the name of the device. This operation is not recommended if the device is running or has some addresses already configured.

txqueuelen *NUMBER*

txqlen *NUMBER*

change the transmit queue length of the device.

mtu *NUMBER*

change the *MTU* of the device.

address *LLADDRESS*
change the station address of the interface.

broadcast *LLADDRESS*

brd *LLADDRESS*

peer *LLADDRESS*
change the link layer broadcast address or the peer address when the interface is *POINTOPOINT*.

Warning: If multiple parameter changes are requested, **ip** aborts immediately after any of the changes have failed. This is the only case when **ip** can move the system to an unpredictable state. The solution is to avoid changing several parameters with one **ip link set** call.

ip link show - display device attributes

dev *NAME* **(default)**
NAME specifies the network device to show. If this argument is omitted all devices are listed.

up only display running interfaces.

ip address - protocol address management.

The **address** is a protocol (IP or IPv6) address attached to a network device. Each device must have at least one address to use the corresponding protocol. It is possible to have several different addresses attached to one device. These addresses are not discriminated, so that the term **alias** is not quite appropriate for them and we do not use it in this document.

The **ip addr** command displays addresses and their properties, adds new addresses and deletes old ones.

ip address add - add new protocol address.

dev *NAME*
the name of the device to add the address to.

local *ADDRESS* **(default)**
the address of the interface. The format of the address depends on the protocol. It is a dotted quad for IP and a sequence of hexadecimal halfwords separated by colons for IPv6. The *ADDRESS* may be followed by a slash and a decimal number which encodes the network prefix length.

peer *ADDRESS*
the address of the remote endpoint for pointopoint interfaces. Again, the *ADDRESS* may be followed by a slash and a decimal number, encoding the network prefix length. If a peer address is specified, the local address cannot have a prefix length. The network prefix is associated with the peer rather than with the local address.

broadcast *ADDRESS*
the broadcast address on the interface.

It is possible to use the special symbols **'+'** and **'-'** instead of the broadcast address. In this case, the broadcast address is derived by setting/resetting the host bits of the interface prefix.

label *NAME*

Each address may be tagged with a label string. In order to preserve compatibility with Linux-2.0 net aliases, this string must coincide with the name of the device or must be prefixed with the device name followed by colon.

scope *SCOPE_VALUE*

the scope of the area where this address is valid. The available scopes are listed in file **/etc/iproute2/rt_scopes**. Predefined scope values are:

global - the address is globally valid.

site - (IPv6 only) the address is site local, i.e. it is valid inside this site.

link - the address is link local, i.e. it is valid only on this device.

host - the address is valid only inside this host.

ip address delete - delete protocol address

Arguments: coincide with the arguments of **ip addr add.** The device name is a required argument. The rest are optional. If no arguments are given, the first address is deleted.

ip address show - look at protocol addresses

dev *NAME* **(default)**

name of device.

scope *SCOPE_VAL*

only list addresses with this scope.

to *PREFIX*

only list addresses matching this prefix.

label *PATTERN*

only list addresses with labels matching the *PATTERN*. *PATTERN* is a usual shell style pattern.

dynamic and **permanent**

(IPv6 only) only list addresses installed due to stateless address configuration or only list permanent (not dynamic) addresses.

tentative

(IPv6 only) only list addresses which did not pass duplicate address detection.

deprecated

(IPv6 only) only list deprecated addresses.

primary and **secondary**

only list primary (or secondary) addresses.

ip address flush - flush protocol addresses

This command flushes the protocol addresses selected by some criteria.

This command has the same arguments as **show.** The difference is that it does not run when no arguments are given.

Warning: This command (and other **flush** commands described below) is pretty dangerous. If you make a mistake, it will not forgive it, but will cruelly purge all the addresses.

