Rocky Enterprise Linux 9.2 Manual Pages on command 'nftables.8'

$ man nftables.8

NFT(8) NFT(8)

NAME

nft - Administration tool of the nftables framework for packet filtering and classification

SYNOPSIS

nft [-nNscaeSupyjt] [-I directory] [-f filename | -i | cmd ...]

nft -h

nft -v

DESCRIPTION

nft is the command line tool used to set up, maintain and inspect packet filtering and classification rules in the Linux kernel, in the nftables framework. The Linux kernel subsystem is known as nf_tables, and ?nf? stands for Netfilter.

OPTIONS

The command accepts several different options which are documented here in groups for better understanding of their meaning. You can get information about options by running nft --help.

General options:

-h, --help

Show help message and all options.

-v, --version

Show version.

-V

Show long version information, including compile-time configuration.

Ruleset input handling options that specify to how to load rulesets:

-f, --file filename

Read input from filename. If filename is -, read from stdin.

-D, --define name=value

Define a variable. You can only combine this option with -f.

-i, --interactive

Read input from an interactive readline CLI. You can use quit to exit, or use the EOF marker, normally this is CTRL-D.

-I, --includepath directory

Add the directory directory to the list of directories to be searched for included files. This option may be specified multiple times.

-c, --check

Check commands validity without actually applying the changes.

-o, --optimize

Optimize your ruleset. You can combine this option with -c to inspect the proposed optimizations.

Ruleset list output formatting that modify the output of the list ruleset command:

-a, --handle

Show object handles in output.

-s, --stateless

Omit stateful information of rules and stateful objects.

-t, --terse

Omit contents of sets from output.

-S, --service

Translate ports to service names as defined by /etc/services.

-N, --reversedns

Translate IP address to names via reverse DNS lookup. This may slow down your listing since it generates network traffic.

-u, --guid

Translate numeric UID/GID to names as defined by /etc/passwd and /etc/group.

-n, --numeric

Print fully numerical output.

-y, --numeric-priority

Display base chain priority numerically.

-p, --numeric-protocol

Display layer 4 protocol numerically.

-T, --numeric-time

Show time, day and hour values in numeric format.

Command output formatting:

-e, --echo

When inserting items into the ruleset using add, insert or replace commands, print notifications just like nft monitor.

-j, --json

Format output in JSON. See libnftables-json(5) for a schema description.

-d, --debug level

Enable debugging output. The debug level can be any of scanner, parser, eval, netlink, mnl, proto-ctx, segtree, all. You can combine more than one by separating by the , symbol, for example -d eval,mnl.

INPUT FILE FORMATS

LEXICAL CONVENTIONS

Input is parsed line-wise. When the last character of a line, just before the newline character, is a non-quoted backslash (\), the next line is treated as a continuation.

Multiple commands on the same line can be separated using a semicolon (;).

A hash sign (#) begins a comment. All following characters on the same line are ignored.

Identifiers begin with an alphabetic character (a-z,A-Z), followed by zero or more alphanumeric characters (a-z,A-Z,0-9) and the characters slash (/), backslash (\), underscore (_) and dot (.). Identifiers using different characters or clashing with a keyword need to be enclosed in double quotes (").

INCLUDE FILES

```
include filename
```

Other files can be included by using the include statement. The directories to be searched for include files can be specified using the -I/--includepath option. You can override this behaviour either by prepending ?./? to your path to force inclusion of files located in the current working directory (i.e. relative path) or / for file location expressed as an absolute path.

If -I/--includepath is not specified, then nft relies on the default directory that is specified at compile time. You can retrieve this default directory via the -h/--help option.

Include statements support the usual shell wildcard symbols (,?,[]). Having no matches for an include statement is not an error, if wildcard symbols are used in the include statement. This allows having potentially empty include directories for statements like include "/etc/firewall/rules/". The wildcard matches are loaded in alphabetical order. Files beginning with dot (.) are not matched by include statements.

SYMBOLIC VARIABLES

```
define variable = expr
undefine variable
redefine variable = expr
$variable
```

Symbolic variables can be defined using the define statement. Variable references are expressions and can be used to initialize other variables. The scope of a definition is the current block and all blocks contained within. Symbolic variables can be undefined using the undefine statement, and modified using the redefine statement.

Using symbolic variables.

```
define int_if1 = eth0
define int_if2 = eth1
define int_ifs = { $int_if1, $int_if2 }
redefine int_if2 = wlan0
undefine int_if2
filter input iif $int_ifs accept
```

ADDRESS FAMILIES

Address families determine the type of packets which are processed. For each address family, the kernel contains so called hooks at specific stages of the packet processing paths, which invoke nftables if rules for these hooks exist.

ip IPv4 address family.

ip6 IPv6 address family.

inet Internet (IPv4/IPv6) address family.

arp ARP address family, handling IPv4 ARP packets.

bridge Bridge address family, handling packets which traverse a bridge device.

netdev Netdev address family, handling packets on ingress and egress.

All nftables objects exist in address family specific namespaces, therefore all identifiers include an address family. If an identifier is specified without an address family, the ip family is used by default.

IPV4/IPV6/INET ADDRESS FAMILIES

The IPv4/IPv6/Inet address families handle IPv4, IPv6 or both types of packets. They contain five hooks at different packet processing stages in the network stack.

Table 1. IPv4/IPv6/Inet address family hooks

Hook	Description
prerouting	All packets entering the system are processed by the prerouting hook. It is invoked before the routing process and is used for early filtering or changing packet attributes that affect routing.
input	Packets delivered to the local system are processed by the input hook.
forward	Packets forwarded to a different host are processed by the forward hook.

```
?????????????????????????????????????????????????
?           ?                                   ?
?output     ? Packets sent by local processes   ?
?           ? are processed by the output       ?
?           ? hook.                             ?
?????????????????????????????????????????????????
?           ?                                   ?
?postrouting ? All packets leaving the system    ?
?           ? are processed by the postrouting  ?
?           ? hook.                             ?
?????????????????????????????????????????????????
?           ?                                   ?
?ingress    ? All packets entering the system   ?
?           ? are processed by this hook. It    ?
?           ? is invoked before layer 3         ?
?           ? protocol handlers, hence before   ?
?           ? the prerouting hook, and it can   ?
?           ? be used for filtering and         ?
?           ? policing. Ingress is only         ?
?           ? available for Inet family (since  ?
?           ? Linux kernel 5.10).               ?
?????????????????????????????????????????????????
```

ARP ADDRESS FAMILY

The ARP address family handles ARP packets received and sent by the system. It is commonly used to mangle ARP packets for clustering.

Table 2. ARP address family hooks

```
????????????????????????????????????????????
?Hook   ? Description                      ?
????????????????????????????????????????????
?       ?                                  ?
?input  ? Packets delivered to the local   ?
?       ? system are processed by the      ?
?       ? input hook.                      ?
```

```
???????????????????????????????????????????
?      ?                                  ?
?output ? Packets send by the local system ?
?      ? are processed by the output      ?
?      ? hook.                            ?
???????????????????????????????????????????
```

BRIDGE ADDRESS FAMILY

The bridge address family handles Ethernet packets traversing bridge devices.

The list of supported hooks is identical to IPv4/IPv6/Inet address families above.

NETDEV ADDRESS FAMILY

The Netdev address family handles packets from the device ingress and egress path. This family allows you to filter packets of any ethertype such as ARP, VLAN 802.1q, VLAN 802.1ad (Q-in-Q) as well as IPv4 and IPv6 packets.

Table 3. Netdev address family hooks

```
???????????????????????????????????????????
?Hook    ? Description                    ?
???????????????????????????????????????????
?        ?                                ?
?ingress ? All packets entering the system ?
?        ? are processed by this hook. It  ?
?        ? is invoked after the network    ?
?        ? taps (ie. tcpdump), right after ?
?        ? tc ingress and before layer 3   ?
?        ? protocol handlers, it can be    ?
?        ? used for early filtering and    ?
?        ? policing.                       ?
???????????????????????????????????????????
?        ?                                ?
?egress  ? All packets leaving the system  ?
?        ? are processed by this hook. It  ?
?        ? is invoked after layer 3        ?
?        ? protocol handlers and before tc ?
?        ? egress. It can be used for late ?
```

? ? filtering and policing. ?

??

Tunneled packets (such as vxlan) are processed by netdev family hooks both in decapsulated and encapsulated (tunneled) form. So a packet can be filtered on the overlay network as well as on the underlying network.

Note that the order of netfilter and tc is mirrored on ingress versus egress. This ensures symmetry for NAT and other packet mangling.

Ingress packets which are redirected out some other interface are only processed by netfilter on egress if they have passed through netfilter ingress processing before. Thus, ingress packets which are redirected by tc are not subjected to netfilter. But they are if they are redirected by netfilter on ingress. Conceptually, tc and netfilter can be thought of as layers, with netfilter layered above tc: If the packet hasn?t been passed up from the tc layer to the netfilter layer, it?s not subjected to netfilter on egress.

RULESET

```
{list | flush} ruleset [family]
```

The ruleset keyword is used to identify the whole set of tables, chains, etc. currently in place in kernel. The following ruleset commands exist:

list Print the ruleset in human-readable format.

flush Clear the whole ruleset. Note that, unlike iptables, this will remove all tables and whatever they contain, effectively leading to an empty ruleset - no packet filtering will happen anymore, so the kernel accepts any valid packet it receives.

It is possible to limit list and flush to a specific address family only. For a list of valid family names, see the section called ?ADDRESS FAMILIES? above.

By design, list ruleset command output may be used as input to nft -f. Effectively, this is the nft-equivalent of iptables-save and iptables-restore.

TABLES

```
{add | create} table [family] table [ {comment comment ;} { flags 'flags ; }]
```

```
{delete | list | flush} table [family] table
list tables [family]
delete table [family] handle handle
```

Tables are containers for chains, sets and stateful objects. They are identified by their address family and their name. The address family must be one of ip, ip6, inet, arp, bridge, netdev. The inet address family is a dummy family which is used to create hybrid IPv4/IPv6 tables. The meta expression nfproto keyword can be used to test which family (ipv4 or ipv6) context the packet is being processed in. When no address family is specified, ip is used by default. The only difference between add and create is that the former will not return an error if the specified table already exists while create will return an error.

Table 4. Table flags

Flag	Description
dormant	table is not evaluated any more (base chains are unregistered).

Add, change, delete a table.

```
# start nft in interactive mode
nft --interactive
# create a new table.
create table inet mytable
# add a new base chain: get input packets
add chain inet mytable myin { type filter hook input priority filter; }
# add a single counter to the chain
add rule inet mytable myin counter
# disable the table temporarily -- rules are not evaluated anymore
add table inet mytable { flags dormant; }
# make table active again:
add table inet mytable
```

add   Add a new table for the given

family with the given name.

delete Delete the specified table.

list List all chains and rules of the specified table.

flush Flush all chains and rules of the specified table.

CHAINS

{add | create} chain [family] table chain [{ type type hook hook [device device] priority priority ; [policy policy ;] [comment comment ;] }]

{delete | list | flush} chain ['family] table chain

list chains [family]

delete chain [family] table handle handle

rename chain [family] table chain newname

Chains are containers for rules. They exist in two kinds, base chains and regular chains.

A base chain is an entry point for packets from the networking stack, a regular chain may

be used as jump target and is used for better rule organization.

add Add a new chain in the specified table. When a hook and priority value are specified, the chain is created as a base chain and hooked up to the networking stack.

create Similar to the add command, but returns an error if the chain already exists.

delete Delete the specified chain. The chain must not contain any rules or be used as jump target.

rename Rename the specified chain.

list List all rules of the specified chain.

flush Flush all rules of the specified chain.

For base chains, type, hook and priority parameters are mandatory.

Table 5. Supported chain types

```
????????????????????????????????????????????????????????????????????
?Type   ? Families      ? Hooks               ? Description        ?
????????????????????????????????????????????????????????????????????
?       ?               ?                     ?                    ?
?filter ? all           ? all                 ? Standard chain type ?
?       ?               ?                     ? to use in doubt.   ?
????????????????????????????????????????????????????????????????????
?       ?               ?                     ?                    ?
?nat    ? ip, ip6, inet ? prerouting, input,  ? Chains of this type ?
?       ?               ? output, postrouting ? perform Native     ?
?       ?               ?                     ? Address Translation ?
?       ?               ?                     ? based on conntrack  ?
?       ?               ?                     ? entries. Only the  ?
?       ?               ?                     ? first packet of a  ?
?       ?               ?                     ? connection actually ?
?       ?               ?                     ? traverses this     ?
?       ?               ?                     ? chain - its rules  ?
?       ?               ?                     ? usually define     ?
?       ?               ?                     ? details of the     ?
?       ?               ?                     ? created conntrack  ?
?       ?               ?                     ? entry (NAT         ?
?       ?               ?                     ? statements for     ?
?       ?               ?                     ? instance).         ?
????????????????????????????????????????????????????????????????????
?       ?               ?                     ?                    ?
?route  ? ip, ip6       ? output              ? If a packet has    ?
?       ?               ?                     ? traversed a chain  ?
?       ?               ?                     ? of this type and is ?
?       ?               ?                     ? about to be        ?
?       ?               ?                     ? accepted, a new    ?
?       ?               ?                     ? route lookup is    ?
?      ?             ?                  ? performed if       ?
?      ?             ?                  ? relevant parts of   ?
?      ?             ?                  ? the IP header have  ?
?      ?             ?                  ? changed. This      ?
?      ?             ?                  ? allows to e.g.     ?
?      ?             ?                  ? implement policy    ?
?      ?             ?                  ? routing selectors   ?
?      ?             ?                  ? in nftables.       ?
??????????????????????????????????????????????????????????????????????
```

Apart from the special cases illustrated above (e.g. nat type not supporting forward hook or route type only supporting output hook), there are three further quirks worth noticing:

? The netdev family supports merely two combinations, namely filter type with ingress hook and filter type with egress hook. Base chains in this family also require the device parameter to be present since they exist per interface only.

? The arp family supports only the input and output hooks, both in chains of type filter.

? The inet family also supports the ingress hook (since Linux kernel 5.10), to filter IPv4 and IPv6 packet at the same location as the netdev ingress hook. This inet hook allows you to share sets and maps between the usual prerouting, input, forward, output, postrouting and this ingress hook.

The priority parameter accepts a signed integer value or a standard priority name which specifies the order in which chains with the same hook value are traversed. The ordering is ascending, i.e. lower priority values have precedence over higher ones.

Standard priority values can be replaced with easily memorizable names. Not all names make sense in every family with every hook (see the compatibility matrices below) but their numerical value can still be used for prioritizing chains.

These names and values are defined and made available based on what priorities are used by xtables when registering their default chains.

Most of the families use the same values, but bridge uses different ones from the others.

See the following tables that describe the values and compatibility.

