nftables

ytshih (2026, CC BY-SA)

Content

- Introduction to nftables
  - What is netfilter / nftables
  - nftables Rulesets, Families, Tables, Hooks, Chains, Rules
- Configuring nftables
  - Command line utils
  - nftables components
  - Expressions and statements
- Applications

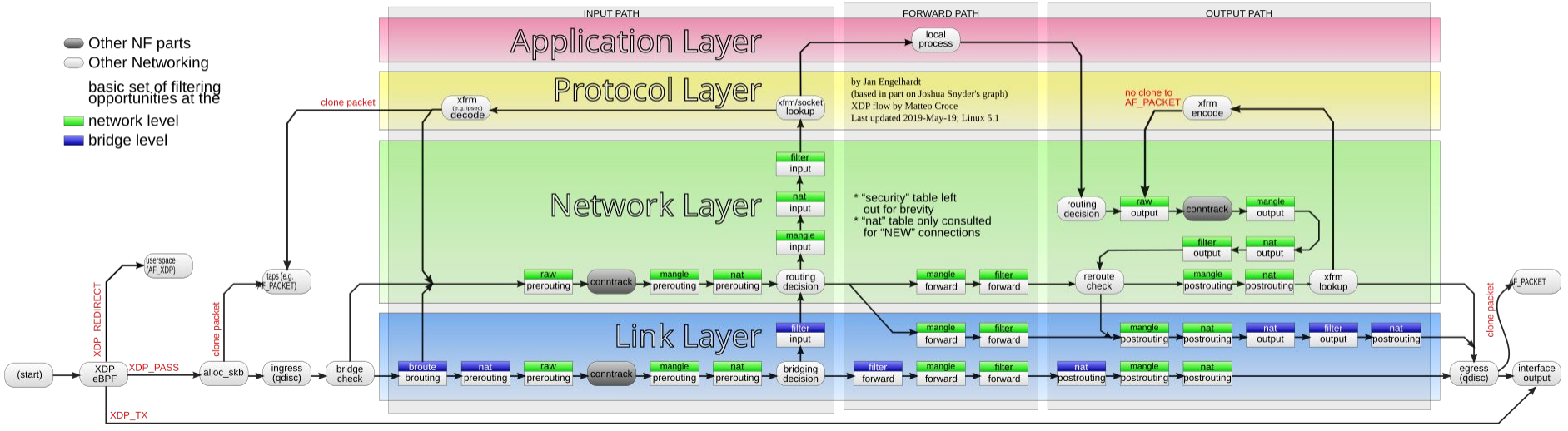
Introduction to nftables

What is netfilter

- Community-driven collaborative FOSS project (Free and Open Source Software)
- Provides packet filtering software for the Linux 2.4.x and later kernel
  - **Packet filtering**
  - **Network address (and port) translation**
  - Userspace packet queueing
  - Packet mangling

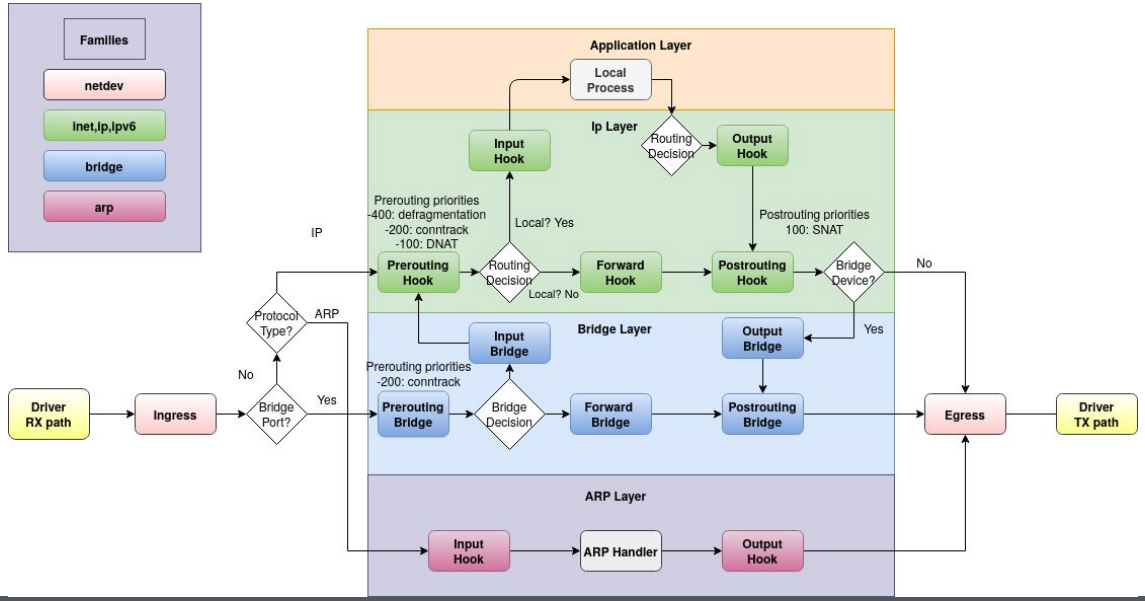
Packet Flow in Netfilter (xtables)

