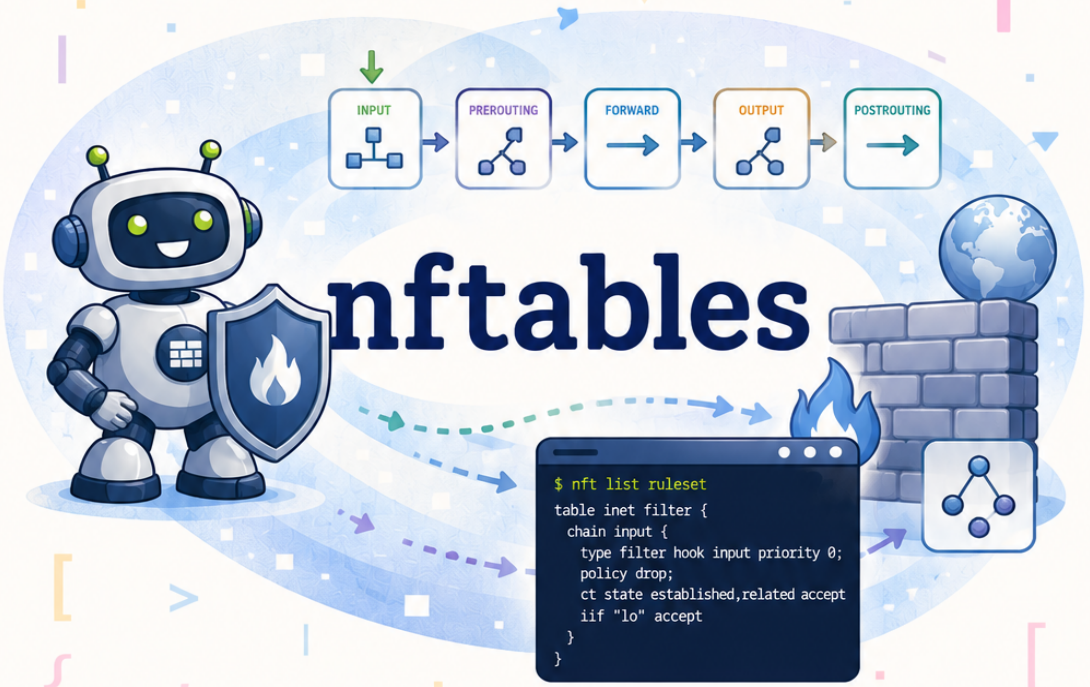
URADICAL

POCKET REFERENCE



nftables Pocket Reference

Alan Bradley

nftables Pocket Reference

The Modern Linux Packet Filtering Framework

Alan Bradley

uradical.io

nftables Pocket Reference

Alan Bradley

Table of Contents

Chapter 1: Introduction & Architecture

nftables is the unified packet classification framework for Linux, introduced in kernel 3.13 and replacing the legacy `iptables` / `ip6tables` / `arptables` / `ebtables` stack. It provides packet filtering, network address translation, and packet mangling in one coherent subsystem with a single userspace tool: `nft`.

The Data Model

nftables is a strict hierarchy. Tables contain chains, chains contain rules, and rules reference sets and maps.

Layer	Description
Address family	Selects the protocol stack: ip, ip6, inet, arp, bridge, netdev
Table	A named container for chains, sets, maps, and stateful objects; belongs to one address family
Chain	An ordered list of rules. A base chain is hooked into netfilter. A regular chain is reachable only via jump or goto
Rule	One or more match expressions followed by one or more statements
Set / Map	Named or anonymous collections of values used in matches or verdict dispatch
Stateful object	Named counter, quota, limit, ct helper, ct timeout, ct expectation
Flowtable	A software or hardware fastpath that offloads established connections

Address Families

Family	Scope	Replaces
ip	IPv4 traffic	iptables
ip6	IPv6 traffic	ip6tables
inet	IPv4 + IPv6 (dual-stack)	—
arp	ARP packets	arptables
bridge	Bridged Ethernet	ebtables
netdev	Per-interface ingress/egress	tc (partial)

The `inet` family registers hooks in both the `ip` and `ip6` subsystems, letting one ruleset handle dual-stack traffic.

Netfilter Hook Points

Hook	Fires when…
prerouting	Packet arrives at the host, before the routing decision
input	Packet is destined for a local process
forward	Packet is being forwarded to another host
output	Packet generated by a local process, before routing
postrouting	Packet is leaving, after the routing decision
ingress	netdev only — before prerouting, before even tc (kernel 4.2+)

egress	netdev only — after postrouting, after tc (kernel 5.16+)

The `bridge` family uses the same hook names but operates on bridged traffic before it enters the routing path.

Kernel Version Reference

Feature	Minimum kernel
nftables core	3.13
Verdict maps (vmap)	3.16
Named counters, quotas, limits	3.17
Set concatenations	4.15
fib expressions	4.10 / 4.14*
numgen, jhash, symhash	4.1
Flowtables (software fastpath)	4.16
JSON input/output (-j)	4.16
tproxy statement	4.18
socket expression	4.18
synproxy statement	4.19
osf expression	5.2
ct count expression	5.3
ct timeout / ct expectation objects	5.4
Rule comments (comment "…")	5.10
Flowtable hardware offload	5.13
netdev egress hook	5.16

*`fib saddr . iif` requires kernel 4.14 for correct behaviour.

Chapter 2: nft Command Reference

Global Flags

Flag	Meaning
-n / --numeric	Print IP addresses and ports numerically
-N / --reversedns	Reverse DNS lookup on IP addresses in output
-a / --handle	Show object handles (required for deletion by handle)
-e / --echo	Echo each command as it is applied
-j / --json	JSON input/output mode
-f FILE	Read commands from FILE and apply atomically
-i / --interactive	Start an interactive REPL
-c / --check	Syntax-check only — do not apply changes
-I DIR	Add DIR to the include path for include directives
--debug LEVEL	scanner, parser, eval, netlink, mnl, proto-ctx, segtree, all

Command Verbs

Verb	Applies to
add	table, chain, rule, set, map, element, flowtable, counter, quota, limit, ct helper, ct timeout, ct expectation

create	table, chain, set, map, flowtable — fails if the object already exists
insert	rule — insert at chain head or before a handle
replace	rule — replace at a given handle
delete	table, chain, rule, set, map, element, flowtable, counter, quota, limit
rename	chain
flush	table, chain, set, map, ruleset
list	tables, table, chain, ruleset, sets, maps, flowtables, counters, quotas, limits, ct helpers
reset	counters, quotas, rules — resets byte/packet counts without removing objects
monitor	Live event stream of ruleset changes
describe	Describe an expression or datatype

> **Note:** `nft export` was removed in nftables 0.9.1. Use `nft list ruleset` or `nft -j list ruleset` instead.

Common Invocations

```
nft list ruleset                          # list everything
nft -a list ruleset                       # with handles
nft -j list ruleset                       # as JSON
nft -f /etc/nftables.conf                 # apply atomically
nft -c -f /etc/nftables.conf              # syntax-check only
nft list ruleset > /etc/nftables.conf     # save to file
nft monitor                               # watch live changes
nft flush ruleset                         # clear everything
(destructive)
nft -i                                    # interactive REPL
```

Chapter 3: Ruleset Files, Variables & Includes

`nft -f` applies an entire file in a single kernel transaction. The file format extends inline syntax with variables, includes, and the `flush` directive.

Variables

```
define WAN     = eth0
define LAN     = eth1
define LAN_NET = 192.168.1.0/24
define ADMIN   = { 10.0.0.10, 192.168.1.1 }

table inet filter {
    chain forward {
        type filter hook forward priority filter; policy drop;
        iifname $LAN ip saddr $LAN_NET oifname $WAN accept
    }
}
```

`redefine` overwrites an existing variable without error:

```
define PORT = 80
redefine PORT = 443
```

Includes

```
#!/usr/sbin/nft -f
flush ruleset
include "/etc/nftables.d/*.conf"
```

Include paths are searched relative to the `-I` flag directories or the directory of the including file.

flush Is Atomic When Inside a File

`flush ruleset` at the top of a file loaded with `nft -f` is part of the transaction. If the file has a syntax error, the flush is rolled back and existing rules are preserved.