Table 6. Standard priority names, family and hook compatibility matrix

```
??????????????????????????????????????????????????????
?Name     ? Value ? Families          ? Hooks       ?
```

raw	-300	ip, ip6, inet	all
mangle	-150	ip, ip6, inet	all
dstnat	-100	ip, ip6, inet	prerouting
filter	0	ip, ip6, inet, arp, netdev	all
security	50	ip, ip6, inet	all
srcnat	100	ip, ip6, inet	postrouting

Table 7. Standard priority names and hook compatibility for the bridge family

Name	Value	Hooks
dstnat	-300	prerouting
filter	-200	all
out	100	output

```
???????????????????????????????
?       ?       ?             ?
?srcnat ? 300   ? postrouting ?
???????????????????????????????
```

Basic arithmetic expressions (addition and subtraction) can also be achieved with these standard names to ease relative prioritizing, e.g. mangle - 5 stands for -155. Values will also be printed like this until the value is not further than 10 from the standard value.

Base chains also allow to set the chain?s policy, i.e. what happens to packets not explicitly accepted or refused in contained rules. Supported policy values are accept (which is the default) or drop.

RULES

{add **|** insert} rule [family] table chain [handle handle **|** index index] statement ... [comment comment]

replace rule [family] table chain handle handle statement ... [comment comment]

delete rule [family] table chain handle handle

Rules are added to chains in the given table. If the family is not specified, the ip family is used. Rules are constructed from two kinds of components according to a set of grammatical rules: expressions and statements.

The add and insert commands support an optional location specifier, which is either a handle or the index (starting at zero) of an existing rule. Internally, rule locations are always identified by handle and the translation from index happens in userspace. This has two potential implications in case a concurrent ruleset change happens after the translation was done: The effective rule index might change if a rule was inserted or deleted before the referred one. If the referred rule was deleted, the command is rejected by the kernel just as if an invalid handle was given.

A comment is a single word or a double-quoted (") multi-word string which can be used to make notes regarding the actual rule. Note: If you use bash for adding rules, you have to escape the quotation marks, e.g. \"enable ssh for servers\".

add   Add a new rule described by the
list of statements. The rule is
appended to the given chain
unless a location is specified,
in which case the rule is
inserted after the specified

rule.

insert Same as add except the rule is inserted at the beginning of the chain or before the specified rule.

replace Similar to add, but the rule replaces the specified rule.

delete Delete the specified rule.

add a rule to ip table output chain.

```
nft add rule filter output ip daddr 192.168.0.0/24 accept # 'ip filter' is assumed
# same command, slightly more verbose
nft add rule ip filter output ip daddr 192.168.0.0/24 accept
```

delete rule from inet table.

```
# nft -a list ruleset
table inet filter {
        chain input {
                type filter hook input priority filter; policy accept;
                ct state established,related accept # handle 4
                ip saddr 10.1.1.1 tcp dport ssh accept # handle 5
          ...
# delete the rule with handle 5
nft delete rule inet filter input handle 5
```

SETS

nftables offers two kinds of set concepts. Anonymous sets are sets that have no specific name. The set members are enclosed in curly braces, with commas to separate elements when creating the rule the set is used in. Once that rule is removed, the set is removed as well. They cannot be updated, i.e. once an anonymous set is declared it cannot be changed anymore except by removing/altering the rule that uses the anonymous set.

Using anonymous sets to accept particular subnets and ports.

```
nft add rule filter input ip saddr { 10.0.0.0/8, 192.168.0.0/16 } tcp dport { 22, 443 } accept
```

Named sets are sets that need to be defined first before they can be referenced in rules. Unlike anonymous sets, elements can be added to or removed from a named set at any time. Sets are referenced from rules using an @ prefixed to the sets name.

Using named sets to accept addresses and ports.

```
nft add rule filter input ip saddr @allowed_hosts tcp dport @allowed_ports accept
```

The sets allowed_hosts and allowed_ports need to be created first. The next section describes nft set syntax in more detail.

```
    add set [family] table set { type type | typeof expression ; [flags flags ;] [timeout timeout ;] [gc-interval gc-interval ;]
[elements = { element[, ...] } ;] [size size ;] [comment comment ;] [policy 'policy ;] [auto-merge ;] }
{delete | list | flush} set [family] table set
list sets [family]
delete set [family] table handle handle
{add | delete} element [family] table set { element[, ...] }
```

Sets are element containers of a user-defined data type, they are uniquely identified by a user-defined name and attached to tables. Their behaviour can be tuned with the flags that can be specified at set creation time.

add   Add a new set in the specified table. See the Set specification table below for more information about how to specify properties of a set.

delete   Delete the specified set.

list   Display the elements in the specified set.

flush   Remove all elements from the specified set.

Table 8. Set specifications

Keyword	Description	Type
type	data type of set elements	string: ipv4_addr, ipv6_addr, ether_addr, inet_proto, inet_service, mark

```
?           ?                         ?                          ?
?typeof     ? data type of set element ? expression to derive the ?
?           ?                         ? data type from           ?
?????????????????????????????????????????????????????????????????
?           ?                         ?                          ?
?flags      ? set flags               ? string: constant,        ?
?           ?                         ? dynamic, interval,       ?
?           ?                         ? timeout                  ?
?????????????????????????????????????????????????????????????????
?           ?                         ?                          ?
?timeout    ? time an element stays in ? string, decimal followed ?
?           ? the set, mandatory if   ? by unit. Units are: d,   ?
?           ? set is added to from the ? h, m, s                  ?
?           ? packet path (ruleset)   ?                          ?
?????????????????????????????????????????????????????????????????
?           ?                         ?                          ?
?gc-interval ? garbage collection      ? string, decimal followed ?
?           ? interval, only available ? by unit. Units are: d,   ?
?           ? when timeout or flag    ? h, m, s                  ?
?           ? timeout are active      ?                          ?
?????????????????????????????????????????????????????????????????
?           ?                         ?                          ?
?elements   ? elements contained by   ? set data type            ?
?           ? the set                 ?                          ?
?????????????????????????????????????????????????????????????????
?           ?                         ?                          ?
?size       ? maximum number of       ? unsigned integer (64     ?
?           ? elements in the set,    ? bit)                     ?
?           ? mandatory if set is     ?                          ?
?           ? added to from the packet ?                          ?
?           ? path (ruleset)          ?                          ?
?????????????????????????????????????????????????????????????????
?           ?                         ?                          ?
```

```
?policy      ? set policy              ? string: performance      ?
?            ?                         ? [default], memory        ?
????????????????????????????????????????????????????????????????????????
?            ?                         ?                          ?
?auto-merge  ? automatic merge of      ?                          ?
?            ? adjacent/overlapping set ?                         ?
?            ? elements (only for      ?                          ?
?            ? interval sets)          ?                          ?
????????????????????????????????????????????????????????????????????????
```

MAPS

add map [family] table map { type type **|** typeof expression [flags flags ;] [elements = { element[, ...] } ;] [size size ;] [comment comment ;] [policy 'policy ;] }

{delete **|** list **|** flush} map [family] table map

list maps [family]

Maps store data based on some specific key used as input. They are uniquely identified by a user-defined name and attached to tables.

add   Add a new map in the specified
               table.

delete   Delete the specified map.

list   Display the elements in the
               specified map.

flush   Remove all elements from the
               specified map.

add element   Comma-separated list of elements
               to add into the specified map.

delete element Comma-separated list of element
               keys to delete from the
               specified map.

Table 9. Map specifications

```
?????????????????????????????????????????????????????????????????????
?Keyword  ? Description            ? Type                    ?
?????????????????????????????????????????????????????????????????????
?         ?                        ?                         ?
```

```
?type     ? data type of map          ? string: ipv4_addr,       ?
?         ? elements                  ? ipv6_addr, ether_addr,   ?
?         ?                           ? inet_proto,              ?
?         ?                           ? inet_service, mark,      ?
?         ?                           ? counter, quota. Counter  ?
?         ?                           ? and quota can?t be used  ?
?         ?                           ? as keys                  ?
??????????????????????????????????????????????????????????????????
?         ?                           ?                          ?
?typeof   ? data type of set element ? expression to derive the ?
?         ?                           ? data type from           ?
??????????????????????????????????????????????????????????????????
?         ?                           ?                          ?
?flags    ? map flags                 ? string: constant,        ?
?         ?                           ? interval                 ?
??????????????????????????????????????????????????????????????????
?         ?                           ?                          ?
?elements ? elements contained by     ? map data type            ?
?         ? the map                   ?                          ?
??????????????????????????????????????????????????????????????????
?         ?                           ?                          ?
?size     ? maximum number of         ? unsigned integer (64     ?
?         ? elements in the map       ? bit)                     ?
??????????????????????????????????????????????????????????????????
?         ?                           ?                          ?
?policy   ? map policy                ? string: performance      ?
?         ?                           ? [default], memory        ?
??????????????????????????????????????????????????????????????????
```

ELEMENTS

```
{add | create | delete | get } element [family] table set { ELEMENT[, ...] }

ELEMENT := key_expression OPTIONS [: value_expression]

OPTIONS := [timeout TIMESPEC] [expires TIMESPEC] [comment string]

TIMESPEC := [numd][numh][numm][num[s]]
```

Element-related commands allow to change contents of named sets and maps. key_expression is typically a value matching the set type. value_expression is not allowed in sets but mandatory when adding to maps, where it matches the data part in its type definition. When deleting from maps, it may be specified but is optional as key_expression uniquely identifies the element.

create command is similar to add with the exception that none of the listed elements may already exist.

get command is useful to check if an element is contained in a set which may be non-trivial in very large and/or interval sets. In the latter case, the containing interval is returned instead of just the element itself.

Table 10. Element options

Option	Description
timeout	timeout value for sets/maps with flag timeout
expires	the time until given element expires, useful for ruleset replication only
comment	per element comment field

FLOWTABLES

```
{add | create} flowtable [family] table flowtable { hook hook priority priority ; devices = { device[, ...] } ; }
list flowtables [family]
{delete | list} flowtable [family] table flowtable
delete flowtable [family] table handle handle
```

Flowtables allow you to accelerate packet forwarding in software. Flowtables entries are represented through a tuple that is composed of the input interface, source and destination address, source and destination port; and layer 3/4 protocols. Each entry also caches the destination interface and the gateway address - to update the destination link-layer address - to forward packets. The ttl and hoplimit fields are also decremented. Hence, flowtables provides an alternative path that allow packets to bypass the classic forwarding path. Flowtables reside in the ingress hook that is located before the prerouting hook. You can select which flows you want to offload through the flow expression from the forward chain. Flowtables are identified by their address family and their name. The address family must be one of ip, ip6, or inet. The inet address family is a dummy family which is used to create hybrid IPv4/IPv6 tables. When no address family is specified, ip is used by default.

The priority can be a signed integer or filter which stands for 0. Addition and subtraction can be used to set relative priority, e.g. filter + 5 equals to 5.

add   Add a new flowtable for the given family with the given name.

delete   Delete the specified flowtable.

list   List all flowtables.

LISTING

list { secmarks | synproxys | flow tables | meters | hooks } [family]

list { secmarks | synproxys | flow tables | meters | hooks } table [family] table

list ct { timeout | expectation | helper | helpers } table [family] table

Inspect configured objects. list hooks shows the full hook pipeline, including those registered by kernel modules, such as nf_conntrack.

STATEFUL OBJECTS

{add | delete | list | reset} type [family] table object

delete type [family] table handle handle

list counters [family]

list quotas [family]

list limits [family]

Stateful objects are attached to tables and are identified by a unique name. They group stateful information from rules, to reference them in rules the keywords "type name" are used e.g. "counter name".

add   Add a new stateful object in the specified table.

delete   Delete the specified object.

list   Display stateful information the
         object holds.

reset   List-and-reset stateful object.

CT HELPER

```
add ct helper [family] table name { type type protocol protocol ; [l3proto family ;] }

delete ct helper [family] table name

list ct helpers
```

Ct helper is used to define connection tracking helpers that can then be used in combination with the ct helper set statement. type and protocol are mandatory, l3proto is derived from the table family by default, i.e. in the inet table the kernel will try to load both the ipv4 and ipv6 helper backends, if they are supported by the kernel.

Table 11. conntrack helper specifications

```
????????????????????????????????????????????????????????????????
?Keyword  ? Description         ? Type                ?
????????????????????????????????????????????????????????????????
?         ?                     ?                     ?
?type     ? name of helper type ? quoted string (e.g. ?
?         ?                     ? "ftp")              ?
????????????????????????????????????????????????????????????????
?         ?                     ?                     ?
?protocol ? layer 4 protocol of the ? string (e.g. ip)    ?
?         ? helper              ?                     ?
????????????????????????????????????????????????????????????????
?         ?                     ?                     ?
?l3proto  ? layer 3 protocol of the ? address family (e.g. ip) ?
?         ? helper              ?                     ?
????????????????????????????????????????????????????????????????
?         ?                     ?                     ?
?comment  ? per ct helper comment ? string              ?
?         ? field               ?                     ?
????????????????????????????????????????????????????????????????
```

defining and assigning ftp helper.

Unlike iptables, helper assignment needs to be performed after the conntrack lookup has completed, for example with the default 0 hook priority.

```
table inet myhelpers {
  ct helper ftp-standard {
     type "ftp" protocol tcp
  }
  chain prerouting {
      type filter hook prerouting priority filter;
      tcp dport 21 ct helper set "ftp-standard"
  }
}
```

CT TIMEOUT

```
add ct timeout  [family] table name { protocol protocol ; policy = { state: value [, ...] } ; [l3proto family ;] }
delete ct timeout [family] table name
list ct timeouts
```

Ct timeout is used to update connection tracking timeout values.Timeout policies are assigned with the ct timeout set statement. protocol and policy are mandatory, l3proto is derived from the table family by default.

Table 12. conntrack timeout specifications

```
?????????????????????????????????????????????????????????????????
?Keyword  ? Description           ? Type                ?
?????????????????????????????????????????????????????????????????
?         ?                       ?                     ?
?protocol ? layer 4 protocol of the ? string (e.g. ip)     ?
?         ? timeout object         ?                     ?
?????????????????????????????????????????????????????????????????
?         ?                       ?                     ?
?state    ? connection state name   ? string (e.g.         ?
?         ?                       ? "established")        ?
?????????????????????????????????????????????????????????????????
?         ?                       ?                     ?
?value    ? timeout value for      ? unsigned integer      ?
```

```
?         ? connection state        ?                         ?
???????????????????????????????????????????????????????????????
?         ?                         ?                         ?
?l3proto  ? layer 3 protocol of the ? address family (e.g. ip) ?
?         ? timeout object          ?                         ?
???????????????????????????????????????????????????????????????
?         ?                         ?                         ?
?comment  ? per ct timeout comment  ? string                  ?
?         ? field                   ?                         ?
???????????????????????????????????????????????????????????????
```

tcp connection state names that can have a specific timeout value are:

close, close_wait, established, fin_wait, last_ack, retrans, syn_recv, syn_sent, time_wait

and unack.

You can use sysctl -a |grep net.netfilter.nf_conntrack_tcp_timeout_ to view and change the

system-wide defaults. ct timeout allows for flow-specific settings, without changing the

global timeouts.