- iptables, arptables, ebtables
- Allows kernel modules to register callback functions



Packet Flow in Netfilter (nftables)

- Replaces the legacy xtables with **nftables families**
- No predefined tables in nftables framework.



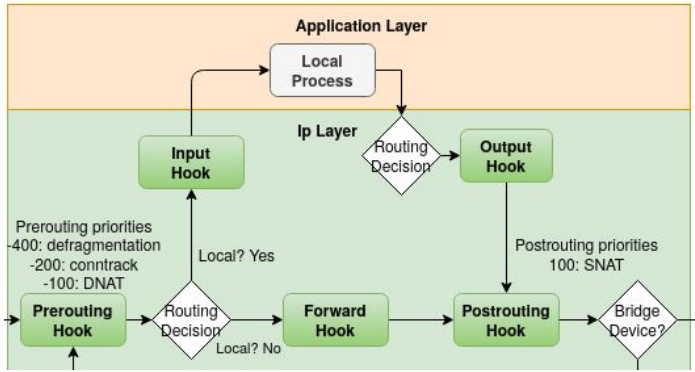
What is nftables

- The modern Linux kernel packet classification framework
- Uses a new syntax inspired by *tcpdump*
- Configured via the user-space tool ***nft***
- Replaces the legacy xtables with **nftables families**
  - iptables => ip family
  - arptables => arp family
  - ebtables => bridge family

Families

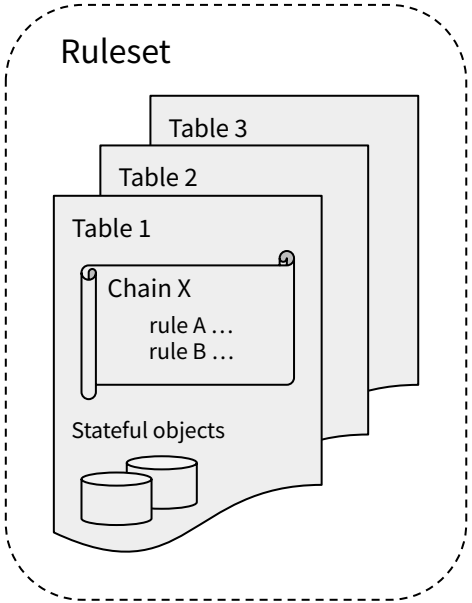
- With nftables, multiple networking levels are abstracted into families.
- Available families are
  - ip
  - ip6
  - inet (ipv4 + ipv6 dual stack)
  - arp
  - bridge
  - netdev
- We are only focusing on **ip** and **inet** family in this course.

Netfilter Hooks



- Prerouting
  - Sees all incoming packets, before any routing decision.
- Input
  - Sees incoming packets that have now been routed to the local system.
- Forward
  - Sees incoming packets that are not addressed to the local system
- Output
  - Sees packets that originated from processes in the local machine.
- Postrouting
  - Sees all packets after routing, just before they leave the local system.

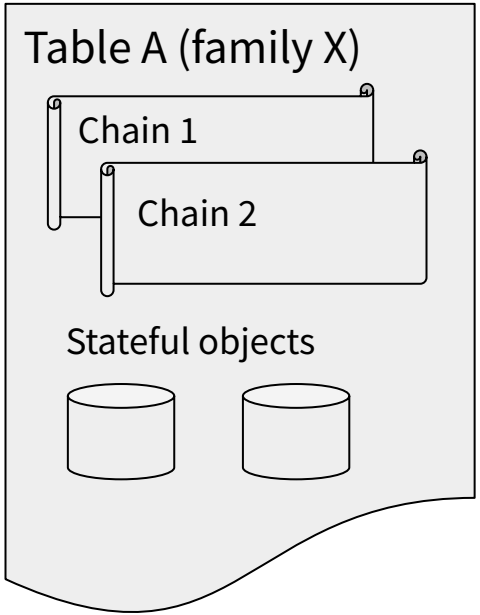
nftables Components



- Ruleset
  - A collection of tables that currently in place in kernel.
- Tables
  - Containers for chains and stateful objects.
- Chains
  - Containers for rules.
- Rules
  - Take action on network packets based on whether they match specified criteria.
- Stateful objects
  - Maintain information about packet flows and connection states.
  - e.g. source ip, source port, destination ip, destination port