Running `nft flush ruleset` on the command line is immediate and irreversible. Never:

```
nft flush ruleset          # immediate — no rollback
nft -f /etc/nftables.conf  # if this fails, firewall is now
empty
```

Always:

```
nft -f /etc/nftables.conf  # file starts with: flush ruleset
```

Chapter 4: Tables

Table Operations

```
nft add table inet filter           # add (no-op if exists)
nft create table inet filter        # create (fails if exists)
nft list tables                     # list all
nft list table inet filter          # list table contents
nft flush table inet filter         # flush rules, keep
chains/sets
nft delete table inet filter        # delete table and all
contents
```

Table Flags

Flag	Effect
dormant	Base chains are not registered — traffic bypasses the table

```
nft add table inet filter { flags dormant; }  # disable
nft add table inet filter                     # re-enable
```

Chapter 5: Chains

A **base chain** is attached to a hook. A **regular chain** has no hook and is reachable only via `jump` or `goto`.

Chain Operations

```
nft add chain inet filter input \
    { type filter hook input priority filter; policy drop; }
nft add chain inet filter tcp-services              # regular
chain
nft rename chain inet filter old-name new-name
nft flush chain inet filter input                   # flush
rules, keep chain
nft delete chain inet filter input                  # delete
(must be empty)
```

Chain Types

Type	Families	Valid Hooks	Purpose
filter	all	all	Packet filtering
nat	ip, ip6, inet	prerouting, output, postrouting	NAT — once per connection
route	ip, ip6	output	Reroute if IP header modified

Chain Priorities

Name	Value	Use
raw	.300	Bypass conntrack
mangle	.150	Header mangling

dstnat	.100	DNAT at prerouting
filter	0	Standard filtering
security	50	SELinux marks
srcnat	100	SNAT at postrouting

The `bridge` family uses different numeric values — always use symbolic names. Expressions like `filter + 10` are valid.

Chain Policies

Only base chains have a policy. It is the default verdict for unmatched packets.

Policy	Effect
accept	Pass to the next stage (default if unspecified)
drop	Silently discard

Always declare `policy drop` explicitly for input chains on servers — the implicit `accept` default is not safe.

Chapter 6: Rules

Rule Operations

```
nft add rule inet filter input tcp dport 22 accept          #
append
nft insert rule inet filter input tcp dport 22 accept       #
prepend
nft insert rule inet filter input handle 10 tcp dport 80
accept  # before handle
nft add    rule inet filter input handle 10 tcp dport 443
accept # after handle
nft replace rule inet filter input handle 10 tcp dport 8080
accept
nft delete  rule inet filter input handle 10
nft -a list chain inet filter input                         #
show handles
```

Rule Syntax

```
# Implicit AND
ip saddr 192.168.1.0/24 tcp dport { 80, 443 } accept

# Negation
ip saddr != { 10.0.0.0/8, 172.16.0.0/12 } drop

# Anonymous set with ranges
tcp dport { 1-1023, 8080 } drop

# Comment (kernel 5.10+)
tcp dport 22 accept comment "allow SSH"

# Per-rule counter
ip protocol icmp counter accept
```

Verdict Statements

Verdict	Behaviour
accept	Pass the packet; stop evaluating this chain
drop	Discard; stop evaluating
return	Stop this chain; return to caller, or apply base chain policy
jump <chain>	Evaluate <chain>; return and continue here
goto <chain>	Evaluate <chain>; do not return
continue	Continue to next rule (implicit default)

> `goto` **subtlety:** If the target chain has no matching rule, the packet gets the **base chain's policy** — not a continue. This surprises people expecting iptables semantics.

Chapter 7: Match Expressions

Operators

Operator	Meaning
==	Equal (may be omitted)
!=	Not equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
&	Binary AND (bitmask)
|	Binary OR
^	Binary XOR
<<	Left shift
>>	Right shift

Meta Expressions

Key	Type	Description
meta iifname	string	Input interface name — resolved at match time
meta oifname	string	Output interface name — resolved at match time
meta iif	integer	Input interface index — resolved at rule load time

meta oif	integer	Output interface index — resolved at rule load time
meta iiftype	integer	Input interface hardware type (ARPHRD_*)
meta oiftype	integer	Output interface hardware type
meta skuid	integer	UID of originating process
meta skgid	integer	GID of originating process
meta mark	integer	Packet mark (nfmark); writable
meta priority	integer	TC classid; writable
meta length	integer	Total packet length in bytes
meta protocol	integer	EtherType
meta nfproto	integer	Address family at hook (2=IPv4, 10=IPv6)
meta l4proto	integer	Layer-4 protocol number
meta pkttype	string	host, broadcast, multicast, other
meta cpu	integer	CPU ID processing the packet
meta random	integer	Pseudo-random 32-bit value, re-evaluated per packet
meta secmark	integer	SELinux security mark; writable
meta ibrname	string	Bridge input port name
meta obrname	string	Bridge output port name

`iif` **vs** `iifname`**:** `iif` stores the interface *index* as an integer at rule-load time. If the interface is recreated (VPN reconnect, container restart, bridge teardown), it gets a new index and the rule silently stops matching. `iif` also causes `nft -f` to fail with "Interface does not exist" if the interface has not yet been created — a common boot-ordering issue with bridges, WireGuard tunnels, and VLANs. Default to `iifname` / `oifname` in all production rulesets.

```
iifname "eth0" accept    # safe — survives recreation
iif     "eth0" accept    # fast but fragile
```

IP / IPv6 Header Fields

Field	Family	Writable	Description
ip version	ip		IP version (4)
ip hdrlength	ip		Header length in 32-bit words
ip dscp	ip	.	DSCP bits
ip ecn	ip	.	ECN bits
ip length	ip		Total length
ip id	ip		Fragment identification
ip frag-off	ip		Fragment offset
ip ttl	ip	.	Time to live
ip protocol	ip		Layer-4 protocol number
ip checksum	ip		Header checksum
ip saddr	ip		Source IPv4 address

ip daddr	ip		Destination IPv4 address
ip6 version	ip6		IP version (6)
ip6 dscp	ip6	.	Traffic class DSCP bits
ip6 ecn	ip6	.	Traffic class ECN bits
ip6 flowlabel	ip6		Flow label
ip6 length	ip6		Payload length
ip6 nexthdr	ip6		Next header protocol
ip6 hoplimit	ip6	.	Hop limit
ip6 saddr	ip6		Source IPv6 address
ip6 daddr	ip6		Destination IPv6 address

```
ip saddr 192.168.1.0/24 accept
ip daddr != { 10.0.0.1, 10.0.0.2 } drop
ip ttl < 5 drop

ip6 saddr ::1 accept
ip6 nexthdr tcp accept

# In inet family, write separate rules per L3 family
ip  saddr 192.168.0.0/16 accept
ip6 saddr fc00::/7       accept
```

TCP

Field	Description
tcp sport	Source port

tcp dport	Destination port
tcp sequence	Sequence number
tcp ackseq	Acknowledgement number
tcp doff	Data offset
tcp reserved	Reserved bits
tcp flags	fin syn rst psh ack urg ecn cwr
tcp window	Window size
tcp checksum	Checksum
tcp urgptr	Urgent pointer
tcp option	TCP option access

```
tcp dport 22 accept
tcp dport { 80, 443, 8080 } accept
tcp sport 1024-65535 accept

tcp flags & (syn | ack) == syn              # SYN without ACK
tcp flags & (fin | syn | rst | ack) == syn  # SYN only — SYN
scan detection
tcp flags & (fin | syn) == (fin | syn)      # Illegal — drop

tcp option maxseg size 1460 accept
tcp option sack-perm exists accept
```