For example, tcp port 53 could have much lower settings than other traffic.

udp state names that can have a specific timeout value are replied and unreplied.

defining and assigning ct timeout policy.

```
    table ip filter {
            ct timeout customtimeout {
                    protocol tcp;
                    l3proto ip
                    policy = { established: 120, close: 20 }
            }
            chain output {
                    type filter hook output priority filter; policy accept;
                    ct timeout set "customtimeout"
            }
    }
```

testing the updated timeout policy.

```
    % conntrack -E
    It should display:
```

```
[UPDATE] tcp      6 120 ESTABLISHED src=172.16.19.128 dst=172.16.19.1

sport=22 dport=41360 [UNREPLIED] src=172.16.19.1 dst=172.16.19.128

sport=41360 dport=22
```

CT EXPECTATION

```
add ct expectation  [family] table name { protocol protocol ; dport dport ; timeout timeout ; size size ; [*l3proto family ;]
}

delete ct expectation  [family] table name

list ct expectations
```

Ct expectation is used to create connection expectations. Expectations are assigned with the ct expectation set statement. protocol, dport, timeout and size are mandatory, l3proto is derived from the table family by default.

Table 13. conntrack expectation specifications

```
???????????????????????????????????????????????????????????????
?Keyword  ? Description            ? Type                    ?
???????????????????????????????????????????????????????????????
?         ?                        ?                         ?
?protocol ? layer 4 protocol of the ? string (e.g. ip)        ?
?         ? expectation object     ?                         ?
???????????????????????????????????????????????????????????????
?         ?                        ?                         ?
?dport    ? destination port of    ? unsigned integer        ?
?         ? expected connection    ?                         ?
???????????????????????????????????????????????????????????????
?         ?                        ?                         ?
?timeout  ? timeout value for      ? unsigned integer        ?
?         ? expectation            ?                         ?
???????????????????????????????????????????????????????????????
?         ?                        ?                         ?
?size     ? size value for         ? unsigned integer        ?
?         ? expectation            ?                         ?
???????????????????????????????????????????????????????????????
?         ?                        ?                         ?
?l3proto  ? layer 3 protocol of the ? address family (e.g. ip) ?
```

```
?          ? expectation object      ?                          ?
????????????????????????????????????????????????????????????????
?          ?                         ?                          ?
?comment  ? per ct expectation      ? string                   ?
?          ? comment field           ?                          ?
????????????????????????????????????????????????????????????????
```

defining and assigning ct expectation policy.

```
table ip filter {
        ct expectation expect {
                protocol udp
                dport 9876
                timeout 2m
                size 8
                l3proto ip
        }
        chain input {
                type filter hook input priority filter; policy accept;
                ct expectation set "expect"
        }
}
```

COUNTER

```
add counter [family] table name [{ [ packets packets bytes bytes ; ] [ comment comment ; }]
delete counter [family] table name
list counters
```

Table 14. Counter specifications

```
????????????????????????????????????????????????????????????
?Keyword ? Description              ? Type                 ?
????????????????????????????????????????????????????????????
?        ?                          ?                      ?
?packets ? initial count of packets ? unsigned integer (64 ?
?        ?                          ? bit)                 ?
????????????????????????????????????????????????????????????
?        ?                          ?                      ?
```

```
?bytes   ? initial count of bytes   ? unsigned integer (64 ?
?        ?                          ? bit)                ?
????????????????????????????????????????????????????????????
?        ?                          ?                     ?
?comment ? per counter comment      ? string              ?
?        ? field                    ?                     ?
????????????????????????????????????????????????????????????
```

Using named counters.

```
nft add counter filter http
nft add rule filter input tcp dport 80 counter name \"http\"
```

Using named counters with maps.

```
nft add counter filter http
nft add counter filter https
nft add rule filter input counter name tcp dport map { 80 : \"http\", 443 : \"https\" }
```

QUOTA

```
add quota [family] table name { [over|until] bytes BYTE_UNIT [ used bytes BYTE_UNIT ] ; [ comment comment ; ] }
BYTE_UNIT := bytes | kbytes | mbytes
delete quota [family] table name
list quotas
```

Table 15. Quota specifications

```
???????????????????????????????????????????????????????????????
?Keyword ? Description              ? Type                    ?
???????????????????????????????????????????????????????????????
?        ?                          ?                         ?
?quota   ? quota limit, used as the ? Two arguments, unsigned ?
?        ? quota name               ? integer (64 bit) and    ?
?        ?                          ? string: bytes, kbytes,  ?
?        ?                          ? mbytes. "over" and      ?
?        ?                          ? "until" go before these ?
?        ?                          ? arguments               ?
???????????????????????????????????????????????????????????????
?        ?                          ?                         ?
?used    ? initial value of used    ? Two arguments, unsigned ?
```

```
?      ? quota              ? integer (64 bit) and   ?
?      ?                    ? string: bytes, kbytes,  ?
?      ?                    ? mbytes               ?
??????????????????????????????????????????????????????????????
?      ?                    ?                   ?
?comment ? per quota comment field  ? string              ?
??????????????????????????????????????????????????????????????
```

Using named quotas.

```
nft add quota filter user123 { over 20 mbytes }
nft add rule filter input ip saddr 192.168.10.123 quota name \"user123\"
```

Using named quotas with maps.

```
nft add quota filter user123 { over 20 mbytes }
nft add quota filter user124 { over 20 mbytes }
nft add rule filter input quota name ip saddr map { 192.168.10.123 : \"user123\", 192.168.10.124 : \"user124\" }
```

EXPRESSIONS

Expressions represent values, either constants like network addresses, port numbers, etc., or data gathered from the packet during ruleset evaluation. Expressions can be combined using binary, logical, relational and other types of expressions to form complex or relational (match) expressions. They are also used as arguments to certain types of operations, like NAT, packet marking etc.

Each expression has a data type, which determines the size, parsing and representation of symbolic values and type compatibility with other expressions.

DESCRIBE COMMAND

```
describe expression | data type
```

The describe command shows information about the type of an expression and its data type.

A data type may also be given, in which nft will display more information about the type.

The describe command.

```
$ nft describe tcp flags
payload expression, datatype tcp_flag (TCP flag) (basetype bitmask, integer), 8 bits
predefined symbolic constants:
fin                    0x01
syn                      0x02
rst                    0x04
```

```
psh                     0x08
ack                     0x10
urg                     0x20
ecn                     0x40
cwr                     0x80
```

DATA TYPES

Data types determine the size, parsing and representation of symbolic values and type compatibility of expressions. A number of global data types exist, in addition some expression types define further data types specific to the expression type. Most data types have a fixed size, some however may have a dynamic size, f.i. the string type. Some types also have predefined symbolic constants. Those can be listed using the nft describe command:

```
$ nft describe ct_state
datatype ct_state (conntrack state) (basetype bitmask, integer), 32 bits
pre-defined symbolic constants (in hexadecimal):
invalid                     0x00000001
new ...
```

Types may be derived from lower order types, f.i. the IPv4 address type is derived from the integer type, meaning an IPv4 address can also be specified as an integer value.

In certain contexts (set and map definitions), it is necessary to explicitly specify a data type. Each type has a name which is used for this.

INTEGER TYPE

```
???????????????????????????????????????????
?Name    ? Keyword ? Size     ? Base type ?
???????????????????????????????????????????
?        ?         ?          ?           ?
?Integer ? integer ? variable ? -         ?
???????????????????????????????????????????
```

The integer type is used for numeric values. It may be specified as a decimal, hexadecimal or octal number. The integer type does not have a fixed size, its size is determined by the expression for which it is used.

BITMASK TYPE

```
???????????????????????????????????????????
```

?Name   ? Keyword ? Size   ? Base type ?

???

?   ?   ?   ?   ?

?Bitmask ? bitmask ? variable ? integer   ?

???

The bitmask type (bitmask) is used for bitmasks.

STRING TYPE

??

?Name   ? Keyword ? Size   ? Base type ?

??

?   ?   ?   ?   ?

?String ? string ? variable ? -   ?

??

The string type is used for character strings. A string begins with an alphabetic character (a-zA-Z) followed by zero or more alphanumeric characters or the characters /, -, _ and .. In addition, anything enclosed in double quotes (") is recognized as a string.

String specification.

# Interface name

filter input iifname eth0

# Weird interface name

filter input iifname "(eth0)"

LINK LAYER ADDRESS TYPE

??

?Name   ? Keyword ? Size   ? Base type ?

??

?   ?   ?   ?   ?

?Link layer address ? lladdr ? variable ? integer   ?

??

The link layer address type is used for link layer addresses. Link layer addresses are specified as a variable amount of groups of two hexadecimal digits separated using colons (:).

Link layer address specification.

# Ethernet destination MAC address

```
filter input ether daddr 20:c9:d0:43:12:d9
```

IPV4 ADDRESS TYPE

Name	Keyword	Size	Base type
IPV4 address	ipv4_addr	32 bit	integer

The IPv4 address type is used for IPv4 addresses. Addresses are specified in either dotted decimal, dotted hexadecimal, dotted octal, decimal, hexadecimal, octal notation or as a host name. A host name will be resolved using the standard system resolver.

IPv4 address specification.

```
# dotted decimal notation
filter output ip daddr 127.0.0.1
# host name
filter output ip daddr localhost
```

IPV6 ADDRESS TYPE

Name	Keyword	Size	Base type
IPv6 address	ipv6_addr	128 bit	integer

The IPv6 address type is used for IPv6 addresses. Addresses are specified as a host name or as hexadecimal halfwords separated by colons. Addresses might be enclosed in square brackets ("[]") to differentiate them from port numbers.

IPv6 address specification.

```
# abbreviated loopback address
filter output ip6 daddr ::1
```

IPv6 address specification with bracket notation.

```
# without [] the port number (22) would be parsed as part of the
# ipv6 address
ip6 nat prerouting tcp dport 2222 dnat to [1ce::d0]:22
```

BOOLEAN TYPE

```
????????????????????????????????????????
?Name    ? Keyword ? Size  ? Base type ?
????????????????????????????????????????
?        ?         ?       ?           ?
?Boolean ? boolean ? 1 bit ? integer   ?
????????????????????????????????????????
```

The boolean type is a syntactical helper type in userspace. Its use is in the right-hand side of a (typically implicit) relational expression to change the expression on the left-hand side into a boolean check (usually for existence).

Table 16. The following keywords will automatically resolve into a boolean type with given value

```
??????????????????
?Keyword ? Value ?
??????????????????
?        ?       ?
?exists  ? 1     ?
??????????????????
?        ?       ?
?missing ? 0     ?
??????????????????
```

Table 17. expressions support a boolean comparison

```
???????????????????????????????????????????
?Expression ? Behaviour                   ?
???????????????????????????????????????????
?           ?                             ?
?fib        ? Check route existence.      ?
???????????????????????????????????????????
?           ?                             ?
?exthdr     ? Check IPv6 extension header ?
?           ? existence.                  ?
???????????????????????????????????????????
?           ?                             ?
```

```
?tcp option ? Check TCP option header     ?
?           ? existence.                  ?
???????????????????????????????????????????
```

Boolean specification.

```
# match if route exists
filter input fib daddr . iif oif exists
# match only non-fragmented packets in IPv6 traffic
filter input exthdr frag missing
# match if TCP timestamp option is present
filter input tcp option timestamp exists
```

ICMP TYPE TYPE

```
????????????????????????????????????????????
?Name      ? Keyword   ? Size  ? Base type ?
????????????????????????????????????????????
?          ?           ?       ?           ?
?ICMP Type ? icmp_type ? 8 bit ? integer   ?
????????????????????????????????????????????
```

The ICMP Type type is used to conveniently specify the ICMP header?s type field.

Table 18. Keywords may be used when specifying the ICMP type

```
??????????????????????????????????
?Keyword                 ? Value ?
??????????????????????????????????
?                        ?       ?
?echo-reply              ? 0     ?
??????????????????????????????????
?                        ?       ?
?destination-unreachable ? 3     ?
??????????????????????????????????
?                        ?       ?
?source-quench           ? 4     ?
??????????????????????????????????
?                        ?       ?
?redirect                ? 5     ?
```

```
??????????????????????????????????
?                     ?       ?
?echo-request         ? 8     ?
??????????????????????????????????
?                     ?       ?
?router-advertisement ? 9     ?
??????????????????????????????????
?                     ?       ?
?router-solicitation  ? 10    ?
??????????????????????????????????
?                     ?       ?
?time-exceeded        ? 11    ?
??????????????????????????????????
?                     ?       ?
?parameter-problem    ? 12    ?
??????????????????????????????????
?                     ?       ?
?timestamp-request    ? 13    ?
??????????????????????????????????
?                     ?       ?
?timestamp-reply      ? 14    ?
??????????????????????????????????
?                     ?       ?
?info-request         ? 15    ?
??????????????????????????????????
?                     ?       ?
?info-reply           ? 16    ?
??????????????????????????????????
?                     ?       ?
?address-mask-request ? 17    ?
??????????????????????????????????
?                     ?       ?
?address-mask-reply   ? 18    ?
```

???????????????????????????????????

ICMP Type specification.

```
# match ping packets
filter output icmp type { echo-request, echo-reply }
```

ICMP CODE TYPE

Name	Keyword	Size	Base type
ICMP Code	icmp_code	8 bit	integer

The ICMP Code type is used to conveniently specify the ICMP header?s code field.

Table 19. Keywords may be used when specifying the ICMP code

Keyword	Value
net-unreachable	0
host-unreachable	1
prot-unreachable	2
port-unreachable	3
frag-needed	4
net-prohibited	9

host-prohibited	10
admin-prohibited	13

ICMPV6 TYPE TYPE

Name	Keyword	Size	Base type
ICMPv6 Type	icmpx_code	8 bit	integer

The ICMPv6 Type type is used to conveniently specify the ICMPv6 header?s type field.

Table 20. keywords may be used when specifying the ICMPv6 type:

Keyword	Value
destination-unreachable	1
packet-too-big	2
time-exceeded	3
parameter-problem	4
echo-request	128

```
??????????????????????????????????
?                         ?       ?
?echo-reply               ? 129   ?
??????????????????????????????????
?                         ?       ?
?mld-listener-query       ? 130   ?
??????????????????????????????????
?                         ?       ?
?mld-listener-report      ? 131   ?
??????????????????????????????????
?                         ?       ?
?mld-listener-done        ? 132   ?
??????????????????????????????????
?                         ?       ?
?mld-listener-reduction   ? 132   ?
??????????????????????????????????
?                         ?       ?
?nd-router-solicit        ? 133   ?
??????????????????????????????????
?                         ?       ?
?nd-router-advert         ? 134   ?
??????????????????????????????????
?                         ?       ?
?nd-neighbor-solicit      ? 135   ?
??????????????????????????????????
?                         ?       ?
?nd-neighbor-advert       ? 136   ?
??????????????????????????????????
?                         ?       ?
?nd-redirect              ? 137   ?
??????????????????????????????????
?                         ?       ?
?router-renumbering       ? 138   ?
```

ind-neighbor-solicit	141
ind-neighbor-advert	142
mld2-listener-report	143

ICMPv6 Type specification.

```
# match ICMPv6 ping packets
filter output icmpv6 type { echo-request, echo-reply }
```

ICMPV6 CODE TYPE

Name	Keyword	Size	Base type
ICMPv6 Code	icmpv6_code	8 bit	integer

The ICMPv6 Code type is used to conveniently specify the ICMPv6 header?s code field.

Table 21. keywords may be used when specifying the ICMPv6 code

Keyword	Value
no-route	0
admin-prohibited	1
addr-unreachable	3

???????????????????????????

?   ?   ?

?port-unreachable ? 4   ?

???????????????????????????

?   ?   ?

?policy-fail   ? 5   ?

???????????????????????????

?   ?   ?