With the **-statistics** option, the command becomes verbose. It prints out the number of deleted addresses and the number of rounds made to flush the address list. If this option is given twice, **ip addr flush** also dumps all the deleted addresses in the format described in the previous subsection.

ip neighbour - neighbour/arp tables management.

neighbour objects establish bindings between protocol addresses and link layer addresses for hosts sharing the same link. Neighbour entries are organized into tables. The IPv4 neighbour table is known by another name - the ARP table.

The corresponding commands display neighbour bindings and their properties, add new neighbour entries and delete old ones.

ip neighbour add - add a new neighbour entry
ip neighbour change - change an existing entry
ip neighbour replace - add a new entry or change an existing one

These commands create new neighbour records or update existing ones.

to *ADDRESS* **(default)**
the protocol address of the neighbour. It is either an IPv4 or IPv6 address.

dev *NAME*
the interface to which this neighbour is attached.

lladdr *LLADDRESS*
the link layer address of the neighbour. *LLADDRESS* can also be **null**.

nud *NUD_STATE*
the state of the neighbour entry. **nud** is an abbreviation for 'Neigh bour Unreachability Detection'. The state can take one of the following values:

permanent - the neighbour entry is valid forever and can be only be removed administratively.

noarp - the neighbour entry is valid. No attempts to validate this entry will be made but it can be removed when its lifetime expires.

reachable - the neighbour entry is valid until the reachability timeout expires.

stale - the neighbour entry is valid but suspicious. This option to **ip neigh** does not change the neighbour state if it was valid and the address is not changed by this command.

ip neighbour delete - delete a neighbour entry

This command invalidates a neighbour entry.

The arguments are the same as with **ip neigh add**, except that **lladdr** and **nud** are ignored.

Warning: Attempts to delete or manually change a **noarp** entry created by the kernel may result in unpredictable behaviour. Particularly, the kernel may try to resolve this address even on a **NOARP** interface or if the address is multicast or broadcast.

ip neighbour show - list neighbour entries

This commands displays neighbour tables.

to *ADDRESS* **(default)**
the prefix selecting the neighbours to list.

dev *NAME*
only list the neighbours attached to this device.

unused
only list neighbours which are not currently in use.

nud *NUD_STATE*
only list neighbour entries in this state. *NUD_STATE* takes values listed below or the special value **all** which means all states. This option may occur more than once. If this option is absent, **ip** lists all entries except for **none** and **noarp**.

ip neighbour flush - flush neighbour entries

This command flushes neighbour tables, selecting entries to flush by some criteria.

This command has the same arguments as **show.** The differences are that it does not run when no arguments are given, and that the default neighbour states to be flushed do not include **permanent** and **noarp**.

With the **-statistics** option, the command becomes verbose. It prints out the number of deleted neighbours and the number of rounds made to flush the neighbour table. If the option is given twice, **ip neigh flush** also dumps all the deleted neighbours.

ip route - routing table management

Manipulate route entries in the kernel routing tables keep information about paths to other networked nodes.

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**.

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 255 or by name from the file **/etc/iproute2/rt_tables main** table (ID 254) and the kernel only uses this table when calculating routes.

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.

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.

dev *NAME*

the output device name.

via *ADDRESS*

the address of the nexthop router. 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 *NUMBER*

the initial RTT ('Round Trip Time') estimate.

rttvar *NUMBER* **(2.3.15+ only)**

the initial RTT variance estimate.

ssthresh *NUMBER* **(2.3.15+ only)**

an estimate for the initial slow start threshold.

cwnd *NUMBER* **(2.3.15+ only)**

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

advmss *NUMBER* **(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* **(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 *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.

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.

equalize

allow packet by packet randomization on multipath routes. Without this modifier, the route will be frozen to one selected nexthop, so that load splitting will only occur on per-flow base. **equalize** only works if the kernel is patched.

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.

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 *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.

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.

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 rule - routing policy database management

Rules in the routing policy database control the route selection algorithm.

Classic routing algorithms used in the Internet make routing decisions based only on the destination address of packets (and in theory, but not in practice, on the TOS field).