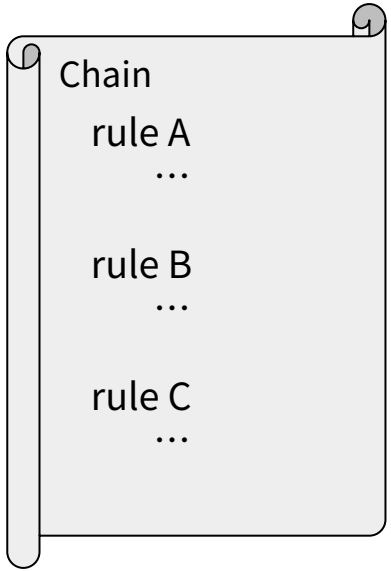
Tables

- The top-level structure within an nftables ruleset.
- Contain chains and other stateful objects.
- Each table belongs to exactly one family.

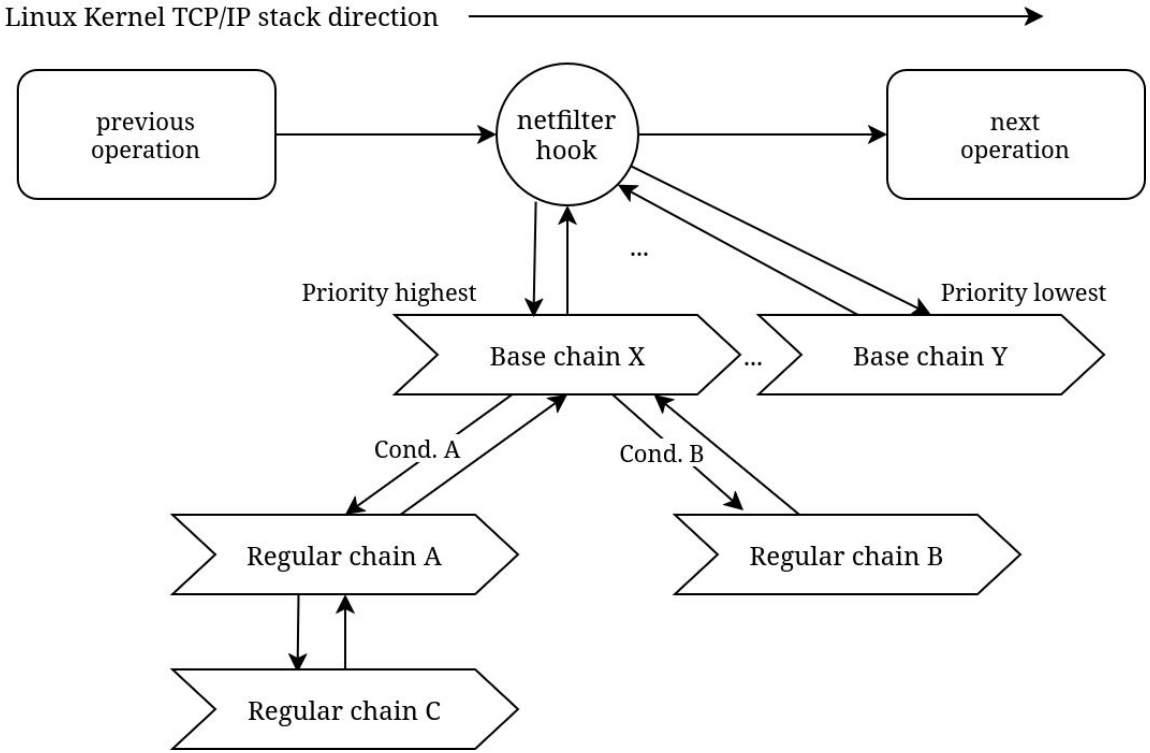


Chains

- The structure that contains rules.
- Base Chains
  - Those that are registered into the netfilter hooks.
  - These chains see packets flowing through your Linux TCP/IP stack.
- Regular Chains
  - Not attached to a Netfilter hook
  - By itself a regular chain does not see any traffic.