?reject-route   ? 6   ?

???????????????????????????

ICMPVX CODE TYPE

??

?Name   ? Keyword   ? Size ? Base type ?

??

?   ?   ?   ?   ?

?ICMPvX Code ? icmpv6_type ? 8 bit ? integer   ?

??

The ICMPvX Code type abstraction is a set of values which overlap between ICMP and ICMPv6

Code types to be used from the inet family.

Table 22. keywords may be used when specifying the ICMPvX code

???????????????????????????

?Keyword   ? Value ?

???????????????????????????

?   ?   ?

?no-route   ? 0   ?

???????????????????????????

?   ?   ?

?port-unreachable ? 1   ?

???????????????????????????

?   ?   ?

?host-unreachable ? 2   ?

???????????????????????????

?   ?   ?

?admin-prohibited ? 3 ?

???????????????????????????

CONNTRACK TYPES

Table 23. overview of types used in ct expression and statement

??

?Name ? Keyword ? Size ? Base type ?

??

? ? ? ? ?

?conntrack state ? ct_state ? 4 byte ? bitmask ?

??

? ? ? ? ?

?conntrack direction ? ct_dir ? 8 bit ? integer ?

??

? ? ? ? ?

?conntrack status ? ct_status ? 4 byte ? bitmask ?

??

? ? ? ? ?

?conntrack event ? ct_event ? 4 byte ? bitmask ?

?bits ? ? ? ?

??

? ? ? ? ?

?conntrack label ? ct_label ? 128 bit ? bitmask ?

??

For each of the types above, keywords are available for convenience:

Table 24. conntrack state (ct_state)

??????????????????????

?Keyword ? Value ?

??????????????????????

? ? ?

?invalid ? 1 ?

??????????????????????

? ? ?

?established ? 2 ?

related	4
new	8
untracked	64

Table 25. conntrack direction (ct_dir)

Keyword	Value
original	0
reply	1

Table 26. conntrack status (ct_status)

Keyword	Value
expected	1
seen-reply	2
assured	4

?   ?   ?

?confirmed ? 8   ?

?????????????????????

?   ?   ?

?snat   ? 16   ?

?????????????????????

?   ?   ?

?dnat   ? 32   ?

?????????????????????

?   ?   ?

?dying   ? 512   ?

?????????????????????

Table 27. conntrack event bits (ct_event)

????????????????????

?Keyword   ? Value ?

????????????????????

?   ?   ?

?new   ? 1   ?

????????????????????

?   ?   ?

?related   ? 2   ?

????????????????????

?   ?   ?

?destroy   ? 4   ?

????????????????????

?   ?   ?

?reply   ? 8   ?

????????????????????

?   ?   ?

?assured   ? 16   ?

????????????????????

?   ?   ?

?protoinfo ? 32   ?

```
????????????????????
?          ?       ?
?helper    ? 64    ?
????????????????????
?          ?       ?
?mark      ? 128   ?
????????????????????
?          ?       ?
?seqadj    ? 256   ?
????????????????????
?          ?       ?
?secmark   ? 512   ?
????????????????????
?          ?       ?
?label     ? 1024  ?
????????????????????
```

Possible keywords for conntrack label type (ct_label) are read at runtime from /etc/connlabel.conf.

DCCP PKTTYPE TYPE

```
?????????????????????????????????????????????????????
?Name             ? Keyword      ? Size  ? Base type ?
?????????????????????????????????????????????????????
?                 ?              ?       ?           ?
?DCCP packet type ? dccp_pkttype ? 4 bit ? integer   ?
?????????????????????????????????????????????????????
```

The DCCP packet type abstracts the different legal values of the respective four bit field in the DCCP header, as stated by RFC4340. Note that possible values 10-15 are considered reserved and therefore not allowed to be used. In iptables' dccp match, these values are aliased INVALID. With nftables, one may simply match on the numeric value range, i.e. 10-15.

Table 28. keywords may be used when specifying the DCCP packet type

```
???????????????????
?Keyword  ? Value ?
```

```
???????????????????
?         ?       ?
?request  ? 0     ?
???????????????????
?         ?       ?
?response ? 1     ?
???????????????????
?         ?       ?
?data     ? 2     ?
???????????????????
?         ?       ?
?ack      ? 3     ?
???????????????????
?         ?       ?
?dataack  ? 4     ?
???????????????????
?         ?       ?
?closereq ? 5     ?
???????????????????
?         ?       ?
?close    ? 6     ?
???????????????????
?         ?       ?
?reset    ? 7     ?
???????????????????
?         ?       ?
?sync     ? 8     ?
???????????????????
?         ?       ?
?syncack  ? 9     ?
???????????????????
```

PRIMARY EXPRESSIONS

The lowest order expression is a primary expression, representing either a constant or a single datum from a packet?s payload, meta data or a stateful module.

META EXPRESSIONS

meta {length | nfproto | l4proto | protocol | priority}

[meta] {mark | iif | iifname | iiftype | oif | oifname | oiftype | skuid | skgid | nftrace | rtclassid | ibrname | obrname | pkttype | cpu | iifgroup | oifgroup | cgroup | random | ipsec | iifkind | oifkind | time | hour | day }

A meta expression refers to meta data associated with a packet.

There are two types of meta expressions: unqualified and qualified meta expressions. Qualified meta expressions require the meta keyword before the meta key, unqualified meta expressions can be specified by using the meta key directly or as qualified meta expressions. Meta l4proto is useful to match a particular transport protocol that is part of either an IPv4 or IPv6 packet. It will also skip any IPv6 extension headers present in an IPv6 packet.

meta iif, oif, iifname and oifname are used to match the interface a packet arrived on or is about to be sent out on.

iif and oif are used to match on the interface index, whereas iifname and oifname are used to match on the interface name. This is not the same ? assuming the rule

```
filter input meta iif "foo"
```

Then this rule can only be added if the interface "foo" exists. Also, the rule will continue to match even if the interface "foo" is renamed to "bar".

This is because internally the interface index is used. In case of dynamically created interfaces, such as tun/tap or dialup interfaces (ppp for example), it might be better to use iifname or oifname instead.

In these cases, the name is used so the interface doesn?t have to exist to add such a rule, it will stop matching if the interface gets renamed and it will match again in case interface gets deleted and later a new interface with the same name is created.

Like with iptables, wildcard matching on interface name prefixes is available for iifname and oifname matches by appending an asterisk (*) character. Note however that unlike iptables, nftables does not accept interface names consisting of the wildcard character only - users are supposed to just skip those always matching expressions. In order to match on literal asterisk character, one may escape it using backslash (\).

Table 29. Meta expression types

```
????????????????????????????????????????????????????????????????
?Keyword   ? Description              ? Type                ?
```

```
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?length   ? Length of the packet in  ? integer (32-bit)    ?
?      ? bytes            ?           ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?nfproto  ? real hook protocol    ? integer (32 bit)    ?
?      ? family, useful only in  ?           ?
?      ? inet table          ?           ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?l4proto  ? layer 4 protocol, skips ? integer (8 bit)    ?
?      ? ipv6 extension headers  ?           ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?protocol ? EtherType protocol value ? ether_type       ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?priority ? TC packet priority    ? tc_handle        ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?mark     ? Packet mark          ? mark            ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?iif     ? Input interface index  ? iface_index       ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?iifname  ? Input interface name   ? ifname          ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
?iiftype  ? Input interface type   ? iface_type       ?
?????????????????????????????????????????????????????????????
?      ?              ?           ?
```

```
?oif      ? Output interface index  ? iface_index      ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
?oifname  ? Output interface name   ? ifname           ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
?oiftype  ? Output interface        ? iface_type       ?
?         ? hardware type           ?                  ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
?sdif     ? Slave device input      ? iface_index      ?
?         ? interface index         ?                  ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
?sdifname ? Slave device interface  ? ifname           ?
?         ? name                    ?                  ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
?skuid    ? UID associated with     ? uid              ?
?         ? originating socket      ?                  ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
?skgid    ? GID associated with     ? gid              ?
?         ? originating socket      ?                  ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
?rtclassid ? Routing realm          ? realm            ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
?ibrname  ? Input bridge interface  ? ifname           ?
?         ? name                    ?                  ?
??????????????????????????????????????????????????????????????????
?         ?                         ?                  ?
```

```
?obrname   ? Output bridge interface  ? ifname            ?
?          ? name                     ?                   ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?pkttype   ? packet type              ? pkt_type          ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?cpu       ? cpu number processing    ? integer (32 bit)  ?
?          ? the packet               ?                   ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?iifgroup  ? incoming device group    ? devgroup          ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?oifgroup  ? outgoing device group    ? devgroup          ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?cgroup    ? control group id         ? integer (32 bit)  ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?random    ? pseudo-random number     ? integer (32 bit)  ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?ipsec     ? true if packet was ipsec ? boolean (1 bit)   ?
?          ? encrypted                ?                   ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?iifkind   ? Input interface kind     ?                   ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
?oifkind   ? Output interface kind    ?                   ?
????????????????????????????????????????????????????????????
?          ?                          ?                   ?
```

```
?time      ? Absolute time of packet  ? Integer (32 bit) or ?
?          ? reception                ? string              ?
????????????????????????????????????????????????????????????
?          ?                          ?                     ?
?day       ? Day of week              ? Integer (8 bit) or  ?
?          ?                          ? string              ?
????????????????????????????????????????????????????????????
?          ?                          ?                     ?
?hour      ? Hour of day              ? String              ?
????????????????????????????????????????????????????????????
```

Table 30. Meta expression specific types

```
??????????????????????????????????????????????????
?Type          ? Description                     ?
??????????????????????????????????????????????????
?              ?                                 ?
?iface_index   ? Interface index (32 bit number). ?
?              ? Can be specified numerically or  ?
?              ? as name of an existing          ?
?              ? interface.                      ?
??????????????????????????????????????????????????
?              ?                                 ?
?ifname        ? Interface name (16 byte string). ?
?              ? Does not have to exist.         ?
??????????????????????????????????????????????????
?              ?                                 ?
?iface_type    ? Interface type (16 bit number).  ?
??????????????????????????????????????????????????
?              ?                                 ?
?uid           ? User ID (32 bit number). Can be  ?
?              ? specified numerically or as user ?
?              ? name.                           ?
??????????????????????????????????????????????????
?              ?                                 ?
```

```
?gid           ? Group ID (32 bit number). Can be ?
?              ? specified numerically or as      ?
?              ? group name.                      ?
??????????????????????????????????????????????????
?              ?                                  ?
?realm         ? Routing Realm (32 bit number).   ?
?              ? Can be specified numerically or  ?
?              ? as symbolic name defined in      ?
?              ? /etc/iproute2/rt_realms.         ?
??????????????????????????????????????????????????
?              ?                                  ?
?devgroup_type ? Device group (32 bit number).    ?
?              ? Can be specified numerically or  ?
?              ? as symbolic name defined in      ?
?              ? /etc/iproute2/group.             ?
??????????????????????????????????????????????????
?              ?                                  ?
?pkt_type      ? Packet type: host (addressed to  ?
?              ? local host), broadcast (to all), ?
?              ? multicast (to group), other      ?
?              ? (addressed to another host).     ?
??????????????????????????????????????????????????
?              ?                                  ?
?ifkind        ? Interface kind (16 byte string). ?
?              ? See TYPES in ip-link(8) for a    ?
?              ? list.                            ?
??????????????????????????????????????????????????
?              ?                                  ?
?time          ? Either an integer or a date in   ?
?              ? ISO format. For example:         ?
?              ? "2019-06-06 17:00". Hour and     ?
?              ? seconds are optional and can be  ?
?              ? omitted if desired. If omitted,  ?
?         ? midnight will be assumed. The  ?
?         ? following three would be       ?
?         ? equivalent: "2019-06-06",      ?
?         ? "2019-06-06 00:00" and         ?
?         ? "2019-06-06 00:00:00". When an ?
?         ? integer is given, it is assumed ?
?         ? to be a UNIX timestamp.        ?
???????????????????????????????????????????????????
?         ?                                ?
?day      ? Either a day of week ("Monday", ?
?         ? "Tuesday", etc.), or an integer ?
?         ? between 0 and 6. Strings are   ?
?         ? matched case-insensitively, and ?
?         ? a full match is not expected   ?
?         ? (e.g. "Mon" would match        ?
?         ? "Monday"). When an integer is  ?
?         ? given, 0 is Sunday and 6 is    ?
?         ? Saturday.                      ?
???????????????????????????????????????????????????
?         ?                                ?
?hour     ? A string representing an hour in ?
?         ? 24-hour format. Seconds can    ?
?         ? optionally be specified. For   ?
?         ? example, 17:00 and 17:00:00    ?
?         ? would be equivalent.           ?
???????????????????????????????????????????????????
```

Using meta expressions.

```
  # qualified meta expression
  filter output meta oif eth0
  filter forward meta iifkind { "tun", "veth" }
  # unqualified meta expression
  filter output oif eth0
  # incoming packet was subject to ipsec processing
```

```
raw prerouting meta ipsec exists accept
```

SOCKET EXPRESSION

```
socket {transparent | mark | wildcard}
socket cgroupv2 level NUM
```

Socket expression can be used to search for an existing open TCP/UDP socket and its attributes that can be associated with a packet. It looks for an established or non-zero bound listening socket (possibly with a non-local address). You can also use it to match on the socket cgroupv2 at a given ancestor level, e.g. if the socket belongs to cgroupv2 a/b, ancestor level 1 checks for a matching on cgroup a and ancestor level 2 checks for a matching on cgroup b.

Table 31. Available socket attributes

Name	Description	Type
transparent	Value of the IP_TRANSPARENT socket option in the found socket. It can be 0 or 1.	boolean (1 bit)
mark	Value of the socket mark (SOL_SOCKET, SO_MARK).	mark
wildcard	Indicates whether the socket is wildcard-bound (e.g. 0.0.0.0 or ::0).	boolean (1 bit)
cgroupv2	cgroup version 2 for this socket (path from	cgroupv2

?   ? /sys/fs/cgroup)   ?   ?

??

Using socket expression.

```
   # Mark packets that correspond to a transparent socket. "socket wildcard 0"
   # means that zero-bound listener sockets are NOT matched (which is usually
   # exactly what you want).
   table inet x {
      chain y {
         type filter hook prerouting priority mangle; policy accept;
         socket transparent 1 socket wildcard 0 mark set 0x00000001 accept
      }
   }
   # Trace packets that corresponds to a socket with a mark value of 15
   table inet x {
      chain y {
         type filter hook prerouting priority mangle; policy accept;
         socket mark 0x0000000f nftrace set 1
      }
   }
   # Set packet mark to socket mark
   table inet x {
      chain y {
         type filter hook prerouting priority mangle; policy accept;
         tcp dport 8080 mark set socket mark
      }
   }
   # Count packets for cgroupv2 "user.slice" at level 1
   table inet x {
      chain y {
         type filter hook input priority filter; policy accept;
         socket cgroupv2 level 1 "user.slice" counter
      }
   }
```

OSF EXPRESSION

osf [ttl {loose | skip}] {name | version}

The osf expression does passive operating system fingerprinting. This expression compares some data (Window Size, MSS, options and their order, DF, and others) from packets with the SYN bit set.