UDP, UDP-Lite, SCTP, DCCP

Field	Protocol	Description
udp sport	udp	Source port
udp dport	udp	Destination port
udp length	udp	Header + data length
udp checksum	udp	Checksum

udplite sport	udplite	Source port
udplite dport	udplite	Destination port
udplite checksum-coverage	udplite	Checksum coverage
sctp sport	sctp	Source port
sctp dport	sctp	Destination port
sctp vtag	sctp	Verification tag
sctp checksum	sctp	Checksum
dccp sport	dccp	Source port
dccp dport	dccp	Destination port
dccp type	dccp	Packet type

ICMP & ICMPv6

Field	Protocol	Description
icmp type	icmp	Message type
icmp code	icmp	Message code
icmp checksum	icmp	Checksum
icmp id	icmp	Echo identifier
icmp sequence	icmp	Echo sequence
icmpv6 type	icmpv6	Message type
icmpv6 code	icmpv6	Message code
icmpv6 checksum	icmpv6	Checksum
icmpv6 id	icmpv6	Echo identifier
icmpv6 sequence	icmpv6	Echo sequence
icmpv6 mtu	icmpv6	MTU in Packet Too Big

icmpv6 max-delay	icmpv6	MLD max response delay

Named ICMPv4 types: `echo-reply`, `destination-unreachable`, `source-quench`, `redirect`, `echo-request`, `time-exceeded`, `parameter-problem`, `timestamp-request`, `timestamp-reply`

Named ICMPv6 types: `destination-unreachable`, `packet-too-big`, `time-exceeded`, `parameter-problem`, `echo-request`, `echo-reply`, `mld-listener-query`, `mld-listener-report`, `mld-listener-done`, `nd-router-solicit`, `nd-router-advert`, `nd-neighbor-solicit`, `nd-neighbor-advert`, `nd-redirect`, `mld2-listener-report`

```
icmp type echo-request accept

icmpv6 type {
    nd-neighbor-solicit, nd-neighbor-advert,
    nd-router-solicit,   nd-router-advert,
    mld-listener-query,  mld-listener-report
} accept
icmpv6 type echo-request limit rate 10/second accept

# inet family: icmpx picks icmp or icmpv6 automatically
reject with icmpx type port-unreachable
```

Connection Tracking (ct)

Conntrack is loaded automatically when a `nat`-type chain or a `ct state` rule is present.

Key	Description
ct state	new, established, related, invalid, untracked
ct direction	original or reply
ct status	expected, seen-reply, assured, confirmed, snat, dnat, dying

ct mark	32-bit conntrack mark; writable
ct label	128-bit conntrack label; writable
ct expiration	Time until the conntrack entry expires
ct helper	Name of the conntrack helper
ct count	Count of connections matching the same tuple (kernel 5.3+)
ct id	Conntrack entry ID
ct l3proto	Layer-3 protocol number
ct saddr	Source address in original direction
ct daddr	Destination address in original direction
ct proto-src	Source port in original direction
ct proto-dst	Destination port in original direction

```
ct state { established, related } accept
ct state invalid                  drop
ct state new tcp dport 22         accept

meta mark set ct mark       # restore conntrack mark
ct mark set meta mark       # save packet mark to conntrack

tcp dport 80 ct state new ct count over 20 drop  # per-IP
limit
```

> `ct state invalid` **and asymmetric routing:** `ct state invalid drop` is correct for single-path stateful firewalls. In asymmetric routing (ECMP, BGP anycast, LVS direct routing, multi-path), return traffic may arrive on a node that never saw the SYN — it is classified `invalid` and silently dropped. Before deploying this rule on any multi-path infrastructure, add logging first: `ct state invalid log prefix "ct-invalid: " drop` and confirm what you see.

FIB Expressions

FIB expressions perform route lookups at evaluation time. They are the native nftables approach to reverse path filtering and source-based routing — absent from most reference material but used in production on every properly configured router.

FIB result types:

Result	Description
oif	Output interface index for this lookup
oifname	Output interface name for this lookup
type	Route type: unicast, local, broadcast, multicast, blackhole, unreachable, prohibit

FIB flags (lookup keys):

Flag	Description
saddr	Use source address
daddr	Use destination address
mark	Include packet mark
iif	Include input interface
oif	Include output interface

```
# Reverse path filter: drop packets with no route back
# out the interface they arrived on (rp_filter equivalent)
fib saddr . iif oif missing drop

# Drop spoofed packets claiming to be from loopback
fib saddr type != { local, broadcast, multicast } \
    iifname "lo" drop
```

```
# Accept only packets destined for local addresses
fib daddr . iif type local accept
```

Requires kernel 4.14+ for `fib saddr . iif` to work correctly.

IPv6 Extension Headers (exthdr)

```
# Drop packets with routing extension headers
exthdr rt exists drop

# Restrict to known next-headers only
ip6 nexthdr != { tcp, udp, icmpv6, esp, ah } \
    log prefix "unknown-nexthdr: " drop
```

Routing Expressions (rt)

Field	Description
rt nexthop	Next-hop address
rt classid	Routing realm classid
rt type	Route type

```
rt nexthop 192.168.1.1 accept
rt type blackhole drop
```

Socket Expression (kernel 4.18+)

```
# Match packets targeting a transparent socket (for TPROXY
setups)
socket transparent 1 accept

# Match by socket mark
socket mark 0x100 accept
```

OS Fingerprinting (osf, kernel 5.2+)

```
osf name "Linux"   accept
osf name "Windows" log prefix "windows-client: "
```

Requires the `nft_osf` module and `/etc/nftables/osf/pf.os` fingerprint database.

Ethernet, VLAN & ARP

Field	Family	Description
ether saddr	bridge	Source MAC address
ether daddr	bridge	Destination MAC address
ether type	bridge	EtherType
vlan id	bridge	802.1Q VLAN ID
vlan cfi	bridge	802.1Q CFI bit
vlan pcp	bridge	802.1Q priority code point
vlan type	bridge	EtherType in VLAN header
arp htype	arp	Hardware type
arp ptype	arp	Protocol type
arp operation	arp	request or reply
arp saddr ether	arp	Sender MAC address
arp daddr ether	arp	Target MAC address
arp saddr ip	arp	Sender IPv4 address
arp daddr ip	arp	Target IPv4 address

Chapter 8: Verdict & Action Statements

Accept, Drop, Reject

```
drop
accept

reject                                          #
port-unreachable (default)
reject with icmp type host-unreachable
reject with icmp type net-unreachable
reject with icmp type prot-unreachable
reject with icmp type port-unreachable

reject with icmpv6 type port-unreachable
reject with icmpv6 type admin-prohibited

reject with icmpx type port-unreachable         # inet: auto
icmp/icmpv6
reject with icmpx type admin-prohibited
reject with icmpx type no-route
reject with icmpx type host-unreachable

reject with tcp reset                           # TCP only
```

Logging

Option	Values	Description
prefix	quoted string (max 127 chars)	Log line prefix
group	0–65535	NFLOG netlink group
snaplen	integer bytes	Payload bytes to capture
queue-threshold	integer	Packets per NFLOG message

level	emerg alert crit err warning notice info debug	Syslog severity
flags	tcp sequence, tcp options, ip options, skuid, ether, all	Extra fields

```
log prefix "DROP: "
log level warn prefix "SUSPECT: "
log flags tcp sequence prefix "[TCP] "
log group 1 snaplen 128 queue-threshold 100
log prefix "DROP: " drop
```

Counter

```
tcp dport 443 counter accept           # anonymous per-rule
counter

nft add counter inet filter https-hits
nft add rule inet filter input \
    tcp dport 443 counter name https-hits accept

nft list counter  inet filter https-hits
nft reset counter inet filter https-hits
nft reset counters
```

Named counters declared in a table:

```
table inet filter {
    counter accepted-in {}
    counter dropped-in  {}
}
```

Rate Limiting & Quotas

```
tcp dport 22 ct state new limit rate 5/minute accept
tcp dport 22 ct state new limit rate 5/minute burst 10 packets
accept
limit rate 1 mbytes/second accept
```

```
# Inverse: match packets that EXCEED the rate
icmp type echo-request limit rate over 100/second drop

# Named limit
nft add limit inet filter ssh-rate { rate 5/minute; }
tcp dport 22 ct state new limit name ssh-rate accept

# Quota
nft add quota inet filter daily-out { 10 gbytes; }
quota name daily-out accept
nft add quota inet filter daily-cap { over 10 gbytes; }
quota name daily-cap drop
```

Named objects in a table:

```
table inet filter {
    limit ssh-rate   { rate 5/minute; }
    quota daily-out  { 10 gbytes; }
}
```

NAT Statements

NAT requires a chain of type `nat`. Conntrack applies NAT **on the first packet only** — subsequent packets are transformed automatically. This means:

- `accept` in a nat chain means "stop processing nat rules"; it does not bypass filter chains. A packet accepted in prerouting continues to the input or forward hook.
- Counters in a nat chain count connection initiations, not total packets.