In some circumstances we want to route packets differently depending not only on destination addresses, but also on other packet fields: source address, IP protocol, transport protocol ports or even packet payload. This task is called 'policy routing'.

To solve this task, the conventional destination based routing table, ordered according to the longest match rule, is replaced with a 'routing policy database' (or RPDB), which selects routes by executing some set of rules.

Each policy routing rule consists of a **selector** and an **action predicate.** The RPDB is scanned in the order of increasing priority. The selector of each rule is applied to {source address, destination address, incoming interface, tos, fwmark} and, if the selector matches the packet, the action is performed. The action predicate may return with success. In this case, it will either give a route or failure indication and the RPDB lookup is terminated. Otherwise, the RPDB program continues on the next rule.

Semantically, natural action is to select the nexthop and the output device.

At startup time the kernel configures the default RPDB consisting of three rules:

1. Priority: 0, Selector: match anything, Action: lookup routing table **local** (ID 255). The **local** table is a special routing table containing high priority control routes for local and broadcast addresses.

   Rule 0 is special. It cannot be deleted or overridden.

2. Priority: 32766, Selector: match anything, Action: lookup routing table **main** (ID 254). The **main** table is the normal routing table containing all non-policy routes. This rule may be deleted and/or overridden with other ones by the administrator.

3. Priority: 32767, Selector: match anything, Action: lookup routing table **default** (ID 253). The **default** table is empty. It is reserved for some post-processing if no previous default rules selected the packet. This rule may also be deleted.

Each RPDB entry has additional attributes. F.e. each rule has a pointer to some routing table. NAT and masquerading rules have an attribute to select new IP address to translate/masquerade. Besides that, rules have some optional attributes, which routes have, namely **realms**. These values do not override those contained in the routing tables. They are only used if the route did not select any attributes.

The RPDB may contain rules of the following types:

unicast - the rule prescribes to return the route found in the routing table referenced by the rule.

blackhole - the rule prescribes to silently drop the packet.

unreachable - the rule prescribes to generate a 'Network is unreachable' error.

prohibit - the rule prescribes to generate 'Communication is administratively prohibited' error.

nat - the rule prescribes to translate the source address of the IP packet into some other value.

ip rule add - insert a new rule
ip rule delete - delete a rule

type *TYPE* **(default)**

the type of this rule. The list of valid types was given in the previous subsection.

from *PREFIX*
select the source prefix to match.

to *PREFIX*
select the destination prefix to match.

iif *NAME*
select the incoming device to match. If the interface is loopback, the rule only matches packets originating from this host. This means that you may create separate routing tables for forwarded and local packets and, hence, completely segregate them.

tos *TOS*

dsfield *TOS*
select the TOS value to match.

fwmark *MARK*
select the **fwmark** value to match.

priority *PREFERENCE*
the priority of this rule. Each rule should have an explicitly set *unique* priority value.

table *TABLEID*
the routing table identifier to lookup if the rule selector matches.

realms *FROM/TO*
Realms to select if the rule matched and the routing table lookup succeeded. Realm *TO* is only used if the route did not select any realm.

nat *ADDRESS*
The base of the IP address block to translate (for source addresses). The *ADDRESS* may be either the start of the block of NAT addresses (selected by NAT routes) or a local host address (or even zero). In the last case the router does not translate the packets, but masquerades them to this address.

Warning: Changes to the RPDB made with these commands do not become active immediately. It is assumed that after a script finishes a batch of updates, it flushes the routing cache with **ip route flush cache**.

ip rule show - list rules

This command has no arguments.

ip maddress - multicast addresses management

maddress objects are multicast addresses.

ip maddress show - list multicast addresses

dev *NAME* **(default)**
the device name.

ip maddress add - add a multicast address
ip maddress delete - delete a multicast address

these commands attach/detach a static link layer multicast address to listen on the interface. Note that it is impossible to join protocol multicast groups statically. This command only manages link layer addresses.

address *LLADDRESS* **(default)**
the link layer multicast address.

dev *NAME*
the device to join/leave this multicast address.

ip mroute - multicast routing cache management

mroute objects are multicast routing cache entries created by a user level mrouting daemon (f.e. **pimd** or **mrouted**).