Table 32. Available osf attributes

Name	Description	Type
ttl	Do TTL checks on the packet to determine the operating system.	string
version	Do OS version checks on the packet.	
name	Name of the OS signature to match. All signatures can be found at pf.os file. Use "unknown" for OS signatures that the expression could not detect.	string

Available ttl values.

If no TTL attribute is passed, make a true IP header and fingerprint TTL true comparison. This generally works for LANs.

* loose: Check if the IP header's TTL is less than the fingerprint one. Works for globally-routable addresses.

* skip: Do not compare the TTL at all.

Using osf expression.

```
    # Accept packets that match the "Linux" OS genre signature without comparing TTL.
    table inet x {
        chain y {
            type filter hook input priority filter; policy accept;
            osf ttl skip name "Linux"
        }
    }
```

FIB EXPRESSIONS

```
    fib {saddr | daddr | mark | iif | oif} [. ...] {oif | oifname | type}
```

A fib expression queries the fib (forwarding information base) to obtain information such as the output interface index a particular address would use. The input is a tuple of elements that is used as input to the fib lookup functions.

Table 33. fib expression specific types

```
??????????????????????????????????????????????????????
?Keyword ? Description           ? Type            ?
??????????????????????????????????????????????????????
?        ?                        ?                 ?
?oif     ? Output interface index ? integer (32 bit) ?
??????????????????????????????????????????????????????
?        ?                        ?                 ?
?oifname ? Output interface name  ? string          ?
??????????????????????????????????????????????????????
?        ?                        ?                 ?
?type    ? Address type           ? fib_addrtype    ?
??????????????????????????????????????????????????????
```

Use nft describe fib_addrtype to get a list of all address types.

Using fib expressions.

```
    # drop packets without a reverse path
    filter prerouting fib saddr . iif oif missing drop
```

In this example, 'saddr . iif' looks up routing information based on the source address and the input interface.

oif picks the output interface index from the routing information.

If no route was found for the source address/input interface combination, the output interface index is zero.

In case the input interface is specified as part of the input key, the output interface index is always the same as the input interface index or zero.

If only 'saddr oif' is given, then oif can be any interface index or zero.

```
# drop packets to address not configured on incoming interface
filter prerouting fib daddr . iif type != { local, broadcast, multicast } drop
# perform lookup in a specific 'blackhole' table (0xdead, needs ip appropriate ip rule)
filter prerouting meta mark set 0xdead fib daddr . mark type vmap { blackhole : drop, prohibit : jump prohibited, unreachable : drop }
```

ROUTING EXPRESSIONS

rt [ip | ip6] {classid | nexthop | mtu | ipsec}

A routing expression refers to routing data associated with a packet.

Table 34. Routing expression types

Keyword	Description	Type
classid	Routing realm	realm
nexthop	Routing nexthop	ipv4_addr/ipv6_addr
mtu	TCP maximum segment size of route	integer (16 bit)
ipsec	route via ipsec tunnel or transport	boolean

Table 35. Routing expression specific types

Type	Description

```
?realm ? Routing Realm (32 bit number).  ?
?      ? Can be specified numerically or ?
?      ? as symbolic name defined in     ?
?      ? /etc/iproute2/rt_realms.        ?
??????????????????????????????????????????
```

Using routing expressions.

```
# IP family independent rt expression
filter output rt classid 10
# IP family dependent rt expressions
ip filter output rt nexthop 192.168.0.1
ip6 filter output rt nexthop fd00::1
inet filter output rt ip nexthop 192.168.0.1
inet filter output rt ip6 nexthop fd00::1
# outgoing packet will be encapsulated/encrypted by ipsec
filter output rt ipsec exists
```

IPSEC EXPRESSIONS

```
ipsec {in | out} [ spnum NUM ]  {reqid | spi}
ipsec {in | out} [ spnum NUM ]  {ip | ip6} {saddr | daddr}
```

An ipsec expression refers to ipsec data associated with a packet.

The in or out keyword needs to be used to specify if the expression should examine inbound or outbound policies. The in keyword can be used in the prerouting, input and forward hooks. The out keyword applies to forward, output and postrouting hooks. The optional keyword spnum can be used to match a specific state in a chain, it defaults to 0.

Table 36. Ipsec expression types

```
???????????????????????????????????????????????????????????
?Keyword ? Description              ? Type                ?
???????????????????????????????????????????????????????????
?        ?                          ?                     ?
?reqid   ? Request ID               ? integer (32 bit)    ?
???????????????????????????????????????????????????????????
?        ?                          ?                     ?
?spi     ? Security Parameter Index ? integer (32 bit)    ?
???????????????????????????????????????????????????????????
```

```
?      ?                  ?               ?
?saddr  ? Source address of the   ? ipv4_addr/ipv6_addr ?
?      ? tunnel              ?               ?
???????????????????????????????????????????????????????????
?      ?                  ?               ?
?daddr  ? Destination address of  ? ipv4_addr/ipv6_addr ?
?      ? the tunnel           ?               ?
???????????????????????????????????????????????????????????
```

NUMGEN EXPRESSION

```
numgen {inc | random} mod NUM [ offset NUM ]
```

Create a number generator. The inc or random keywords control its operation mode: In inc mode, the last returned value is simply incremented. In random mode, a new random number is returned. The value after mod keyword specifies an upper boundary (read: modulus) which is not reached by returned numbers. The optional offset allows to increment the returned value by a fixed offset.

A typical use-case for numgen is load-balancing:

Using numgen expression.

```
# round-robin between 192.168.10.100 and 192.168.20.200:
add rule nat prerouting dnat to numgen inc mod 2 map \
        { 0 : 192.168.10.100, 1 : 192.168.20.200 }
# probability-based with odd bias using intervals:
add rule nat prerouting dnat to numgen random mod 10 map \
        { 0-2 : 192.168.10.100, 3-9 : 192.168.20.200 }
```

HASH EXPRESSIONS

```
jhash {ip saddr | ip6 daddr | tcp dport | udp sport | ether saddr} [. ...] mod NUM [ seed NUM ] [ offset NUM ]
symhash mod NUM [ offset NUM ]
```

Use a hashing function to generate a number. The functions available are jhash, known as Jenkins Hash, and symhash, for Symmetric Hash. The jhash requires an expression to determine the parameters of the packet header to apply the hashing, concatenations are possible as well. The value after mod keyword specifies an upper boundary (read: modulus) which is not reached by returned numbers. The optional seed is used to specify an init value used as seed in the hashing function. The optional offset allows to increment the returned value by a fixed offset.

A typical use-case for jhash and symhash is load-balancing:

Using hash expressions.

```
# load balance based on source ip between 2 ip addresses:
add rule nat prerouting dnat to jhash ip saddr mod 2 map \
        { 0 : 192.168.10.100, 1 : 192.168.20.200 }
# symmetric load balancing between 2 ip addresses:
add rule nat prerouting dnat to symhash mod 2 map \
        { 0 : 192.168.10.100, 1 : 192.168.20.200 }
```

PAYLOAD EXPRESSIONS

Payload expressions refer to data from the packet?s payload.

ETHERNET HEADER EXPRESSION

```
ether {daddr | saddr | type}
```

Table 37. Ethernet header expression types

Keyword	Description	Type
daddr	Destination MAC address	ether_addr
saddr	Source MAC address	ether_addr
type	EtherType	ether_type

VLAN HEADER EXPRESSION

```
vlan {id | dei | pcp | type}
```

Table 38. VLAN header expression

Keyword	Description	Type
id	VLAN ID (VID)	integer (12 bit)

dei	Drop Eligible Indicator	integer (1 bit)
pcp	Priority code point	integer (3 bit)
type	EtherType	ether_type

ARP HEADER EXPRESSION

```
arp {htype | ptype | hlen | plen | operation | saddr { ip | ether } | daddr { ip | ether }
```

Table 39. ARP header expression

Keyword	Description	Type
htype	ARP hardware type	integer (16 bit)
ptype	EtherType	ether_type
hlen	Hardware address len	integer (8 bit)
plen	Protocol address len	integer (8 bit)
operation	Operation	arp_op
saddr ether	Ethernet sender address	ether_addr

```
???????????????????????????????????????????????????????????
?       ?              ?         ?
?daddr ether ? Ethernet target address ? ether_addr   ?
???????????????????????????????????????????????????????????
?       ?              ?         ?
?saddr ip  ? IPv4 sender address   ? ipv4_addr    ?
???????????????????????????????????????????????????????????
?       ?              ?         ?
?daddr ip  ? IPv4 target address   ? ipv4_addr    ?
???????????????????????????????????????????????????????????
```

IPV4 HEADER EXPRESSION

```
ip {version | hdrlength | dscp | ecn | length | id | frag-off | ttl | protocol | checksum | saddr | daddr }
```

Table 40. IPv4 header expression

```
??????????????????????????????????????????????????????????????
?Keyword  ? Description       ? Type          ?
??????????????????????????????????????????????????????????????
?      ?              ?            ?
?version  ? IP header version (4)  ? integer (4 bit)    ?
??????????????????????????????????????????????????????????????
?      ?              ?            ?
?hdrlength ? IP header length     ? integer (4 bit) FIXME ?
?      ? including options    ? scaling        ?
??????????????????????????????????????????????????????????????
?      ?              ?            ?
?dscp    ? Differentiated Services ? dscp          ?
?      ? Code Point        ?            ?
??????????????????????????????????????????????????????????????
?      ?              ?            ?
?ecn     ? Explicit Congestion   ? ecn          ?
?      ? Notification       ?            ?
??????????????????????????????????????????????????????????????
?      ?              ?            ?
?length   ? Total packet length   ? integer (16 bit)   ?
```

```
?????????????????????????????????????????????????????????????
?          ?                      ?                     ?
?id        ? IP ID                ? integer (16 bit)     ?
?????????????????????????????????????????????????????????????
?          ?                      ?                     ?
?frag-off  ? Fragment offset      ? integer (16 bit)     ?
?????????????????????????????????????????????????????????????
?          ?                      ?                     ?
?ttl       ? Time to live         ? integer (8 bit)      ?
?????????????????????????????????????????????????????????????
?          ?                      ?                     ?
?protocol  ? Upper layer protocol ? inet_proto           ?
?????????????????????????????????????????????????????????????
?          ?                      ?                     ?
?checksum  ? IP header checksum   ? integer (16 bit)     ?
?????????????????????????????????????????????????????????????
?          ?                      ?                     ?
?saddr     ? Source address       ? ipv4_addr            ?
?????????????????????????????????????????????????????????????
?          ?                      ?                     ?
?daddr     ? Destination address  ? ipv4_addr            ?
?????????????????????????????????????????????????????????????
```

ICMP HEADER EXPRESSION

```
icmp {type | code | checksum | id | sequence | gateway | mtu}
```

This expression refers to ICMP header fields. When using it in inet, bridge or netdev families, it will cause an implicit dependency on IPv4 to be created. To match on unusual cases like ICMP over IPv6, one has to add an explicit meta protocol ip6 match to the rule.

Table 41. ICMP header expression

```
???????????????????????????????????????????????????????
?Keyword  ? Description          ? Type            ?
???????????????????????????????????????????????????????
?         ?                      ?                 ?
?type     ? ICMP type field      ? icmp_type       ?
```

```
???????????????????????????????????????????????????????
?         ?                         ?                 ?
?code     ? ICMP code field         ? integer (8 bit)  ?
???????????????????????????????????????????????????????
?         ?                         ?                 ?
?checksum ? ICMP checksum field     ? integer (16 bit) ?
???????????????????????????????????????????????????????
?         ?                         ?                 ?
?id       ? ID of echo              ? integer (16 bit) ?
?         ? request/response        ?                 ?
???????????????????????????????????????????????????????
?         ?                         ?                 ?
?sequence ? sequence number of echo ? integer (16 bit) ?
?         ? request/response        ?                 ?
???????????????????????????????????????????????????????
?         ?                         ?                 ?
?gateway  ? gateway of redirects    ? integer (32 bit) ?
???????????????????????????????????????????????????????
?         ?                         ?                 ?
?mtu      ? MTU of path MTU         ? integer (16 bit) ?
?         ? discovery               ?                 ?
???????????????????????????????????????????????????????
```

IGMP HEADER EXPRESSION

    igmp {type | mrt | checksum | group}

This expression refers to IGMP header fields. When using it in inet, bridge or netdev families, it will cause an implicit dependency on IPv4 to be created. To match on unusual cases like IGMP over IPv6, one has to add an explicit meta protocol ip6 match to the rule.

Table 42. IGMP header expression

```
?????????????????????????????????????????????????????
?Keyword  ? Description           ? Type            ?
?????????????????????????????????????????????????????
?         ?                       ?                 ?
?type     ? IGMP type field       ? igmp_type       ?
```

```
?????????????????????????????????????????????????????
?        ?                      ?                  ?
?mrt     ? IGMP maximum response ? integer (8 bit)  ?
?        ? time field           ?                  ?
?????????????????????????????????????????????????????
?        ?                      ?                  ?
?checksum ? IGMP checksum field   ? integer (16 bit) ?
?????????????????????????????????????????????????????
?        ?                      ?                  ?
?group    ? Group address        ? integer (32 bit) ?
?????????????????????????????????????????????????????
```

IPV6 HEADER EXPRESSION

ip6 {version | dscp | ecn | flowlabel | length | nexthdr | hoplimit | saddr | daddr}

This expression refers to the ipv6 header fields. Caution when using ip6 nexthdr, the value only refers to the next header, i.e. ip6 nexthdr tcp will only match if the ipv6 packet does not contain any extension headers. Packets that are fragmented or e.g. contain a routing extension headers will not be matched. Please use meta l4proto if you wish to match the real transport header and ignore any additional extension headers instead.

Table 43. IPv6 header expression

```
????????????????????????????????????????????????????????
?Keyword   ? Description            ? Type             ?
????????????????????????????????????????????????????????
?          ?                        ?                  ?
?version   ? IP header version (6)   ? integer (4 bit)  ?
????????????????????????????????????????????????????????
?          ?                        ?                  ?
?dscp      ? Differentiated Services ? dscp             ?
?          ? Code Point             ?                  ?
????????????????????????????????????????????????????????
?          ?                        ?                  ?
?ecn       ? Explicit Congestion     ? ecn              ?
?          ? Notification           ?                  ?
????????????????????????????????????????????????????????
```

flowlabel	Flow label	integer (20 bit)
length	Payload length	integer (16 bit)
nexthdr	Nexthdr protocol	inet_proto
hoplimit	Hop limit	integer (8 bit)
saddr	Source address	ipv6_addr
daddr	Destination address	ipv6_addr

Using ip6 header expressions.

```
# matching if first extension header indicates a fragment
ip6 nexthdr ipv6-frag
```

ICMPV6 HEADER EXPRESSION

```
icmpv6 {type | code | checksum | parameter-problem | packet-too-big | id | sequence | max-delay}
```

This expression refers to ICMPv6 header fields. When using it in inet, bridge or netdev families, it will cause an implicit dependency on IPv6 to be created. To match on unusual cases like ICMPv6 over IPv4, one has to add an explicit meta protocol ip match to the rule.