Statement	Hook	Description
snat to addr	postrouting	Source NAT to explicit address
snat to addr-addr	postrouting	Source NAT with address range

snat to addr:port-port	postrouting	Source NAT with port range
masquerade	postrouting	SNAT to outgoing interface address
dnat to addr	prerouting, output	Destination NAT
dnat to addr:port	prerouting, output	DNAT with port rewrite
redirect to :port	prerouting, output	DNAT to local port
redirect to :port-port	prerouting, output	DNAT to local port range

NAT flags: `random`, `random-fully`, `persistent`

```
oifname "eth0" masquerade
ip saddr 192.168.0.0/24 snat to 203.0.113.1
snat to 203.0.113.1:1024-65535
tcp dport 80 dnat to 192.168.1.10:8080
tcp dport 80 redirect to :3128
snat to 203.0.113.1 persistent
```

> `masquerade` **vs** `snat`**:** `masquerade` re-evaluates the outgoing address per packet — necessary for DHCP / dynamic IPs. For a static external address, use `snat to addr` for both correctness and performance.

Packet Mangling

```
meta mark set 0x100
meta mark set meta mark or 0x10
meta mark set meta mark and 0xfff0
meta mark set ct mark        # restore conntrack mark
ct mark set meta mark        # save packet mark to conntrack

ip ttl set 64
ip6 hoplimit set 64
ip dscp set cs3

tcp flags set tcp flags & ~(ecn | cwr)
```

```
meta priority set 0x10:0x1
meta secmark set 42
```

Transparent Proxy (tproxy, kernel 4.18+)

```
table ip mangle {
    chain prerouting {
        type filter hook prerouting priority mangle; policy
accept;
        tcp dport 80 tproxy to 127.0.0.1:3128 meta mark set 1
    }
}

table ip filter {
    chain input {
        type filter hook input priority filter; policy drop;
        iifname lo                          accept
        ct state { established, related }   accept
        meta mark 1                         accept
    }
}
```

Required routing:

```
ip rule add fwmark 1 lookup 100
ip route add local 0.0.0.0/0 dev lo table 100
```

The proxy socket must have `IP_TRANSPARENT` set. Use `socket transparent 1` to match pre-intercepted packets in the input chain.

SYN Proxy (synproxy, kernel 4.19+)

Handles TCP handshakes in the kernel, protecting backend servers from SYN floods.

```
table ip raw {
    chain prerouting {
        type filter hook prerouting priority raw; policy
accept;
        tcp dport 80 tcp flags & (syn|ack) == syn \
```

```
            notrack \
            synproxy mss 1460 wscale 7 timestamp sack-perm
    }
}

table ip filter {
    chain input {
        type filter hook input priority filter; policy accept;
        ct state invalid drop
    }
}
```

Queue (NFQUEUE)

```
queue
queue num 0
queue num 0-3        # load-balance
queue num 0 bypass   # pass through if no listener
queue num 0 fanout   # copy to all listeners
```

Duplicate and Forward (netdev, kernel 4.3+)

```
dup to 10.0.0.250    # duplicate to monitoring host
dup to "mirror0"     # duplicate to a device
fwd to "eth1"        # forward without routing lookup (netdev
family)
```

Chapter 9: Sets

Anonymous Sets

```
tcp dport { 80, 443, 8080 } accept
tcp dport { 1-1023, 8080 } drop
ip saddr { 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 } accept
ip saddr != { 192.168.1.1, 192.168.1.2 } drop
```

Named Set Element Types

Type	Description
ipv4_addr	IPv4 address; with flags interval, accepts CIDR
ipv6_addr	IPv6 address; with flags interval, accepts CIDR
ether_addr	MAC address
inet_proto	Protocol number (0–255)
inet_service	Port number (0–65535); with flags interval, accepts ranges
mark	32-bit mark
ifname	Interface name string

Named Set Flags

Flag	Effect
constant	Contents frozen after creation
interval	Enables CIDR and port range elements

timeout	Elements auto-expire; per-element timeouts override the set default
dynamic	Populated by the dataplane during packet processing

Named Set Operations

```
nft add set inet filter web-ports \
    { type inet_service; elements = { 80, 443, 8080 }; }

nft add set inet filter rfc1918 {
    type ipv4_addr;
    flags interval;
    elements = { 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 };
}

nft add set inet filter blocklist {
    type ipv4_addr;
    flags timeout;
    timeout 1h;
}

nft add set inet filter ssh-abuse {
    type ipv4_addr;
    flags dynamic, timeout;
    timeout 15m;
}

nft add element inet filter blocklist { 203.0.113.5 }
nft add element inet filter blocklist { 203.0.113.6 timeout
30m }
nft delete element inet filter blocklist { 203.0.113.5 }
nft flush set inet filter blocklist
nft list  set inet filter blocklist

# In rules
ip saddr @blocklist drop
ip daddr != @rfc1918 drop
```

Dynamic Sets

```
nft add set inet filter ssh-abuse {
    type ipv4_addr;
    flags dynamic, timeout;
    timeout 15m;
}

nft add rule inet filter input ip saddr @ssh-abuse drop

nft add rule inet filter input \
    tcp dport 22 ct state new \
    add @ssh-abuse { ip saddr limit rate over 5/minute } \
    drop

nft add rule inet filter input tcp dport 22 ct state new
accept
```

Chapter 10: Maps & Verdict Maps

Maps

```
nft add map ip nat portmap {
    type inet_service : ipv4_addr;
    elements = { 80 : 192.168.1.10, 443 : 192.168.1.20 };
}
nft add rule ip nat prerouting dnat to tcp dport map @portmap

nft add map ip nat fwd-map {
    type inet_service : ipv4_addr . inet_service;
    elements = {
        8080 : 10.0.0.1 . 80,
        8443 : 10.0.0.2 . 443
    };
}
nft add rule ip nat prerouting dnat to tcp dport map @fwd-map

nft add element ip nat portmap { 8080 : 192.168.1.30 }
nft delete element ip nat portmap { 8080 }
```

Verdict Maps (vmap)

```
nft add map inet filter port-policy {
    type inet_service : verdict;
    elements = { 22 : accept, 80 : accept, 443 : accept, 25 :
drop };
}
nft add rule inet filter input tcp dport vmap @port-policy

# Inline vmap
nft add rule inet filter input tcp dport vmap {
    22  : accept,
    80  : jump http-chain,
    443 : jump https-chain
}

# Protocol dispatch
```

```
nft add rule inet filter input meta l4proto vmap {
    tcp  : jump tcp-rules,
    udp  : jump udp-rules,
    icmp : jump icmp-rules
}
```

Load Balancing

```
# Round-robin (counter increments per packet)
ip daddr 203.0.113.100 dnat to numgen inc mod 3 map {
    0 : 10.0.0.1,
    1 : 10.0.0.2,
    2 : 10.0.0.3
}

# Random
ip daddr 203.0.113.100 dnat to numgen random mod 3 map {
    0 : 10.0.0.1,
    1 : 10.0.0.2,
    2 : 10.0.0.3
}

# Consistent hashing by source IP (same client -> same
backend)
ip daddr 203.0.113.100 dnat to jhash ip saddr mod 3 map {
    0 : 10.0.0.1,
    1 : 10.0.0.2,
    2 : 10.0.0.3
}

# Symmetric hash
dnat to symhash mod 2 map { 0 : 10.0.0.1, 1 : 10.0.0.2 }
```

Concatenations

```
nft add set inet filter allowed-pairs {
    type ipv4_addr . inet_service;
    elements = { 192.168.1.10 . 22, 10.0.0.5 . 80 };
}
nft add rule inet filter input ip saddr . tcp dport
```

```
@allowed-pairs accept
```

Chapter 11: Stateful Objects

Named stateful objects are declared inside a table and survive rule reloads when the table persists.