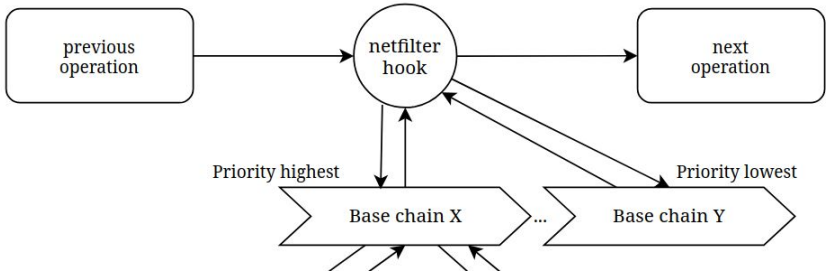
Chains - Flow Chart



Base Chain - Types

- Filter
  - Used to filter packets.
- Route
  - Used to reroute packets, only for the ***output*** hook (other hooks use type filter instead).
- NAT
  - Used to perform Networking Address Translation (NAT).
  - **Only the first packet** of a given flow hits; subsequent packets bypass it.

Base Chain - Priority



- Determines the ordering of the chains.
- If two chains have the same priority, the evaluation order can’t be guaranteed.

***nft* Keyword**	**Value**	**Description**
`raw`	-300	Before connection tracking operation
`mangle`	-150	Mangle operation (Alter ip headers or metadatas)
`dstnat`	-100	Destination NAT
`filter`	0	Filtering operation
`srcnat`	100	Source NAT

Stateful objects

- Sets
  - Store states as unique values.
  - e.g. Set *S* = {1, 2, 3, 5}. *2* is in set *S*, while *4* is not.
- Maps
  - Store states as key-value pairs.
  - e.g. Map *M* = {1: a, 2: b}. The key *1* in map *M* is mapped to value *a*.
- Counters
  - Store total number of packets and total bytes.

nftables Configuration

Symbol Explanation

- Apply to all the code sections afterwards.

Bold	Keywords or necessities
Underscore	Choose from predefined options
Italic	User defined identifier
`[value]`	Optional value
`A|B`	A or B
`<Type>`	A valid value from the type

Configure nftables

- Use command line tool ***nft***
- **nft [-f filename | -i | cmd ...]**
- e.g., Add counter to all icmp packets

Directly by command line

```
> nft add element table inet myfilter set ip-pool { 1.1.1.1 }
```

Read from file

```
> nft -f nftables.conf
```

Configuration file

```
# nftables.conf
destroy table inet filter
```

Frequently Used Commands

- Check the whole ruleset

```
> nft list ruleset
```

- Flush / delete the whole ruleset

```
> nft flush ruleset
```

- Apply ruleset from a file
  - This is an atomic operation.
  - Add `-c` for dry-run and syntax checking.
  - We will mainly use this method in this course.

```
> nft -c -f /path/to/config
```

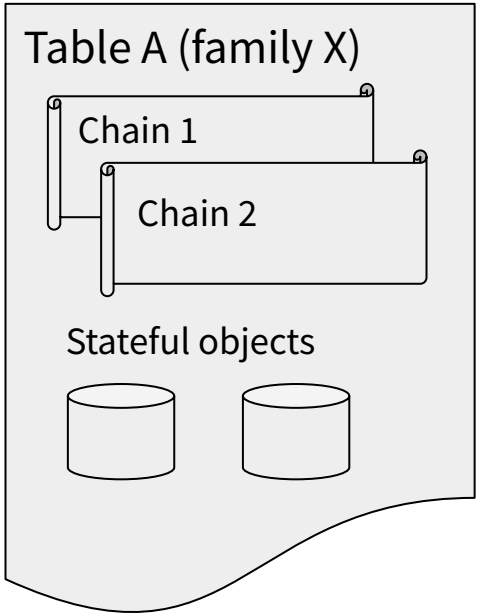
Tables - Configuration File

```
# Add this to flush table before apply the new one.
destroy table family table_name

[add] table family table_name {
  counter counter_name1 {
  }
  (other stateful objects)
  chain chain_name1 {
  }
  chain chain_name2 {
  }
  (other chains)
}
```