Table 44. ICMPv6 header expression

Keyword	Description	Type
type	ICMPv6 type field	icmpv6_type

```
????????????????????????????????????????????????????????????????
?                 ?                        ?                 ?
?code             ? ICMPv6 code field      ? integer (8 bit)  ?
????????????????????????????????????????????????????????????????
?                 ?                        ?                 ?
?checksum         ? ICMPv6 checksum field  ? integer (16 bit) ?
????????????????????????????????????????????????????????????????
?                 ?                        ?                 ?
?parameter-problem ? pointer to problem     ? integer (32 bit) ?
????????????????????????????????????????????????????????????????
?                 ?                        ?                 ?
?packet-too-big   ? oversized MTU          ? integer (32 bit) ?
????????????????????????????????????????????????????????????????
?                 ?                        ?                 ?
?id               ? ID of echo             ? integer (16 bit) ?
?                 ? request/response       ?                 ?
????????????????????????????????????????????????????????????????
?                 ?                        ?                 ?
?sequence         ? sequence number of echo ? integer (16 bit) ?
?                 ? request/response       ?                 ?
????????????????????????????????????????????????????????????????
?                 ?                        ?                 ?
?max-delay        ? maximum response delay  ? integer (16 bit) ?
?                 ? of MLD queries         ?                 ?
????????????????????????????????????????????????????????????????
```

TCP HEADER EXPRESSION

    tcp {sport | dport | sequence | ackseq | doff | reserved | flags | window | checksum | urgptr}

Table 45. TCP header expression

```
???????????????????????????????????????????????????????????
?Keyword  ? Description            ? Type             ?
???????????????????????????????????????????????????????????
?         ?                        ?                  ?
?sport    ? Source port            ? inet_service     ?
```

```
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?dport   ? Destination port ? inet_service    ?
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?sequence ? Sequence number  ? integer (32 bit) ?
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?ackseq  ? Acknowledgement number ? integer (32 bit) ?
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?doff    ? Data offset      ? integer (4 bit) FIXME ?
?        ?                  ? scaling         ?
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?reserved ? Reserved area    ? integer (4 bit)  ?
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?flags   ? TCP flags        ? tcp_flag        ?
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?window  ? Window           ? integer (16 bit) ?
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?checksum ? Checksum         ? integer (16 bit) ?
????????????????????????????????????????????????????????????
?        ?                  ?                 ?
?urgptr  ? Urgent pointer   ? integer (16 bit) ?
????????????????????????????????????????????????????????????
```

UDP HEADER EXPRESSION

    udp {sport | dport | length | checksum}

Table 46. UDP header expression

```
????????????????????????????????????????????????????
```

Keyword	Description	Type
sport	Source port	inet_service
dport	Destination port	inet_service
length	Total packet length	integer (16 bit)
checksum	Checksum	integer (16 bit)

UDP-LITE HEADER EXPRESSION

udplite {sport | dport | checksum}

Table 47. UDP-Lite header expression

Keyword	Description	Type
sport	Source port	inet_service
dport	Destination port	inet_service
checksum	Checksum	integer (16 bit)

SCTP HEADER EXPRESSION

sctp {sport | dport | vtag | checksum}

sctp chunk CHUNK [FIELD]

CHUNK := data | init | init-ack | sack | heartbeat |

```
           heartbeat-ack | abort | shutdown | shutdown-ack | error |
           cookie-echo | cookie-ack | ecne | cwr | shutdown-complete
           | asconf-ack | forward-tsn | asconf
    FIELD := COMMON_FIELD | DATA_FIELD | INIT_FIELD | INIT_ACK_FIELD |
           SACK_FIELD | SHUTDOWN_FIELD | ECNE_FIELD | CWR_FIELD |
           ASCONF_ACK_FIELD | FORWARD_TSN_FIELD | ASCONF_FIELD
    COMMON_FIELD := type | flags | length
    DATA_FIELD := tsn | stream | ssn | ppid
    INIT_FIELD := init-tag | a-rwnd | num-outbound-streams |
                num-inbound-streams | initial-tsn
    INIT_ACK_FIELD := INIT_FIELD
    SACK_FIELD := cum-tsn-ack | a-rwnd | num-gap-ack-blocks |
                num-dup-tsns
    SHUTDOWN_FIELD := cum-tsn-ack
    ECNE_FIELD := lowest-tsn
    CWR_FIELD := lowest-tsn
    ASCONF_ACK_FIELD := seqno
    FORWARD_TSN_FIELD := new-cum-tsn
    ASCONF_FIELD := seqno
```

Table 48. SCTP header expression

Keyword	Description	Type
sport	Source port	inet_service
dport	Destination port	inet_service
vtag	Verification Tag	integer (32 bit)

checksum	Checksum	integer (32 bit)
chunk	Search chunk in packet	without FIELD, boolean indicating existence

Table 49. SCTP chunk fields

Name	Width in bits	Chunk	Notes
type	8	all	not useful, defined by chunk type
flags	8	all	semantics defined on per-chunk basis
length	16	all	length of this chunk in bytes excluding padding
tsn	32	data	transmission sequence number
stream	16	data	stream identifier
ssn	16	data	stream sequence number

ppid	32	data	payload protocol identifier
init-tag	32	init, init-ack	initiate tag
a-rwnd	32	init, init-ack, sack	advertised receiver window credit
num-outbound-streams	16	init, init-ack	number of outbound streams
num-inbound-streams	16	init, init-ack	number of inbound streams
initial-tsn	32	init, init-ack	initial transmit sequence number
cum-tsn-ack	32	sack, shutdown	cumulative transmission sequence number acknowledged
num-gap-ack-blocks	16	sack	number of Gap Ack Blocks included

```
???????????????????????????????????????????????????????????????????????????????????????
?            ?         ?            ?            ?
?num-dup-tsns     ? 16       ? sack        ? number of duplicate ?
?            ?         ?            ? transmission      ?
?            ?         ?            ? sequence numbers  ?
?            ?         ?            ? received          ?
???????????????????????????????????????????????????????????????????????????????????????
?            ?         ?            ?            ?
?lowest-tsn      ? 32       ? ecne, cwr     ? lowest transmission ?
?            ?         ?            ? sequence number   ?
???????????????????????????????????????????????????????????????????????????????????????
?            ?         ?            ?            ?
?seqno          ? 32       ? asconf-ack, asconf ? sequence number   ?
???????????????????????????????????????????????????????????????????????????????????????
?            ?         ?            ?            ?
?new-cum-tsn      ? 32       ? forward-tsn    ? new cumulative    ?
?            ?         ?            ? transmission      ?
?            ?         ?            ? sequence number   ?
???????????????????????????????????????????????????????????????????????????????????????
```

DCCP HEADER EXPRESSION

dccp {sport | dport | type}

Table 50. DCCP header expression

```
???????????????????????????????????????????
?Keyword ? Description      ? Type         ?
???????????????????????????????????????????
?        ?                  ?              ?
?sport   ? Source port      ? inet_service ?
???????????????????????????????????????????
?        ?                  ?              ?
?dport   ? Destination port ? inet_service ?
???????????????????????????????????????????
?        ?                  ?              ?
?type    ? Packet type      ? dccp_pkttype ?
```

AUTHENTICATION HEADER EXPRESSION

ah {nexthdr | hdrlength | reserved | spi | sequence}

Table 51. AH header expression

Keyword	Description	Type
nexthdr	Next header protocol	inet_proto
hdrlength	AH Header length	integer (8 bit)
reserved	Reserved area	integer (16 bit)
spi	Security Parameter Index	integer (32 bit)
sequence	Sequence number	integer (32 bit)

ENCRYPTED SECURITY PAYLOAD HEADER EXPRESSION

esp {spi | sequence}

Table 52. ESP header expression

Keyword	Description	Type
spi	Security Parameter Index	integer (32 bit)
sequence	Sequence number	integer (32 bit)

IPCOMP HEADER EXPRESSION

```
comp {nexthdr | flags | cpi}
```

Table 53. IPComp header expression

Keyword	Description	Type
nexthdr	Next header protocol	inet_proto
flags	Flags	bitmask
cpi	compression Parameter Index	integer (16 bit)

RAW PAYLOAD EXPRESSION

```
@base,offset,length
```

The raw payload expression instructs to load length bits starting at offset bits. Bit 0 refers to the very first bit ? in the C programming language, this corresponds to the topmost bit, i.e. 0x80 in case of an octet. They are useful to match headers that do not have a human-readable template expression yet. Note that nft will not add dependencies for Raw payload expressions. If you e.g. want to match protocol fields of a transport header with protocol number 5, you need to manually exclude packets that have a different transport header, for instance by using meta l4proto 5 before the raw expression.

Table 54. Supported payload protocol bases

Base	Description
ll	Link layer, for example the Ethernet header

```
??????????????????????????????????????????
?     ?                                  ?
?nh   ? Network header, for example IPv4 ?
?     ? or IPv6                          ?
??????????????????????????????????????????
?     ?                                  ?
?th   ? Transport Header, for example    ?
?     ? TCP                              ?
??????????????????????????????????????????
```

Matching destination port of both UDP and TCP.

```
inet filter input meta l4proto {tcp, udp} @th,16,16 { 53, 80 }
```

The above can also be written as

```
inet filter input meta l4proto {tcp, udp} th dport { 53, 80 }
```

it is more convenient, but like the raw expression notation no dependencies are created or checked. It is the users responsibility to restrict matching to those header types that have a notion of ports. Otherwise, rules using raw expressions will errnously match unrelated packets, e.g. mis-interpreting ESP packets SPI field as a port.

Rewrite arp packet target hardware address if target protocol address matches a given address.

```
input meta iifname enp2s0 arp ptype 0x0800 arp htype 1 arp hlen 6 arp plen 4 @nh,192,32 0xc0a88f10 @nh,144,48 set 0x112233445566 accept
```

EXTENSION HEADER EXPRESSIONS

Extension header expressions refer to data from variable-sized protocol headers, such as IPv6 extension headers, TCP options and IPv4 options.

nftables currently supports matching (finding) a given ipv6 extension header, TCP option or IPv4 option.

```
hbh {nexthdr | hdrlength}
frag {nexthdr | frag-off | more-fragments | id}
rt {nexthdr | hdrlength | type | seg-left}
dst {nexthdr | hdrlength}
mh {nexthdr | hdrlength | checksum | type}
srh {flags | tag | sid | seg-left}
    tcp option {eol | nop | maxseg | window | sack-perm | sack | sack0 | sack1 | sack2 | sack3 | timestamp}
```

tcp_option_field

ip option { lsrr | ra | rr | ssrr } ip_option_field

The following syntaxes are valid only in a relational expression with boolean type on right-hand side for checking header existence only:

exthdr {hbh | frag | rt | dst | mh}

tcp option {eol | nop | maxseg | window | sack-perm | sack | sack0 | sack1 | sack2 | sack3 | timestamp}

ip option { lsrr | ra | rr | ssrr }

Table 55. IPv6 extension headers

Keyword	Description
hbh	Hop by Hop
rt	Routing Header
frag	Fragmentation header
dst	dst options
mh	Mobility Header
srh	Segment Routing Header

Table 56. TCP Options

Keyword	Description	TCP option fields

```
?          ?                         ?                      ?
?eol       ? End if option list      ? -                    ?
??????????????????????????????????????????????????????????????
?          ?                         ?                      ?
?nop       ? 1 Byte TCP Nop padding  ? -                    ?
?          ? option                  ?                      ?
??????????????????????????????????????????????????????????????
?          ?                         ?                      ?
?maxseg    ? TCP Maximum Segment Size ? length, size        ?
??????????????????????????????????????????????????????????????
?          ?                         ?                      ?
?window    ? TCP Window Scaling      ? length, count        ?
??????????????????????????????????????????????????????????????
?          ?                         ?                      ?
?sack-perm ? TCP SACK permitted      ? length               ?
??????????????????????????????????????????????????????????????
?          ?                         ?                      ?
?sack      ? TCP Selective           ? length, left, right  ?
?          ? Acknowledgement (alias  ?                      ?
?          ? of block 0)             ?                      ?
??????????????????????????????????????????????????????????????
?          ?                         ?                      ?
?sack0     ? TCP Selective           ? length, left, right  ?
?          ? Acknowledgement (block  ?                      ?
?          ? 0)                      ?                      ?
??????????????????????????????????????????????????????????????
?          ?                         ?                      ?
?sack1     ? TCP Selective           ? length, left, right  ?
?          ? Acknowledgement (block  ?                      ?
?          ? 1)                      ?                      ?
??????????????????????????????????????????????????????????????
?          ?                         ?                      ?
?sack2     ? TCP Selective           ? length, left, right  ?
```

```
?          ? Acknowledgement (block   ?                      ?
?          ? 2)                       ?                      ?
???????????????????????????????????????????????????????????????
?          ?                          ?                      ?
?sack3     ? TCP Selective            ? length, left, right  ?
?          ? Acknowledgement (block   ?                      ?
?          ? 3)                       ?                      ?
???????????????????????????????????????????????????????????????
?          ?                          ?                      ?
?timestamp ? TCP Timestamps           ? length, tsval, tsecr ?
???????????????????????????????????????????????????????????????
```

TCP option matching also supports raw expression syntax to access arbitrary options:

```
    tcp option
    tcp option @number,offset,length
```

Table 57. IP Options

```
???????????????????????????????????????????????????????????
?Keyword ? Description         ? IP option fields        ?
???????????????????????????????????????????????????????????
?        ?                     ?                         ?
?lsrr    ? Loose Source Route  ? type, length, ptr, addr ?
???????????????????????????????????????????????????????????
?        ?                     ?                         ?
?ra      ? Router Alert        ? type, length, value     ?
???????????????????????????????????????????????????????????
?        ?                     ?                         ?
?rr      ? Record Route        ? type, length, ptr, addr ?
???????????????????????????????????????????????????????????
?        ?                     ?                         ?
?ssrr    ? Strict Source Route ? type, length, ptr, addr ?
???????????????????????????????????????????????????????????
```

finding TCP options.

```
    filter input tcp option sack-perm exists counter
```

matching TCP options.

```
filter input tcp option maxseg size lt 536
```

matching IPv6 exthdr.

```
ip6 filter input frag more-fragments 1 counter
```

finding IP option.

```
filter input ip option lsrr exists counter
```

CONNTRACK EXPRESSIONS

Conntrack expressions refer to meta data of the connection tracking entry associated with a packet.

There are three types of conntrack expressions. Some conntrack expressions require the flow direction before the conntrack key, others must be used directly because they are direction agnostic. The packets, bytes and avgpkt keywords can be used with or without a direction. If the direction is omitted, the sum of the original and the reply direction is returned. The same is true for the zone, if a direction is given, the zone is only matched if the zone id is tied to the given direction.

```
ct {state | direction | status | mark | expiration | helper | label | count | id}
ct [original | reply] {l3proto | protocol | bytes | packets | avgpkt | zone}
ct {original | reply} {proto-src | proto-dst}
ct {original | reply} {ip | ip6} {saddr | daddr}
```

The conntrack-specific types in this table are described in the sub-section CONNTRACK TYPES above.