Conntrack Helpers (ct helper)

Conntrack helpers track application-layer protocols that open secondary connections (FTP data, SIP media, etc.). Without a helper, the secondary connection arrives as `ct state new` and is blocked by a stateful firewall.

```
nft add ct helper inet filter ftp-helper { type "ftp" protocol
tcp; }

nft add rule inet filter input \
    tcp dport 21 ct state new \
    ct helper set "ftp-helper" \
    accept

nft add rule inet filter input ct state related accept
```

Common helper types: `ftp`, `tftp`, `sip`, `irc`, `pptp`, `h323`, `netbios-ns`, `snmp`, `amanda`, `rtsp`

Inline declaration:

```
table inet filter {
    ct helper ftp-helper {
        type "ftp" protocol tcp;
    }
}
```

Conntrack Timeouts (ct timeout)

```
nft add ct timeout inet filter web-timeout {
    protocol tcp;
    state {
        established 600,
        close_wait   30,
        fin_wait     30,
        time_wait    60
    };
}

nft add rule inet filter input \
    tcp dport 80 ct state new ct timeout set "web-timeout"
```

The default `nf_conntrack_tcp_timeout_established` is 432000 seconds (5 days) on some distributions. On high-traffic hosts this fills the conntrack table with stale entries. See Operational Gotchas for the full incident pattern.

Chapter 12: Flowtables

Flowtables (kernel 4.16+) offload established TCP and UDP connections from the full nftables pipeline. Only the first packet and control traffic traverse the ruleset — subsequent packets are handled directly.

> **Flowtable caveat:** Offloaded packets do **not** pass through logging, counters, or stateful match rules. Packet and byte counts in other chains will not include flowtable-accelerated traffic. This is by design.

```
nft add flowtable inet filter fastpath {
    hook ingress priority filter;
    devices = { eth0, eth1 };
}

nft add flowtable inet filter hw-fastpath {
    hook ingress priority filter;
    devices = { eth0, eth1 };
    flags offload;                          # hardware offload,
kernel 5.13+
}

nft add rule inet filter forward ct state established flow add
@fastpath

nft list flowtables
nft delete flowtable inet filter fastpath
```

Full Example

```
table inet filter {
    flowtable fastpath {
        hook ingress priority filter
        devices = { eth0, eth1 }
    }
```

```
    chain forward {
        type filter hook forward priority filter; policy drop;
        ct state established flow add @fastpath
        ct state new iifname "eth1" oifname "eth0" accept
    }
}
```

Chapter 13: Ruleset Management

Listing

```
nft list ruleset
nft -a list ruleset
nft -j list ruleset
nft list table inet filter
nft list chain inet filter input
nft list sets
nft list maps
nft list counters
nft list quotas
nft list limits
nft list flowtables
nft list ct helpers
```

Atomic Replacement

```
nft -f /etc/nftables.conf          # apply (file contains
flush ruleset)
nft -c -f /etc/nftables.conf       # syntax-check
nft list ruleset > /etc/nftables.conf.bak
```

Configuration File

```
#!/usr/sbin/nft -f

define WAN      = eth0
define LAN      = eth1
define LAN_NET  = 192.168.1.0/24
define ADMIN_IP = 10.0.0.10

include "/etc/nftables.d/*.conf"

flush ruleset
```

```
table inet filter {
    set trusted {
        type ipv4_addr
        elements = { $ADMIN_IP }
    }

    counter accepted-in {}
    counter dropped-in  {}

    chain input {
        type filter hook input priority filter; policy drop;

        iifname lo                          accept
        ct state { established, related }   accept
        ct state invalid \
            log prefix "ct-invalid: "       drop

        ip saddr @trusted                   accept
        tcp dport { 22, 80, 443 }           accept

        icmp type echo-request              accept
        icmpv6 type {
            nd-neighbor-solicit, nd-neighbor-advert,
            nd-router-solicit,   nd-router-advert
        }                                   accept
        icmpv6 type echo-request            accept

        counter name dropped-in \
            log prefix "input-drop: " level warn
    }

    chain forward {
        type filter hook forward priority filter; policy drop;
    }

    chain output {
        type filter hook output priority filter; policy
accept;
    }
}
```

Flush Commands

```
nft flush ruleset
nft flush table inet filter
nft flush chain inet filter input
nft flush set inet filter blocklist
```

Resetting Counters

```
nft reset rules chain inet filter input   # per-rule counters
in a chain
nft reset counter inet filter https-hits
nft reset counters
nft reset quotas
```

Persistence

```
systemctl enable --now nftables
systemctl reload nftables
systemctl status nftables
```

Chapter 14: iptables Migration

Concept Mapping

iptables concept	nftables equivalent
Table filter	table <family> filter
Table nat	table <family> nat (chain type nat)
Table mangle	table <family> mangle (priority mangle)
Table raw	table <family> raw (priority raw)
Table security	table <family> security (priority security)
Chain INPUT	chain with hook input
Chain OUTPUT	chain with hook output
Chain FORWARD	chain with hook forward
Chain PREROUTING	chain with hook prerouting
Chain POSTROUTING	chain with hook postrouting
-j ACCEPT	accept
-j DROP	drop
-j REJECT	reject [with …]
-j LOG --log-prefix	log prefix "…" [level …] [flags …]
-j MARK --set-mark N	meta mark set N
-j CONNMARK --save-mark	ct mark set meta mark
-j CONNMARK --restore-mark	meta mark set ct mark
-j DNAT --to-destination addr:port	dnat to addr:port
-j SNAT --to-source addr	snat to addr
-j MASQUERADE	masquerade

-j REDIRECT --to-ports port	redirect to :port
-j RETURN	return
-j QUEUE --queue-num N	queue num N
-j CLASSIFY --set-class	meta priority set
-j TPROXY	tproxy to addr:port (kernel 4.18+)
-j SYNPROXY	synproxy mss … wscale … (kernel 4.19+)
-m state --state ESTABLISHED,RELATED	ct state { established, related }
-m conntrack --ctstate INVALID	ct state invalid
-m limit --limit N/s --burst B	limit rate N/second burst B packets
-m recent	dynamic set with flags dynamic, timeout
-m iprange --src-range a-b	interval set: ip saddr @setname
-m multiport --dports 80,443	tcp dport { 80, 443 } (no module needed)
-m set --match-set NAME src	ip saddr @NAME
-m comment --comment "text"	comment "text" (kernel 5.10+)
-m hashlimit (per-source rate limit)	dynamic set with add @set { ip saddr limit rate over N/unit }
-m string --algo bm --string "GET"	No equivalent — use queue + userspace
-m rpfilter	fib saddr . iif oif missing drop
-i IFNAME	iifname "IFNAME"
-o IFNAME	oifname "IFNAME"
-s ADDR	ip saddr ADDR
-d ADDR	ip daddr ADDR
-p tcp	tcp (or ip protocol tcp)
--dport PORT	tcp dport PORT

--sport PORT	tcp sport PORT

Translation Tools

```
iptables-translate -A INPUT -p tcp --dport 22 -j ACCEPT
# . nft add rule ip filter INPUT tcp dport 22 counter accept

iptables-save  | iptables-restore-translate -4 >
/etc/nftables.conf
ip6tables-save | iptables-restore-translate -6 >>
/etc/nftables.conf
nft -c -f /etc/nftables.conf
```

> Translation is imperfect. Extensions like `--physdev`, complex `--tcp-flags` combinations, and multi-dimensional ipsets may not translate cleanly. Always review generated output before applying.