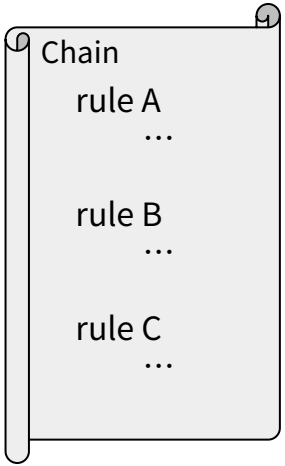
stateful objects

chains



Chains - Configuration File

- The rules in chains are processed **from top to bottom**.




```
table family table_name {
  chain chain_name {
    #(rule 1) # The first to be processed
    #(rule 2) # The second
    #(rule 3) # The last
    # Fallback to default policy
  }
}
```

Base Chains - Configuration File

- If multiple base chains exist in the same hook, an accept in a base chain will not be the final verdict.
  - Packets are still going to other base chains with lower priorities
- Apply **policy** when packets reaching the end of the chain.
- Default policy is accept (passthrough).

```
table family table_name {
  chain chain_nameA { # This chain will run first
    type type hook hook priority priority;
    [policy <accept|drop>;]
    #(other rules)
  }
  chain chain_nameB { # This will run even if chainA accept
    type type hook hook priority priority + 1;
  }
}
```

Rules - Configuration File

- Consists of zero or more expressions followed by one or more statements.
  - Expressions: matching packets
  - Statements: conducting actions
- Expressions and statements are evaluated **from left to right**.

```
table family table_name {
  chain chain_name {
    [expression1 [expression2 ...]] statement1 [statement2 ...];
    ip protocol icmp ip saddr 1.1.1.1 counter accept
    #(other rules)
  }
}
```

nftables Expressions

nftables Expressions

- Matching IP Headers
  - e.g. source ip, destination ip, ip protocols
- Matching Transport Protocols Headers
  - e.g. tcp, udp
- Matching Metainformation
  - metainformation: informations that describe the packet itself
  - e.g. input / output interface, socket uid / gid
- Matching Connection Tracking Metainformation
  - e.g. connection track state

Matching IP Headers

- **ip saddr|daddr** <ipv4_addr>
- **ip protocol** <inet_proto>
- **meta l4proto** <inet_proto>
  For v4 / v6 dual stack support

```
table inet filter {
  chain input {
    type filter hook input priority filter; policy accept;
    ip saddr 1.1.1.1 accept;
    ip daddr 192.168.1.1 accept;
    ip protocol icmp drop;
  }
}
```

Matching Transport Protocols Headers

- **tcp|udp sport|dport** <inet_service>
- **meta l4proto {tcp, udp} th sport|dport** <inet_service>

^ Stands for “transport header”

```
table inet filter {
  chain input {
    type filter hook input priority filter; policy drop;
    tcp dport 22 accept comment "ssh";
    meta l4proto {tcp, udp} th dport 443 accept comment "http 2/3";
  }
}
```

Matching Packet Metainformation

- **meta iif|oif** <iface_index>
- **meta iifname|oifname** <ifname>
- **meta skuid** <uid> / **meta skgid** <gid>

  ^ Stands for “socket uid”

```
table inet filter {
  chain input {
    type filter hook input priority filter; policy drop;
    meta iif "lo" accept comment "allow from localhost";
    meta skuid ccna accept comment "allow user ccna";
  }
}
```

Lab 1 - Simple Matching

- Apply the following nftables config (i.e. `nft -f filter.conf`).
- ping 1.1.1.1 (should pass)
- ping 8.8.8.8 (should fail)
- Flush ruleset to cleanup (i.e. `nft flush ruleset`).

```
# filter.conf
table inet filter {
  chain input {
    type filter hook input priority filter; policy accept;
    ip saddr 1.1.1.1 accept;
    ip protocol icmp drop;
  }
}
```

Connection tracking system

- Tracking the states of connections.
- A component of netfilter.

State	Description
new	Netfilter has seen packets between this pair of hosts in **only one direction**.
established	Netfilter has seen valid packets travel in **both directions** between this pair of hosts.
related	This connection was **initiated after the main connection**, as expected from normal operation of the main connection.
invalid	Assigned to packets that do not follow the expected behavior.
untracked	Dummy state assigned to packets that have been **explicitly excluded** from conntrack.