Table 58. Conntrack expressions

Keyword	Description	Type
state	State of the connection	ct_state
direction	Direction of the packet relative to the connection	ct_dir

```
?status     ? Status of the connection ? ct_status          ?
??????????????????????????????????????????????????????????????
?           ?                          ?                    ?
?mark       ? Connection mark          ? mark               ?
??????????????????????????????????????????????????????????????
?           ?                          ?                    ?
?expiration ? Connection expiration    ? time               ?
?           ? time                     ?                    ?
??????????????????????????????????????????????????????????????
?           ?                          ?                    ?
?helper     ? Helper associated with   ? string             ?
?           ? the connection           ?                    ?
??????????????????????????????????????????????????????????????
?           ?                          ?                    ?
?label      ? Connection tracking      ? ct_label           ?
?           ? label bit or symbolic    ?                    ?
?           ? name defined in          ?                    ?
?           ? connlabel.conf in the    ?                    ?
?           ? nftables include path    ?                    ?
??????????????????????????????????????????????????????????????
?           ?                          ?                    ?
?l3proto    ? Layer 3 protocol of the  ? nf_proto           ?
?           ? connection               ?                    ?
??????????????????????????????????????????????????????????????
?           ?                          ?                    ?
?saddr      ? Source address of the    ? ipv4_addr/ipv6_addr ?
?           ? connection for the given ?                    ?
?           ? direction                ?                    ?
??????????????????????????????????????????????????????????????
?           ?                          ?                    ?
?daddr      ? Destination address of   ? ipv4_addr/ipv6_addr ?
?           ? the connection for the   ?                    ?
?           ? given direction          ?                    ?
```

```
?????????????????????????????????????????????????????????????????
?         ?                      ?                 ?
?protocol ? Layer 4 protocol of the ? inet_proto   ?
?         ? connection for the given ?             ?
?         ? direction            ?                 ?
?????????????????????????????????????????????????????????????????
?         ?                      ?                 ?
?proto-src ? Layer 4 protocol source ? integer (16 bit) ?
?         ? for the given direction ?              ?
?????????????????????????????????????????????????????????????????
?         ?                      ?                 ?
?proto-dst ? Layer 4 protocol    ? integer (16 bit) ?
?         ? destination for the  ?                 ?
?         ? given direction      ?                 ?
?????????????????????????????????????????????????????????????????
?         ?                      ?                 ?
?packets  ? packet count seen in the ? integer (64 bit) ?
?         ? given direction or sum ?               ?
?         ? of original and reply ?                ?
?????????????????????????????????????????????????????????????????
?         ?                      ?                 ?
?bytes    ? byte count seen, see ? integer (64 bit) ?
?         ? description for packets ?              ?
?         ? keyword              ?                 ?
?????????????????????????????????????????????????????????????????
?         ?                      ?                 ?
?avgpkt   ? average bytes per    ? integer (64 bit) ?
?         ? packet, see description ?              ?
?         ? for packets keyword  ?                 ?
?????????????????????????????????????????????????????????????????
?         ?                      ?                 ?
?zone     ? conntrack zone       ? integer (16 bit) ?
?????????????????????????????????????????????????????????????????
```

```
?          ?                      ?                  ?
?count     ? number of current       ? integer (32 bit)   ?
?          ? connections           ?                  ?
?????????????????????????????????????????????????????????????
?          ?                      ?                  ?
?id        ? Connection id          ? ct_id             ?
?????????????????????????????????????????????????????????????
```

restrict the number of parallel connections to a server.

```
nft add set filter ssh_flood '{ type ipv4_addr; flags dynamic; }'
nft add rule filter input tcp dport 22 add @ssh_flood '{ ip saddr ct count over 2 }' reject
```

STATEMENTS

Statements represent actions to be performed. They can alter control flow (return, jump to a different chain, accept or drop the packet) or can perform actions, such as logging, rejecting a packet, etc.

Statements exist in two kinds. Terminal statements unconditionally terminate evaluation of the current rule, non-terminal statements either only conditionally or never terminate evaluation of the current rule, in other words, they are passive from the ruleset evaluation perspective. There can be an arbitrary amount of non-terminal statements in a rule, but only a single terminal statement as the final statement.

VERDICT STATEMENT

The verdict statement alters control flow in the ruleset and issues policy decisions for packets.

```
{accept | drop | queue | continue | return}
{jump | goto} chain
```

accept and drop are absolute verdicts ? they terminate ruleset evaluation immediately.

accept   Terminate ruleset evaluation and
accept the packet. The packet
can still be dropped later by
another hook, for instance
accept in the forward hook still
allows to drop the packet later
in the postrouting hook, or
another forward base chain that

has a higher priority number and is evaluated afterwards in the processing pipeline.

drop Terminate ruleset evaluation and drop the packet. The drop occurs instantly, no further chains or hooks are evaluated. It is not possible to accept the packet in a later chain again, as those are not evaluated anymore for the packet.

queue Terminate ruleset evaluation and queue the packet to userspace. Userspace must provide a drop or accept verdict. In case of accept, processing resumes with the next base chain hook, not the rule following the queue verdict.

continue Continue ruleset evaluation with the next rule. This is the default behaviour in case a rule issues no verdict.

return Return from the current chain and continue evaluation at the next rule in the last chain. If issued in a base chain, it is equivalent to the base chain policy.

jump chain Continue evaluation at the first rule in chain. The current position in the ruleset is pushed to a call stack and evaluation will continue there when the new chain is entirely evaluated or a return verdict is issued. In case an absolute verdict is issued by a rule in the chain, ruleset evaluation terminates immediately and the specific action is taken.

goto chain   Similar to jump, but the current position is not pushed to the call stack, meaning that after the new chain evaluation will continue at the last chain instead of the one containing the goto statement.

Using verdict statements.

```
# process packets from eth0 and the internal network in from_lan
# chain, drop all packets from eth0 with different source addresses.
filter input iif eth0 ip saddr 192.168.0.0/24 jump from_lan
filter input iif eth0 drop
```

PAYLOAD STATEMENT

```
payload_expression set value
```

The payload statement alters packet content. It can be used for example to set ip DSCP (diffserv) header field or ipv6 flow labels.

route some packets instead of bridging.

```
# redirect tcp:http from 192.160.0.0/16 to local machine for routing instead of bridging
# assumes 00:11:22:33:44:55 is local MAC address.
bridge input meta iif eth0 ip saddr 192.168.0.0/16 tcp dport 80 meta pkttype set unicast ether daddr set 00:11:22:33:44:55
```

Set IPv4 DSCP header field.

```
ip forward ip dscp set 42
```

EXTENSION HEADER STATEMENT

```
extension_header_expression set value
```

The extension header statement alters packet content in variable-sized headers. This can currently be used to alter the TCP Maximum segment size of packets, similar to TCPMSS.

change tcp mss.

```
tcp flags syn tcp option maxseg size set 1360
# set a size based on route information:
tcp flags syn tcp option maxseg size set rt mtu
```

LOG STATEMENT

```
log [prefix quoted_string] [level syslog-level] [flags log-flags]
log group nflog_group [prefix quoted_string] [queue-threshold value] [snaplen size]
log level audit
```

The log statement enables logging of matching packets. When this statement is used from a rule, the Linux kernel will print some information on all matching packets, such as header fields, via the kernel log (where it can be read with dmesg(1) or read in the syslog).

In the second form of invocation (if nflog_group is specified), the Linux kernel will pass the packet to nfnetlink_log which will send the log through a netlink socket to the specified group. One userspace process may subscribe to the group to receive the logs, see man(8) ulogd for the Netfilter userspace log daemon and libnetfilter_log documentation for details in case you would like to develop a custom program to digest your logs.

In the third form of invocation (if level audit is specified), the Linux kernel writes a message into the audit buffer suitably formatted for reading with auditd. Therefore no further formatting options (such as prefix or flags) are allowed in this mode.

This is a non-terminating statement, so the rule evaluation continues after the packet is logged.

Table 59. log statement options

Keyword	Description	Type
prefix	Log message prefix	quoted string
level	Syslog level of logging	string: emerg, alert, crit, err, warn

		[default], notice, info, debug, audit
group	NFLOG group to send messages to	unsigned integer (16 bit)
snaplen	Length of packet payload to include in netlink message	unsigned integer (32 bit)
queue-threshold	Number of packets to queue inside the kernel before sending them to userspace	unsigned integer (32 bit)

Table 60. log-flags

Flag	Description
tcp sequence	Log TCP sequence numbers.
tcp options	Log options from the TCP packet header.
ip options	Log options from the IP/IPv6 packet header.

```
?            ?                              ?
?skuid       ? Log the userid of the process   ?
?            ? which generated the packet.     ?
????????????????????????????????????????????????
?            ?                              ?
?ether       ? Decode MAC addresses and       ?
?            ? protocol.                       ?
????????????????????????????????????????????????
?            ?                              ?
?all         ? Enable all log flags listed     ?
?            ? above.                          ?
????????????????????????????????????????????????
```

Using log statement.

```
# log the UID which generated the packet and ip options
ip filter output log flags skuid flags ip options
# log the tcp sequence numbers and tcp options from the TCP packet
ip filter output log flags tcp sequence,options
# enable all supported log flags
ip6 filter output log flags all
```

REJECT STATEMENT

```
reject [ with REJECT_WITH ]
REJECT_WITH := icmp icmp_code |
                 icmpv6 icmpv6_code |
                 icmpx icmpx_code |
                 tcp reset
```

A reject statement is used to send back an error packet in response to the matched packet otherwise it is equivalent to drop so it is a terminating statement, ending rule traversal. This statement is only valid in base chains using the input, forward or output hooks, and user-defined chains which are only called from those chains.

Table 61. different ICMP reject variants are meant for use in different table families

```
????????????????????????????????
?Variant ? Family ? Type       ?
????????????????????????????????
```

```
?        ?        ?              ?
?icmp    ? ip     ? icmp_code    ?
?????????????????????????????????
?        ?        ?              ?
?icmpv6  ? ip6    ? icmpv6_code  ?
?????????????????????????????????
?        ?        ?              ?
?icmpx   ? inet   ? icmpx_code   ?
?????????????????????????????????
```

For a description of the different types and a list of supported keywords refer to DATA TYPES section above. The common default reject value is port-unreachable.

Note that in bridge family, reject statement is only allowed in base chains which hook into input or prerouting.

COUNTER STATEMENT

A counter statement sets the hit count of packets along with the number of bytes.

```
counter packets number bytes number
counter { packets number | bytes number }
```

CONNTRACK STATEMENT

The conntrack statement can be used to set the conntrack mark and conntrack labels.

```
ct {mark | event | label | zone} set value
```

The ct statement sets meta data associated with a connection. The zone id has to be assigned before a conntrack lookup takes place, i.e. this has to be done in prerouting and possibly output (if locally generated packets need to be placed in a distinct zone), with a hook priority of raw (-300).

Unlike iptables, where the helper assignment happens in the raw table, the helper needs to be assigned after a conntrack entry has been found, i.e. it will not work when used with hook priorities equal or before -200.

Table 62. Conntrack statement types

```
???????????????????????????????????????????????????????????
?Keyword ? Description             ? Value                ?
???????????????????????????????????????????????????????????
?        ?                         ?                      ?
?event   ? conntrack event bits    ? bitmask, integer (32 ?
?        ?                           ? bit)                 ?
???????????????????????????????????????????????????????????
?        ?                           ?                      ?
?helper  ? name of ct helper object ? quoted string        ?
?        ? to assign to the         ?                      ?
?        ? connection               ?                      ?
???????????????????????????????????????????????????????????
?        ?                           ?                      ?
?mark    ? Connection tracking mark ? mark                 ?
???????????????????????????????????????????????????????????
?        ?                           ?                      ?
?label   ? Connection tracking      ? label                ?
?        ? label                    ?                      ?
???????????????????????????????????????????????????????????
?        ?                           ?                      ?
?zone    ? conntrack zone           ? integer (16 bit)     ?
???????????????????????????????????????????????????????????
```

save packet nfmark in conntrack.

```
ct mark set meta mark
```

set zone mapped via interface.

```
table inet raw {
  chain prerouting {
      type filter hook prerouting priority raw;
      ct zone set iif map { "eth1" : 1, "veth1" : 2 }
  }
  chain output {
      type filter hook output priority raw;
      ct zone set oif map { "eth1" : 1, "veth1" : 2 }
  }
}
```

restrict events reported by ctnetlink.

```
ct event set new,related,destroy
```

NOTRACK STATEMENT

The notrack statement allows to disable connection tracking for certain packets.

```
notrack
```

Note that for this statement to be effective, it has to be applied to packets before a conntrack lookup happens. Therefore, it needs to sit in a chain with either prerouting or output hook and a hook priority of -300 (raw) or less.

See SYNPROXY STATEMENT for an example usage.

META STATEMENT

A meta statement sets the value of a meta expression. The existing meta fields are: priority, mark, pkttype, nftrace.

```
meta {mark | priority | pkttype | nftrace} set value
```

A meta statement sets meta data associated with a packet.

Table 63. Meta statement types

Keyword	Description	Value
priority	TC packet priority	tc_handle
mark	Packet mark	mark
pkttype	packet type	pkt_type
nftrace	ruleset packet tracing on/off. Use monitor trace command to watch traces	0, 1

LIMIT STATEMENT

```
limit rate [over] packet_number / TIME_UNIT [burst packet_number packets]
limit rate [over] byte_number BYTE_UNIT / TIME_UNIT [burst byte_number BYTE_UNIT]
```

```
TIME_UNIT := second | minute | hour | day

BYTE_UNIT := bytes | kbytes | mbytes
```

A limit statement matches at a limited rate using a token bucket filter. A rule using this statement will match until this limit is reached. It can be used in combination with the log statement to give limited logging. The optional over keyword makes it match over the specified rate. Default burst is 5. if you specify burst, it must be non-zero value.

Table 64. limit statement values

Value	Description	Type
packet_number	Number of packets	unsigned integer (32 bit)
byte_number	Number of bytes	unsigned integer (32 bit)

NAT STATEMENTS

```
snat [[ip | ip6] to] ADDR_SPEC [:PORT_SPEC] [FLAGS]

dnat [[ip | ip6] to] ADDR_SPEC [:PORT_SPEC] [FLAGS]

masquerade [to :PORT_SPEC] [FLAGS]

redirect [to :PORT_SPEC] [FLAGS]

ADDR_SPEC := address | address - address

PORT_SPEC := port | port - port

FLAGS  := FLAG [, FLAGS]

FLAG  := persistent | random | fully-random
```

The nat statements are only valid from nat chain types.

The snat and masquerade statements specify that the source address of the packet should be modified. While snat is only valid in the postrouting and input chains, masquerade makes sense only in postrouting. The dnat and redirect statements are only valid in the prerouting and output chains, they specify that the destination address of the packet should be modified. You can use non-base chains which are called from base chains of nat chain type too. All future packets in this connection will also be mangled, and rules should cease being examined.

The masquerade statement is a special form of snat which always uses the outgoing interface?s IP address to translate to. It is particularly useful on gateways with dynamic (public) IP addresses.

The redirect statement is a special form of dnat which always translates the destination address to the local host?s one. It comes in handy if one only wants to alter the destination port of incoming traffic on different interfaces.

When used in the inet family (available with kernel 5.2), the dnat and snat statements require the use of the ip and ip6 keyword in case an address is provided, see the examples below.

Before kernel 4.18 nat statements require both prerouting and postrouting base chains to be present since otherwise packets on the return path won?t be seen by netfilter and therefore no reverse translation will take place.