Compatibility Shim (iptables-nft)

```
iptables --version
# iptables v1.8.9 (nf_tables)  . nftables backend
# iptables v1.8.9 (legacy)     . legacy xtables backend
```

> **Do not mix native `nft` rules and `iptables-nft` in the same table.** `iptables-nft` creates and owns tables like `ip filter` and `ip nat`. Adding native nft rules to those tables causes unpredictable interactions. Keep native nft tables (e.g. `inet filter`) completely separate from iptables-nft tables.

Chapter 15: Operational Gotchas

Failure modes documented from real production incidents.

1. Docker publishes ports past your nftables rules

On Debian and Ubuntu, Docker writes iptables-legacy DNAT rules in the kernel's nat table. These rules redirect incoming traffic to container IPs **before your nftables input chain ever sees the packet**. A rule dropping port 8080 in nftables has no effect on a published Docker container on that port. This is not a misconfiguration — it is Docker's designed behaviour, and it is a genuine security gap on hosts running sensitive services in containers.

```
# Option 1: Restrict access with a higher-priority nftables
forward chain
# (priority -1 runs before Docker's filter chains at priority
0)
table inet docker-restrict {
    chain forward {
        type filter hook forward priority -1; policy accept;

        oifname "docker0" tcp dport 8080 \
            ip saddr != 192.168.1.0/24 \
            log prefix "docker-blocked: " \
            drop
    }
}

# Option 2: Disable Docker's firewall management
# /etc/docker/daemon.json: { "iptables": false }
# Requires you to write all DNAT and masquerade rules
yourself.
# Container port publishing (-p) stops working without manual
DNAT rules.
```

Check for Docker's rules: `iptables -t nat -L DOCKER` Check for mixing: `nft list tables` — Docker creates `ip filter` and `ip`

`nat`.

2. iif stops matching when an interface is recreated

`iif` stores the interface *index* at rule-load time. Kernel interface indices are monotonically increasing — if the interface is removed and recreated (VPN reconnect, container restart, bridge teardown), it gets a new index and the rule silently stops matching. This also causes `nft -f` to fail with "Interface does not exist" on boot when the interface hasn't been created yet — a documented failure mode with bridges, WireGuard tunnels, and VLAN interfaces.

When the nftables service fails to load on boot due to a missing interface, **the host starts with no firewall rules active**.

```
# Fragile — fails at boot if wg0 isn't up yet, breaks on
reconnect
iif "wg0" accept

# Safe — resolved at match time
iifname "wg0" accept
```

The performance difference is measurable only at very high packet rates. Default to `iifname` / `oifname` in all production rulesets.

3. ct state invalid drop kills asymmetric routing

`ct state invalid drop` is the right rule for single-path stateful firewalls. On any infrastructure with asymmetric packet paths — ECMP load balancing, BGP anycast, LVS direct routing, multi-path uplinks — it causes silent packet loss.

When return traffic arrives on a node that never processed the outbound SYN, conntrack classifies the packet as `invalid` and it is dropped. The symptom is intermittent connection failures that scale with traffic load and routing variation. It is commonly misdiagnosed as an application problem.

It can also cause conntrack table growth: when an LVS load balancer drops RST packets as invalid, neither the load balancer nor the backend sees the connection teardown, and the conntrack entries remain until they time out.

```
# Before deploying, add logging and observe what is actually
invalid
ct state invalid log prefix "ct-invalid: " level warn drop

# On multi-path infrastructure, consider removing the rule or
scoping it:
ct state invalid ip protocol tcp log prefix "ct-invalid: "
drop
```

4. Conntrack table exhaustion — silent drops under load

When `nf_conntrack_max` is reached, new connections are silently dropped. The kernel logs `nf_conntrack: table full, dropping packet` but this is easy to miss. The symptom is random connection failures misdiagnosed as application or network instability. Consul agent deregistrations, intermittent SSH failures, and unexplained service timeouts have all been traced to this root cause.

The default limit is 65536 on many distributions. The default `nf_conntrack_tcp_timeout_established` is 432000 seconds (5 days) on some distributions, meaning dead connections hold table slots for days.

```
# Check immediately
cat /proc/sys/net/netfilter/nf_conntrack_count
cat /proc/sys/net/netfilter/nf_conntrack_max

# Sizing formula (64-bit, N GB RAM):
# nf_conntrack_max = N * 1024^3 / 16384 / 2
# 8 GB  . 262144
# 16 GB . 524288
```

```
sysctl -w net.netfilter.nf_conntrack_max=262144   # immediate,
not persistent

# Persistent
cat > /etc/sysctl.d/10-conntrack.conf << 'EOF'
net.netfilter.nf_conntrack_max                        = 262144
net.netfilter.nf_conntrack_tcp_timeout_established   = 86400
net.netfilter.nf_conntrack_tcp_timeout_close_wait    = 60
net.netfilter.nf_conntrack_tcp_timeout_fin_wait      = 60
net.netfilter.nf_conntrack_tcp_timeout_time_wait     = 60
net.netfilter.nf_conntrack_generic_timeout           = 120
EOF
sysctl -p /etc/sysctl.d/10-conntrack.conf

# Bypass conntrack for high-volume traffic that doesn't need
state
table ip raw {
    chain prerouting {
        type filter hook prerouting priority raw; policy
accept;
        ip protocol udp udp dport 53 notrack
    }
}
```

5. Mixing firewall managers overwrites each other's rules

Running Docker, firewalld, UFW, raw `nft`, and `iptables-nft` on the same host means multiple tools are writing to the same netfilter subsystem. Whichever tool runs last wins. Worse, tools using different iptables versions internally can corrupt rules on `iptables-save` / `iptables-restore` cycles — documented cases include rules transforming from `-A CHAIN -m comment -j DROP` to `-A CHAIN -j DROP`, blocking all traffic and locking engineers off their own hosts.

After a firewalld upgrade that switched its backend from iptables to nftables, custom `firewall-cmd --direct` rules stopped working silently — the commands reported "success" but the rules had no effect.

```
# Identify what is managing your firewall
systemctl is-active firewalld
systemctl is-active ufw
nft list tables
iptables --version
iptables -t nat -L DOCKER 2>/dev/null   # check for Docker
rules
iptables-legacy -L -n 2>/dev/null       # check for legacy
rules
```

One tool, one firewall. If you use native nftables, disable firewalld and UFW. Keep Docker's tables (`ip filter`, `ip nat`) separate from your own.

6. Default deny breaks DHCP

Applying `policy drop` to the input chain without an explicit DHCP rule breaks network connectivity for any interface using DHCP. The host loses its IP address at the next renewal or reboot, frequently locking out remote access entirely.

```
# Allow DHCP renewal on any interface using DHCP
udp sport 67 udp dport 68 accept   # DHCPv4
```

7. Testing from localhost bypasses the input chain

Running `nmap` or `curl` against the host's own IP address from that same host often bypasses the input chain entirely. Traffic from local processes takes the `output` hook, not `prerouting` then `input`. Self-tests frequently produce false confidence that firewall rules are working when they are not.

Always test firewall rules from a separate host on the network.