Matching Connection Tracking Metainformation

- **ct state** <ct_state>

```
table inet filter {
  chain input {
    type filter hook input priority filter; policy accept;
    ct state invalid drop comment "early drop of invalid connections";
    ct state {established, related} accept comment;
    ct state new drop comment "drop that not initiated from us";
  }
}
```

Lab 2 - Matching with Connection Tracking

- Apply the following nftables config (i.e. `nft -f filter.conf`).
- ping 1.1.1.1 (should pass)
- ping from Internet (should fail)
- Flush ruleset to cleanup (i.e. `nft flush ruleset`).

```
# filter.conf
table inet filter {
  chain input {
    type filter hook input priority filter; policy accept;
    ct state {established, related} accept;
    ip protocol icmp drop;
  }
}
```

nftables Statements

nftables Statements

- Verdicts
  - Accept, Drop, Jump, Return
- Reject traffic
- Source NAT
  - Masquerade
- Destination NAT
- Packet duplicating

Statements - Accept

- Accept the packet and stop the remain rules evaluation.

```
table inet filter {
  chain input {
    type filter hook input priority filter;

    accept comment "universal accept verdict";
  }
}
```

Statements - Drop

- Drop the packet and stop the remain rules evaluation.

```
table inet filter {
  chain input {
    type filter hook input priority filter;

    drop comment "universal drop verdict";
  }
}
```

Statements - Jump

- `jump chain`
- Continue at the first rule of *chain*.
- It will continue at the next rule after a return statement is issued.

```
table inet filter {
  chain ssh {
    accept;
  }
  chain input {
    type filter hook input priority filter;
    tcp dport 22 jump ssh;
    tcp dport 22 drop comment "this will never match";
  }
}
```

Statements - Return

- Return from the current chain and continue at the next rule of the last chain.
- In a base chain it is equivalent to accept.

```
table inet filter {
  chain tcp {
    tcp dport 22 return;
    drop;
  }
  chain input {
    type filter hook input priority filter;
    meta l4proto tcp jump tcp;
  }
}
```

Statements - Counter

- A counter counts both the total number of packets and the total bytes.
- Run `nft list ruleset` to see counter contents.

```
table inet filter {
  chain input {
    type filter hook input priority filter; policy accept;
    ip protocol icmp counter;
  }
}
```

Lab 3 - Jump Between Chains

- Apply the following nftables config (i.e. `nft -f filter.conf`).
- ping 1.1.1.1 (only the counter in chain input should increase)
- ping 8.8.8.8 (both counter should increase)
- Flush ruleset to cleanup (i.e. `nft flush ruleset`).

```
# filter.conf
table inet filter {
  chain icmp_counter {
      ip saddr 1.1.1.1 return; # exclude 1.1.1.1
      counter;
  }
  chain input {
      type filter hook input priority filter; policy accept;
      ip protocol icmp jump icmp_counter;
      ip protocol icmp counter;
  }
}
```

Rejecting traffic

- **`reject [with icmp|icmpv6|icmpx type reason]`**
- Reject the traffic by icmp unreachable or prohibited packet and stop the remain rules evaluation.
- Default reason is **`port-unreachable`**.

```
table inet filter {
  chain filter {
    type filter hook input priority filter - 1; policy accept;
    tcp dport 22 reject with icmp port-unreachable;
  }
}
```

Statements - Source NAT (SNAT)

- **`snat to`** `<ipv4_addr>`
- The statefule NAT involves the conntrack kernel engine.
  - **Not every packet will go through prerouting / postrouting nat type chain.**
- Specify in postrouting chains.

```
table inet filter {
  chain postrouting {
     type nat hook postrouting priority srcnat;
     ip saddr 192.168.1.0/24 oif eth0 snat to 1.2.3.4;
  }
}
```

Statements - Masquerade

- **`masquerade`**
- Masquerade is a special case of SNAT that source address is automatically set to the address of the output interface.

```
table inet filter {
  chain postrouting {
     type nat hook postrouting priority srcnat;
     ip saddr 192.168.1.0/24 oif eth0 masquerade;
  }
}
```

Statements - Destination NAT (DNAT)

- **dnat ip|ip6 to** <inet_addr>[:<inet_service>]
- Specify in prerouting chains.

```
table inet filter {
  chain prerouting {
     type nat hook prerouting priority dstnat;
     iif eth0 tcp dport 80 dnat ip to 192.168.1.120:8080;
     iif eth0 tcp dport 443 dnat ip to 192.168.1.120;
  }
}
```