Table 65. NAT statement values

Expression	Description	Type
address	Specifies that the source/destination address of the packet should be modified. You may specify a mapping to relate a list of tuples composed of arbitrary expression key with address value.	ipv4_addr, ipv6_addr, e.g. abcd::1234, or you can use a mapping, e.g. meta mark map { 10 : 192.168.1.2, 20 : 192.168.1.3 }
port	Specifies that the source/destination address of the packet	port number (16 bit)

?   ? should be modified.   ?   ?

???

Table 66. NAT statement flags

???

?Flag   ? Description   ?

???

?   ?   ?

?persistent   ? Gives a client the same   ?

?   ? source-/destination-address for ?

?   ? each connection.   ?

???

?   ?   ?

?random   ? In kernel 5.0 and newer this is ?

?   ? the same as fully-random. In   ?

?   ? earlier kernels the port mapping ?

?   ? will be randomized using a   ?

?   ? seeded MD5 hash mix using source ?

?   ? and destination address and   ?

?   ? destination port.   ?

???

?   ?   ?

?fully-random ? If used then port mapping is   ?

?   ? generated based on a 32-bit   ?

?   ? pseudo-random algorithm.   ?

???

Using NAT statements.

```
# create a suitable table/chain setup for all further examples
add table nat
add chain nat prerouting { type nat hook prerouting priority dstnat; }
add chain nat postrouting { type nat hook postrouting priority srcnat; }
# translate source addresses of all packets leaving via eth0 to address 1.2.3.4
add rule nat postrouting oif eth0 snat to 1.2.3.4
# redirect all traffic entering via eth0 to destination address 192.168.1.120
```

```
add rule nat prerouting iif eth0 dnat to 192.168.1.120

# translate source addresses of all packets leaving via eth0 to whatever

# locally generated packets would use as source to reach the same destination

add rule nat postrouting oif eth0 masquerade

# redirect incoming TCP traffic for port 22 to port 2222

add rule nat prerouting tcp dport 22 redirect to :2222

# inet family:

# handle ip dnat:

add rule inet nat prerouting dnat ip to 10.0.2.99

# handle ip6 dnat:

add rule inet nat prerouting dnat ip6 to fe80::dead

# this masquerades both ipv4 and ipv6:

add rule inet nat postrouting meta oif ppp0 masquerade
```

TPROXY STATEMENT

Tproxy redirects the packet to a local socket without changing the packet header in any way. If any of the arguments is missing the data of the incoming packet is used as parameter. Tproxy matching requires another rule that ensures the presence of transport protocol header is specified.

```
tproxy to address:port

tproxy to {address | :port}
```

This syntax can be used in ip/ip6 tables where network layer protocol is obvious. Either IP address or port can be specified, but at least one of them is necessary.

```
tproxy {ip | ip6} to address[:port]

tproxy to :port
```

This syntax can be used in inet tables. The ip/ip6 parameter defines the family the rule will match. The address parameter must be of this family. When only port is defined, the address family should not be specified. In this case the rule will match for both families.

Table 67. tproxy attributes

```
????????????????????????????????????????????

?Name    ? Description                    ?

????????????????????????????????????????????

?        ?                                ?
```

?address ? IP address the listening socket ?

?   ? with IP_TRANSPARENT option is   ?

?   ? bound to.   ?

??

?   ?   ?

?port   ? Port the listening socket with ?

?   ? IP_TRANSPARENT option is bound ?

?   ? to.   ?

??

Example ruleset for tproxy statement.

```
    table ip x {
        chain y {
            type filter hook prerouting priority mangle; policy accept;
            tcp dport ntp tproxy to 1.1.1.1
            udp dport ssh tproxy to :2222
        }
    }
    table ip6 x {
        chain y {
           type filter hook prerouting priority mangle; policy accept;
           tcp dport ntp tproxy to [dead::beef]
           udp dport ssh tproxy to :2222
        }
    }
    table inet x {
        chain y {
            type filter hook prerouting priority mangle; policy accept;
            tcp dport 321 tproxy to :ssh
            tcp dport 99 tproxy ip to 1.1.1.1:999
            udp dport 155 tproxy ip6 to [dead::beef]:smux
        }
    }
```

SYNPROXY STATEMENT

This statement will process TCP three-way-handshake parallel in netfilter context to protect either local or backend system. This statement requires connection tracking because sequence numbers need to be translated.

```
    synproxy [mss mss_value] [wscale wscale_value] [SYNPROXY_FLAGS]
```

Table 68. synproxy statement attributes

Name	Description
mss	Maximum segment size announced to clients. This must match the backend.
wscale	Window scale announced to clients. This must match the backend.

Table 69. synproxy statement flags

Flag	Description
sack-perm	Pass client selective acknowledgement option to backend (will be disabled if not present).
timestamp	Pass client timestamp option to backend (will be disabled if not present, also needed for selective acknowledgement and

?   ? window scaling).   ?

???

Example ruleset for synproxy statement.

Determine tcp options used by backend, from an external system

```
       tcpdump -pni eth0 -c 1 'tcp[tcpflags] == (tcp-syn|tcp-ack)'
           port 80 &
       telnet 192.0.2.42 80
       18:57:24.693307 IP 192.0.2.42.80 > 192.0.2.43.48757:
           Flags [S.], seq 360414582, ack 788841994, win 14480,
           options [mss 1460,sackOK,
           TS val 1409056151 ecr 9690221,
           nop,wscale 9],
           length 0
```

Switch tcp_loose mode off, so conntrack will mark out-of-flow packets as state INVALID.

```
       echo 0 > /proc/sys/net/netfilter/nf_conntrack_tcp_loose
```

Make SYN packets untracked.

```
    table ip x {
        chain y {
                type filter hook prerouting priority raw; policy accept;
                tcp flags syn notrack
        }
    }
```

Catch UNTRACKED (SYN packets) and INVALID (3WHS ACK packets) states and send them to SYNPROXY. This rule will respond to SYN packets with SYN+ACK syncookies, create ESTABLISHED for valid client response (3WHS ACK packets) and drop incorrect cookies. Flags combinations not expected during 3WHS will not match and continue (e.g. SYN+FIN, SYN+ACK). Finally, drop invalid packets, this will be out-of-flow packets that were not matched by SYNPROXY.

```
    table ip x {
        chain z {
                type filter hook input priority filter; policy accept;
                ct state invalid, untracked synproxy mss 1460 wscale 9 timestamp sack-perm
                ct state invalid drop
        }
    }
```

FLOW STATEMENT

A flow statement allows us to select what flows you want to accelerate forwarding through layer 3 network stack bypass. You have to specify the flowtable name where you want to offload this flow.

```
flow add @flowtable
```

QUEUE STATEMENT

This statement passes the packet to userspace using the nfnetlink_queue handler. The packet is put into the queue identified by its 16-bit queue number. Userspace can inspect and modify the packet if desired. Userspace must then drop or re-inject the packet into the kernel. See libnetfilter_queue documentation for details.

```
queue [flags QUEUE_FLAGS] [to queue_number]
queue [flags QUEUE_FLAGS] [to queue_number_from - queue_number_to]
queue [flags QUEUE_FLAGS] [to QUEUE_EXPRESSION ]
QUEUE_FLAGS := QUEUE_FLAG [, QUEUE_FLAGS]
QUEUE_FLAG  := bypass | fanout
QUEUE_EXPRESSION := numgen | hash | symhash | MAP STATEMENT
```

QUEUE_EXPRESSION can be used to compute a queue number at run-time with the hash or numgen expressions. It also allows to use the map statement to assign fixed queue numbers based on external inputs such as the source ip address or interface names.

Table 70. queue statement values

Value	Description	Type
queue_number	Sets queue number, default is 0.	unsigned integer (16 bit)
queue_number_from	Sets initial queue in the range, if fanout is used.	unsigned integer (16 bit)

queue_number_to	Sets closing queue in the range, if fanout is used.	unsigned integer (16 bit)

Table 71. queue statement flags

Flag	Description
bypass	Let packets go through if userspace application cannot back off. Before using this flag, read libnetfilter_queue documentation for performance tuning recommendations.
fanout	Distribute packets between several queues.

DUP STATEMENT

The dup statement is used to duplicate a packet and send the copy to a different destination.

```
dup to device
dup to address device device
```

Table 72. Dup statement values

Expression	Description	Type
address	Specifies that the copy	ipv4_addr, ipv6_addr,

```
?           ? of the packet should be ? e.g. abcd::1234, or you  ?
?           ? sent to a new gateway.  ? can use a mapping, e.g.  ?
?           ?                         ? ip saddr map {           ?
?           ?                         ? 192.168.1.2 : 10.1.1.1 } ?
?????????????????????????????????????????????????????????????????????
?           ?                         ?                          ?
?device     ? Specifies that the copy ? string                   ?
?           ? should be transmitted   ?                          ?
?           ? via device.             ?                          ?
?????????????????????????????????????????????????????????????????????
```

Using the dup statement.

```
# send to machine with ip address 10.2.3.4 on eth0
ip filter forward dup to 10.2.3.4 device "eth0"
# copy raw frame to another interface
netdev ingress dup to "eth0"
dup to "eth0"
# combine with map dst addr to gateways
dup to ip daddr map { 192.168.7.1 : "eth0", 192.168.7.2 : "eth1" }
```

FWD STATEMENT

The fwd statement is used to redirect a raw packet to another interface. It is only available in the netdev family ingress and egress hooks. It is similar to the dup statement except that no copy is made.

fwd to device

SET STATEMENT

The set statement is used to dynamically add or update elements in a set from the packet path. The set setname must already exist in the given table and must have been created with one or both of the dynamic and the timeout flags. The dynamic flag is required if the set statement expression includes a stateful object. The timeout flag is implied if the set is created with a timeout, and is required if the set statement updates elements, rather than adding them. Furthermore, these sets should specify both a maximum set size (to prevent memory exhaustion), and their elements should have a timeout (so their number will not grow indefinitely) either from the set definition or from the statement that adds or updates them. The set statement can be used to e.g. create dynamic blacklists.

```
{add | update} @setname { expression [timeout timeout] [comment string] }
```

Example for simple blacklist.

```
# declare a set, bound to table "filter", in family "ip".
# Timeout and size are mandatory because we will add elements from packet path.
# Entries will timeout after one minute, after which they might be
# re-added if limit condition persists.
nft add set ip filter blackhole \
    "{ type ipv4_addr; flags dynamic; timeout 1m; size 65536; }"
# declare a set to store the limit per saddr.
# This must be separate from blackhole since the timeout is different
nft add set ip filter flood \
    "{ type ipv4_addr; flags dynamic; timeout 10s; size 128000; }"
# whitelist internal interface.
nft add rule ip filter input meta iifname "internal" accept
# drop packets coming from blacklisted ip addresses.
nft add rule ip filter input ip saddr @blackhole counter drop
# add source ip addresses to the blacklist if more than 10 tcp connection
# requests occurred per second and ip address.
nft add rule ip filter input tcp flags syn tcp dport ssh \
    add @flood { ip saddr limit rate over 10/second } \
    add @blackhole { ip saddr } \
    drop
# inspect state of the sets.
nft list set ip filter flood
nft list set ip filter blackhole
# manually add two addresses to the blackhole.
nft add element filter blackhole { 10.2.3.4, 10.23.1.42 }
```

MAP STATEMENT

The map statement is used to lookup data based on some specific input key.

```
expression map { MAP_ELEMENTS }
MAP_ELEMENTS := MAP_ELEMENT [, MAP_ELEMENTS]
MAP_ELEMENT  := key : value
```

The key is a value returned by expression.

Using the map statement.

```
# select DNAT target based on TCP dport:
# connections to port 80 are redirected to 192.168.1.100,
# connections to port 8888 are redirected to 192.168.1.101
nft add rule ip nat prerouting dnat tcp dport map { 80 : 192.168.1.100, 8888 : 192.168.1.101 }
# source address based SNAT:
# packets from net 192.168.1.0/24 will appear as originating from 10.0.0.1,
# packets from net 192.168.2.0/24 will appear as originating from 10.0.0.2
nft add rule ip nat postrouting snat to ip saddr map { 192.168.1.0/24 : 10.0.0.1, 192.168.2.0/24 : 10.0.0.2 }
```

VMAP STATEMENT

The verdict map (vmap) statement works analogous to the map statement, but contains verdicts as values.

```
expression vmap { VMAP_ELEMENTS }
VMAP_ELEMENTS := VMAP_ELEMENT [, VMAP_ELEMENTS]
VMAP_ELEMENT  := key : verdict
```

Using the vmap statement.

```
# jump to different chains depending on layer 4 protocol type:
nft add rule ip filter input ip protocol vmap { tcp : jump tcp-chain, udp : jump udp-chain , icmp : jump icmp-chain }
```

ADDITIONAL COMMANDS

These are some additional commands included in nft.

MONITOR

The monitor command allows you to listen to Netlink events produced by the nf_tables subsystem. These are either related to creation and deletion of objects or to packets for which meta nftrace was enabled. When they occur, nft will print to stdout the monitored events in either JSON or native nft format.

```
monitor [new | destroy] MONITOR_OBJECT
monitor trace
MONITOR_OBJECT := tables | chains | sets | rules | elements | ruleset
```

To filter events related to a concrete object, use one of the keywords in MONITOR_OBJECT.

To filter events related to a concrete action, use keyword new or destroy.

The second form of invocation takes no further options and exclusively prints events generated for packets with nftrace enabled.

Hit ^C to finish the monitor operation.

Listen to all events, report in native nft format.

```
% nft monitor
```

Listen to deleted rules, report in JSON format.

```
% nft -j monitor destroy rules
```

Listen to both new and destroyed chains, in native nft format.

```
% nft monitor chains
```

Listen to ruleset events such as table, chain, rule, set, counters and quotas, in native nft format.

```
% nft monitor ruleset
```

Trace incoming packets from host 10.0.0.1.

```
% nft add rule filter input ip saddr 10.0.0.1 meta nftrace set 1
% nft monitor trace
```

ERROR REPORTING

When an error is detected, nft shows the line(s) containing the error, the position of the erroneous parts in the input stream and marks up the erroneous parts using carets (^). If the error results from the combination of two expressions or statements, the part imposing the constraints which are violated is marked using tildes (~).

For errors returned by the kernel, nft cannot detect which parts of the input caused the error and the entire command is marked.

Error caused by single incorrect expression.

```
<cmdline>:1:19-22: Error: Interface does not exist
filter output oif eth0
                  ^^^^
```

Error caused by invalid combination of two expressions.

```
<cmdline>:1:28-36: Error: Right hand side of relational expression (==) must be constant
filter output tcp dport == tcp dport
                        ~~ ^^^^^^^^^
```

Error returned by the kernel.

```
<cmdline>:0:0-23: Error: Could not process rule: Operation not permitted
filter output oif wlan0
^^^^^^^^^^^^^^^^^^^^^^^
```

EXIT STATUS

On success, nft exits with a status of 0. Unspecified errors cause it to exit with a status of 1, memory allocation errors with a status of 2, unable to open Netlink socket with 3.

SEE ALSO

libnftables(3), libnftables-json(5), iptables(8), ip6tables(8), arptables(8), ebtables(8), ip(8), tc(8)

There is an official wiki at: https://wiki.nftables.org

AUTHORS

nftables was written by Patrick McHardy and Pablo Neira Ayuso, among many other contributors from the Netfilter community.

COPYRIGHT