Chapter 16: Troubleshooting & Debugging

Diagnosing a Rule That Isn't Matching

```
# 1. Confirm the rule is loaded
nft -a list chain inet filter input

# 2. Add a temporary counter and check packet counts
nft add rule inet filter input tcp dport 80 counter
# test, then: nft list ruleset | grep -A2 "dport 80"

# 3. Watch live ruleset changes
nft monitor

# 4. Check conntrack
conntrack -L
conntrack -L | grep 80
conntrack -E            # live events
```

Packet Tracing

Traces a specific flow through the entire netfilter stack. Generates high log volume — use specific matches.

```
nft add table inet trace-table
nft add chain inet trace-table input \
    { type filter hook input priority -500; }
nft add rule inet trace-table input \
    ip saddr 203.0.113.1 tcp dport 22 \
    meta nftrace set 1

nft monitor trace

nft delete table inet trace-table    # remove when done
```

Debug Output

```
nft --debug netlink -f /etc/nftables.conf
nft --debug eval    -f /etc/nftables.conf
nft --debug all     -f /etc/nftables.conf 2>&1 | head -100
```

Checking Conntrack

```
cat /proc/sys/net/netfilter/nf_conntrack_count
cat /proc/sys/net/netfilter/nf_conntrack_max
dmesg -w | grep nf_conntrack
journalctl -k -f | grep nf_conntrack
```

Checking for Mixed Firewall Managers

```
nft list tables
iptables --version
iptables-legacy -L -n 2>/dev/null
iptables -t nat -L DOCKER 2>/dev/null
systemctl is-active firewalld
systemctl is-active ufw
```

Chapter 17: Performance & Rule Ordering

nftables evaluates rules in strict sequence. Every packet traverses each rule in the chain until a terminating verdict is reached. The ordering of rules and the choice of match expressions has a direct and measurable effect on CPU load — particularly at high packet rates.

Expression Cost

Not all match expressions cost the same. The kernel processes them left-to-right within a rule, short-circuiting on the first mismatch. Putting cheap expressions before expensive ones pays off when they frequently disagree.

Cost	Expressions
Lowest	meta mark, meta iif, meta cpu — single integer comparison, no packet dissection
Low	ct state, ct mark, ct direction — conntrack hash is already computed; lookup is O(1)
Low	meta iifname, meta oifname — string comparison, but no header parsing
Moderate	ip saddr, ip daddr, ip6 saddr, ip6 daddr — requires reading the IP header
Moderate	ip protocol, ip6 nexthdr, meta l4proto — one byte in the IP header
Moderate	tcp dport, tcp sport, udp dport — requires reading the transport header
Higher	tcp flags — transport header read plus bitmask logic
Higher	tcp option — variable-length header walk
Higher	fib expressions — route table lookup

Highest	osf — fingerprint database scan

Rule Ordering Within a Chain

The single most impactful ordering decision in a stateful firewall: put `ct state established,related accept` first. In typical server traffic, 90–99% of packets belong to already-established connections. Accepting them in the first rule means those packets exit the chain after a single O(1) conntrack hash lookup, never touching any other rule.

```
chain input {
    type filter hook input priority filter; policy drop;

    # 1. Loopback — cheap meta integer match, exits
immediately
    iifname lo accept

    # 2. Established/related — O(1) conntrack lookup, handles
~95% of traffic
    ct state { established, related } accept

    # 3. Invalid — fast exit for bad packets before any header
parsing
    ct state invalid log prefix "ct-invalid: " drop

    # 4. New connections — now we parse headers, but only for
the small
    #    fraction of packets that are actually new
    tcp dport { 22, 80, 443 } accept

    # 5. ICMP — infrequent, fine to be last
    icmp type echo-request limit rate 10/second accept
}
```

The wrong ordering — putting port matches before conntrack — forces the kernel to dissect the TCP header for every packet, including the 95% that are established connections just wanting a fast `accept`.

Use Sets Instead of Individual Rules

Every rule is a sequential check. Replacing N rules with one set lookup reduces chain traversal from O(N) to a single O(1) hash lookup (for non-interval sets) or O(log N) for interval sets.

```
# Slow — three sequential rule evaluations for every new
packet
tcp dport 22  accept
tcp dport 80  accept
tcp dport 443 accept

# Fast — one set lookup
tcp dport { 22, 80, 443 } accept
```

For large IP allowlists or blocklists, a named set with `type ipv4_addr` is implemented as a hash table internally. Matching against 10,000 IPs costs the same as matching against 10 — O(1).

```
# Slow at scale — each rule is a linear chain step
ip saddr 10.0.0.1  accept
ip saddr 10.0.0.2  accept
# ... 10,000 more rules ...

# Fast at any scale — hash table lookup
ip saddr @trusted-hosts accept
```

Interval sets (for CIDR blocks and port ranges, using `flags interval`) use a different internal structure — a sorted range list — with O(log N) lookup. Still far faster than individual rules, but slower than plain hash sets. Use plain sets for exact-match values; reserve interval sets for prefix and range matching.

Bypass Conntrack for Stateless High-Volume Traffic

Conntrack allocates a table entry for every tracked flow. For high-volume stateless traffic — UDP DNS, SNMP polling, syslog — that table fills quickly and the allocation overhead is wasted. Use `notrack` in the `raw` table (priority .300, before conntrack) to bypass tracking entirely.

```
table ip raw {
    chain prerouting {
        type filter hook prerouting priority raw; policy
accept;

        # Bypass conntrack for DNS — high volume, stateless
        udp dport 53 notrack
        tcp dport 53 notrack

        # Bypass for internal monitoring
        ip saddr 10.0.0.0/8 udp dport 161 notrack  # SNMP
    }
}

table ip filter {
    chain input {
        type filter hook input priority filter; policy drop;
        iifname lo                          accept
        ct state { established, related }   accept

        # Packets marked notrack arrive as ct state untracked
        ct state untracked udp dport 53     accept
        ct state untracked tcp dport 53     accept
    }
}
```

Packets processed with `notrack` arrive at filter chains with `ct state untracked`. Match them explicitly rather than relying on `ct state established`.

Offload Forwarding with Flowtables

For a router or gateway, the highest-impact optimisation is a flowtable. Once a connection is offloaded, subsequent packets bypass the entire `prerouting`, `forward`, and `postrouting` pipeline, and the routing

decision is cached in the flowtable. CPU load for established forwarding traffic drops by 50–80% at high throughput.

```
table inet filter {
    flowtable fastpath {
        hook ingress priority filter
        devices = { eth0, eth1 }
    }

    chain forward {
        type filter hook forward priority filter; policy drop;

        # Offload as early as possible — first established
packet gets
        # added to the flowtable, all subsequent ones bypass
the chain
        ct state established flow add @fastpath

        ct state new iifname "eth1" oifname "eth0" accept
    }
}
```

The tradeoff: offloaded packets do not pass through logging rules or counters. If you need per-flow accounting, keep those connections out of the flowtable.

iif vs iifname — Performance vs Correctness

`iif` (integer index comparison) is faster than `iifname` (string comparison) because no string lookup is required at match time. The difference is measurable on dedicated firewall hardware at multi-Gbps line rates, and negligible on a general-purpose server. Given that `iif` breaks silently on interface recreation and prevents ruleset loading if the interface doesn't exist at boot (see Operational Gotchas), the performance gain is almost never worth the operational risk.

Use `iif` only on stable interfaces that will never be recreated (physical NICs named by firmware, not dynamic interfaces) and only when you have profiled and confirmed the gain is meaningful for your workload.

Avoid Redundant Counter Rules

Counters are implemented with atomic 64-bit increments. Each counter on a hot rule adds a memory barrier and cache-line contention on multicore systems. Anonymous counters (`counter` keyword inline) are convenient but add overhead to every packet that hits the rule.

```
# Overhead on every accepted packet
ct state { established, related } counter accept

# No overhead — counter only on the rule you actually want to
measure
ct state { established, related } accept
tcp dport 443 counter name https-hits accept
```

On the established/related rule in particular — which handles the majority of all traffic — the counter overhead is worth removing in performance-sensitive deployments.