Due to the limitations of the current interface to the multicast routing engine, it is impossible to change **mroute** objects administratively, so we may only display them. This limitation will be removed in the future.

ip mroute show - list mroute cache entries

to *PREFIX* **(default)**
the prefix selecting the destination multicast addresses to list.

iif *NAME*
the interface on which multicast packets are received.

from *PREFIX*
the prefix selecting the IP source addresses of the multicast route.

ip tunnel - tunnel configuration

tunnel objects are tunnels, encapsulating packets in IPv4 packets and then sending them over the IP infrastructure.

ip tunnel add - add a new tunnel
ip tunnel change - change an existing tunnel
ip tunnel delete - destroy a tunnel

name *NAME* **(default)**
select the tunnel device name.

mode *MODE*
set the tunnel mode. Three modes are currently available: **ipip**, **sit** and **gre**.

remote *ADDRESS*
set the remote endpoint of the tunnel.

local *ADDRESS*
set the fixed local address for tunneled packets. It must be an address on another interface of this host.

ttl *N* set a fixed TTL *N* on tunneled packets. *N* is a number in the range 1--255. 0 is a special value meaning that packets inherit the TTL value. The default value is: **inherit**.

tos *T*

dsfield *T*
set a fixed TOS *T* on tunneled packets. The default value is: **inherit**.

dev *NAME*
bind the tunnel to the device *NAME* so that tunneled packets will only be routed via this device and will not be able to escape to another device when the route to endpoint changes.

nopmtudisc
disable Path MTU Discovery on this tunnel. It is enabled by default. Note that a fixed ttl is incompatible with this option: tunnelling with a fixed ttl always makes pmtu discovery.

key *K*

ikey *K*

okey *K*
(**only GRE tunnels**) use keyed GRE with key *K*. *K* is either a number or an IP address-like dotted quad. The **key** parameter sets the key to use in both directions. The **ikey** and **okey** parameters set different keys for input and output.

csum, **icsum**, **ocsum**
(**only GRE tunnels**) generate/require checksums for tunneled packets. The **ocsum** flag calculates checksums for outgoing packets. The **icsum** flag requires that all input packets have the correct checksum. The **csum** flag is equivalent to the combination **icsum ocsum**.

seq, **iseq**, **oseq**
(**only GRE tunnels**) serialize packets. The **oseq** flag enables sequencing of outgoing packets. The **iseq** flag requires that all input packets are serialized. The **seq** flag is equivalent to the combination **iseq oseq**. **It isn't work. Don't use it.**

ip tunnel show - list tunnels

This command has no arguments.

ip monitor and rtmon - state monitoring

The **ip** utility can monitor the state of devices, addresses and routes continuously. This option has a slightly different format. Namely, the **monitor** command is the first in the command line and then the object list follows:

ip monitor [**all** | *LISTofOBJECTS*]

OBJECT-LIST is the list of object types that we want to monitor. It may contain **link**, **address** and **route**. If no **file** argument is given, **ip** opens RTNETLINK, listens on it and dumps state changes in the format described in previous sections.

If a file name is given, it does not listen on RTNETLINK, but opens the file containing

RTNETLINK messages saved in binary format and dumps them. Such a history file can be generated with the **rtmon** utility. This utility has a command line syntax similar to **ip monitor**. Ideally, **rtmon** should be started before the first network configuration command is issued. F.e. if you insert:

```
rtmon file /var/log/rtmon.log
```

in a startup script, you will be able to view the full history later.

Certainly, it is possible to start **rtmon** at any time. It prepends the history with the state snapshot dumped at the moment of starting.

HISTORY

ip was written by Alexey N. Kuznetsov and added in Linux 2.2.

SEE ALSO

tc(8)
IP Command reference **ip-cref.ps**
IP tunnels **ip-cref.ps**

AUTHOR

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