Statements - Duplicating packets

- **dup to** <inet_addr>
- Duplicate packets to another destination address.
- Is used to copy selected traffic for further inspection.
- Often use with *mangle* priority in prerouting hook.

```
table ip filter {
  chain prerouting {
     type filter hook prerouting priority mangle;
     tcp dport 80 dup to 172.20.0.2;
  }
}
```

nftables
Stateful Objects

Stateful Objects - Counter

- A counter can be anonymous or named.
- A counter counts both the total number of packets and the total bytes.
- Reset all counters. (only works on named counters)

```
> nft reset counters
```

```
table inet filter {
  counter ping {
    comment "ipv4 icmp counter"
    packets 2 bytes 128
  }
  chain input {
    type filter hook input priority filter; policy accept;
    ip protocol icmp counter # anonymous counter
    ip protocol icmp counter name "ping"
  }
}
```

Stateful Objects - Set

- The set objects are internally represented using performance data structures such as **hashtables** and **binary trees**.
- A set can either be anonymous or named.

```
set monitoring { # named set
  type ipv4_addr
  elements = { 192.168.1.1, 192.168.1.2 }
}
chain prerouting {
  type filter hook prerouting priority mangle;
  ct state new add @monitoring {ip saddr};
  ip saddr @monitoring tcp dport {80, 443} dup to 172.20.0.2;
}
```

Set - Specifications

- **`type`**
  - Define data types
  - e.g. `ipv4_addr`, `inet_service`, `inet_proto`.
- **`typeof`**
  - let nftables resolve the base type for you.
- **`elements`**
  - Initialize the set with some elements in it.
- **`timeout` *`time`***
  - Remove the element after specific time.

```
set monitoring { # named set
  type ipv4_addr
  elements = { 192.168.1.1, 192.168.1.2 }
}
chain prerouting {
  type filter hook prerouting priority mangle;
  ct state new add @monitoring {ip saddr};
  ip saddr @monitoring tcp dport {80, 443} dup to
172.20.0.2;
}
```

Stateful Objects - Map

- A set that returns a value instead of just an “in set / not in set” result.
- A map can either be anonymous or named.

```
table ip nat {
  map porttoip {
     type inet_service: ipv4_addr;
     elements = {
       80 : 192.168.1.100, 8888 : 192.168.1.101
     }
  }
  chain postrouting {
     type nat hook postrouting priority srcnat;
     snat to tcp dport map @porttoip;
  }
}
```

Loadbalancing - Roundrobin

- Use nftables number generator for increasing sequence.
- Use map for dnat destination.

```
map lbpool {
  typeof numgen inc mod 2: ip daddr;
  elements = {
      0: 192.168.10.100,
      1: 192.168.20.200
  }
}
chain prerouting {
  type nat hook prerouting priority dstnat;
  dnat to numgen inc mod 2 map @lbpool;
}
```

Loadbalancing - Weighted Random

- 60% to 192.168.10.100
- 40% to 192.168.20.200

```
map lbpool {
  typeof numgen random mod 10: ip daddr;
  flags interval;
  elements = {
      0-5: 192.168.10.100,
      6-9: 192.168.20.200
  }
}
chain prerouting {
  type nat hook prerouting priority dstnat;
  dnat to numgen random mod 10 map @lbpool;
}
```

Loadbalancing - Hash

- Use nftables internal hashing mechanisms.
- The dot (.) means that the expressions before and after are a tuple.

```
table ip nat {
  chain prerouting {
     type nat hook prerouting priority dstnat;
     dnat to jhash ip saddr . tcp dport mod 2 map {
       0: 192.168.20.100,
       1: 192.168.30.100
     }
  }
}
```

Applications

Fail2ban - SSH Jail

- https://people.cs.nycu.edu.tw/~ytshih/ccna-2025/lab2/fail2ban.conf

```
table inet f2b-table {
  set addr-set-sshd {
    type ipv4_addr
    elements = {}
  }

  chain f2b-chain {
    type filter hook input priority filter - 1; policy accept;
    tcp dport 22 ip saddr @addr-set-sshd reject with icmp port-unreachable
  }
}
```

Lab 4. systemd-networkd NAT - Setup

- Create the file with the following content.

```
# /etc/systemd/nspawn/ccna.nspawn
[Network]
Port=tcp:8080:80
```