Canonical Performance-Ordered Input Chain

Putting the above together:

```
chain input {
    type filter hook input priority filter; policy drop;

    # 1. Loopback: meta integer match, zero packet parsing
    iifname lo                          accept

    # 2. Established: conntrack hash, handles most traffic
    ct state { established, related }   accept

    # 3. Invalid: fast reject before any L4 parsing
    ct state invalid \
        log prefix "ct-invalid: "       drop

    # 4. Untracked: if you use notrack for some traffic
    ct state untracked udp dport 53     accept
```

```
    # 5. New connections: L4 header parsing happens here,
    #    but only for the small fraction of new flows
    tcp dport { 22, 80, 443 }           accept

    # 6. ICMP: infrequent, unproblematic at the end
    icmp type echo-request \
        limit rate 10/second            accept
    icmpv6 type {
        nd-neighbor-solicit, nd-neighbor-advert,
        nd-router-solicit,   nd-router-advert
    }                                   accept
    icmpv6 type echo-request \
        limit rate 10/second            accept

    # 7. Default: log then implicit policy drop
    log prefix "input-drop: " level warn
}
```

Chapter 18: Common Recipes

Minimal Stateful Server Firewall

```
#!/usr/sbin/nft -f
flush ruleset

table inet filter {
    chain input {
        type filter hook input priority filter; policy drop;

        iifname lo                          accept
        ct state { established, related }   accept
        ct state invalid \
            log prefix "ct-invalid: "       drop

        tcp dport { 22, 80, 443 }           accept

        icmp type echo-request \
            limit rate 10/second            accept

        icmpv6 type {
            nd-neighbor-solicit, nd-neighbor-advert,
            nd-router-solicit,   nd-router-advert
        }                                   accept
        icmpv6 type echo-request \
            limit rate 10/second            accept

        log prefix "input-drop: " level warn
    }

    chain forward {
        type filter hook forward priority filter; policy drop;
    }

    chain output {
        type filter hook output priority filter; policy
accept;
    }
}
```

NAT Gateway / Internet Router

Requires `net.ipv4.ip_forward = 1` in sysctl. If the WAN interface uses DHCP, add a DHCP rule in the input chain.

```
#!/usr/sbin/nft -f
flush ruleset

define WAN     = eth0
define LAN     = eth1
define LAN_NET = 192.168.1.0/24

table inet filter {
    chain input {
        type filter hook input priority filter; policy drop;

        iifname lo                          accept
        ct state { established, related }   accept
        ct state invalid \
            log prefix "ct-invalid: "       drop

        # DHCP renewal on WAN (if using DHCP)
        iifname $WAN udp sport 67 udp dport 68 accept

        iifname $LAN tcp dport 22           accept

        icmp type echo-request              accept
        icmpv6 type {
            nd-neighbor-solicit, nd-neighbor-advert,
            nd-router-solicit,   nd-router-advert
        }                                   accept
    }

    chain forward {
        type filter hook forward priority filter; policy drop;

        ct state { established, related }   accept
        ct state invalid                    drop

        iifname $LAN oifname $WAN \
            ip saddr $LAN_NET               accept
```

```
    }

    chain output {
        type filter hook output priority filter; policy
accept;
    }
}

table ip nat {
    chain postrouting {
        type nat hook postrouting priority srcnat; policy
accept;
        oifname $WAN masquerade
    }
}
```

Port Forwarding

```
table ip nat {
    chain prerouting {
        type nat hook prerouting priority dstnat; policy
accept;

        iifname "eth0" tcp dport 2222 dnat to 192.168.1.10:22
        iifname "eth0" tcp dport 80   dnat to 192.168.1.20:80
        iifname "eth0" tcp dport 443  dnat to 192.168.1.20:443
    }

    chain postrouting {
        type nat hook postrouting priority srcnat; policy
accept;
        masquerade
    }
}
```

Brute-Force Protection (Dynamic Blocklist)

```
table inet filter {
    set ssh-abuse {
        type ipv4_addr
```

```
        flags dynamic, timeout
        timeout 15m
    }

    chain input {
        type filter hook input priority filter; policy drop;

        iifname lo                          accept
        ct state { established, related }   accept
        ct state invalid \
            log prefix "ct-invalid: "       drop

        ip saddr @ssh-abuse \
            log prefix "ssh-blocked: "      drop

        tcp dport 22 ct state new \
            add @ssh-abuse { ip saddr limit rate over 5/minute
} \
            drop
        tcp dport 22 ct state new           accept

        tcp dport { 80, 443 }               accept

        icmp type echo-request              accept
        icmpv6 type {
            nd-neighbor-solicit, nd-neighbor-advert,
            nd-router-solicit,   nd-router-advert,
            echo-request
        }                                   accept
    }
}
```

Reverse Path Filtering with fib

```
table inet filter {
    chain prerouting {
        type filter hook prerouting priority raw; policy
accept;
        fib saddr . iif oif missing \
            log prefix "rpfilter: " drop
    }
```

```
    chain input {
        type filter hook input priority filter; policy drop;
        iifname lo                          accept
        ct state { established, related }   accept
        tcp dport { 22, 80, 443 }           accept
    }
}
```

Load Balancing with Consistent Hashing

```
table ip nat {
    chain prerouting {
        type nat hook prerouting priority dstnat; policy
accept;

        ip daddr 203.0.113.100 tcp dport 80 \
            dnat to jhash ip saddr mod 3 map {
                0 : 10.0.0.1,
                1 : 10.0.0.2,
                2 : 10.0.0.3
            }
    }
}
```

Flowtable Forwarding Fastpath

```
table inet filter {
    flowtable fastpath {
        hook ingress priority filter
        devices = { eth0, eth1 }
    }

    chain forward {
        type filter hook forward priority filter; policy drop;
        ct state established flow add @fastpath
        ct state new iifname "eth1" oifname "eth0" accept
    }
}
```

Docker Port Restriction

```
# Runs at priority -1, before Docker's chains at priority 0
table inet docker-restrict {
    chain forward {
        type filter hook forward priority -1; policy accept;

        oifname "docker0" tcp dport 8080 \
            ip saddr != 192.168.1.0/24 \
            log prefix "docker-blocked: " \
            drop
    }
}
```

Transparent Proxy

```
table ip mangle {
    chain prerouting {
        type filter hook prerouting priority mangle; policy
accept;
        tcp dport 80 tproxy to 127.0.0.1:3128 meta mark set 1
    }
}

table ip filter {
    chain input {
        type filter hook input priority filter; policy drop;
        iifname lo                          accept
        ct state { established, related }   accept
        meta mark 1                         accept
    }
}
```

```
ip rule add fwmark 1 lookup 100
ip route add local 0.0.0.0/0 dev lo table 100
```

IPv6 Router Advertisement Guard

```
table bridge filter {
    chain forward {
        type filter hook forward priority filter; policy
accept;
        icmpv6 type nd-router-advert iifname != "uplink" \
            log prefix "rogue-RA: " drop
    }
}
```

Chapter 19: Quick Reference

Hook + Chain Type Combinations

Use case	Family	Hook	Type	Priority
Bypass conntrack	inet	prerouting	filter	raw
Reverse path filter	inet	prerouting	filter	raw
DNAT / transparent proxy	ip	prerouting	nat	dstnat
Header mangling (inbound)	inet	prerouting	filter	mangle
Stateful input filtering	inet	input	filter	filter
Forward filtering	inet	forward	filter	filter
Header mangling (outbound)	inet	output	filter	mangle
SNAT / masquerade	ip	postrouting	nat	srcnat
Ingress filtering	netdev	ingress	filter	filter

Connection States

State	Meaning
new	First packet; no conntrack entry yet
established	Flow tracked in both directions
related	Related secondary flow (e.g. FTP data)
invalid	Cannot be classified — no conntrack entry
untracked	Explicitly exempted with notrack

Named Priority Constants

Name	Value
raw	.300
mangle	.150
dstnat	.100
filter	0
security	50
srcnat	100

Conntrack Sizing

```
nf_conntrack_max = RAM_bytes / 16384 / 2

4 GB  .  131072
8 GB  .  262144
16 GB .  524288
32 GB . 1048576
```

```
cat /proc/sys/net/netfilter/nf_conntrack_count
cat /proc/sys/net/netfilter/nf_conntrack_max
```

Essential sysctl

```
net.ipv4.ip_forward                                 = 1
net.ipv6.conf.all.forwarding                        = 1
net.netfilter.nf_conntrack_max                      = 262144
net.netfilter.nf_conntrack_tcp_timeout_established = 86400
net.netfilter.nf_conntrack_tcp_timeout_close_wait  = 60
net.netfilter.nf_conntrack_tcp_timeout_fin_wait    = 60
net.netfilter.nf_conntrack_tcp_timeout_time_wait   = 60
net.netfilter.nf_conntrack_generic_timeout          = 120
```