- Run the container.

```
> sudo importctl pull-tar -m --verify=checksum \
    https://people.cs.nycu.edu.tw/~ytshih/ccna-2025/lab2/ccna.tar.xz ccna
> sudo machinectl start ccna # Start container
> sudo machinectl shell ccna # Get the container shell
```

Lab 4. systemd-networkd - SNAT (Postrouting)

- https://people.cs.nycu.edu.tw/~ytshih/ccna-2025/lab2/systemd-networkd.conf

```
table ip io.systemd.nat {
  set masq_saddr {
      type ipv4_addr
      flags interval
      elements = { 192.168.169.0/24 }
  }

  chain postrouting {
      type nat hook postrouting priority srcnat + 1; policy accept;
      ip saddr @masq_saddr masquerade
  }
...
```

Lab 4. systemd-networkd - DNAT (Prerouting)

- https://people.cs.nycu.edu.tw/~ytshih/ccna-2025/lab2/systemd-networkd.conf

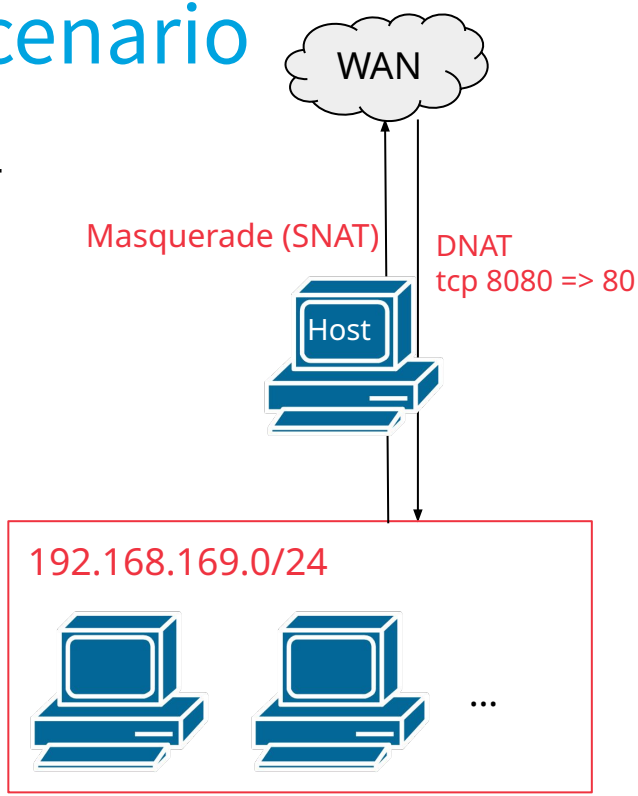
```
...
  map map_port_ipport {
      type inet_proto . inet_service : ipv4_addr . inet_service
      elements = { tcp . 8080 : 192.168.169.172 . 80 }
  }

  chain prerouting {
      type nat hook prerouting priority dstnat + 1; policy accept;
      fib daddr type local dnat ip to meta l4proto . th dport map @map_port_ipport
  }

  chain output {
      type nat hook output priority dstnat + 1; policy accept;
      ip daddr != 127.0.0.0/8 oif "lo" dnat ip to meta l4proto . th dport map @map_port_ipport
  }
...
```

Lab 4. systemd-networkd - Scenario

- Masquerade if source ip in 192.168.169.0/24
- DNAT if destination port in the map



VyOS - Firewall

- https://people.cs.nycu.edu.tw/~ytshih/ccna-2025/lab2/vyos-firewall.conf

```
...
chain VYOS_ZONE_FORWARD {
  type filter hook forward priority filter + 1; policy accept;
  oifname "eth3" counter packets 1752805 bytes 10211037209 jump VZONE_315
  oifname "eth4" counter packets 142412129 bytes 180802490699 jump VZONE_316
  oifname "eth5" counter packets 3274125 bytes 6155288548 jump VZONE_324
  oifname { "lo", "eth6" } counter packets 1127508 bytes 1852749730 jump VZONE_LAN
  oifname { "eth0", "eth1", "eth2" } counter packets 105946188 bytes 55009362522 jump VZONE_WAN
}
...
```

References

- nftables wiki
  - https://wiki.nftables.org/wiki-nftables/index.php/Main_Page
- fail2ban
  - https://github.com/fail2ban/fail2ban
- systemd
  - https://www.freedesktop.org/software/systemd/man/latest/systemd.html
- VyOS
  - https://vyos.io/
- netfilter.org
  - https://www.netfilter.org/