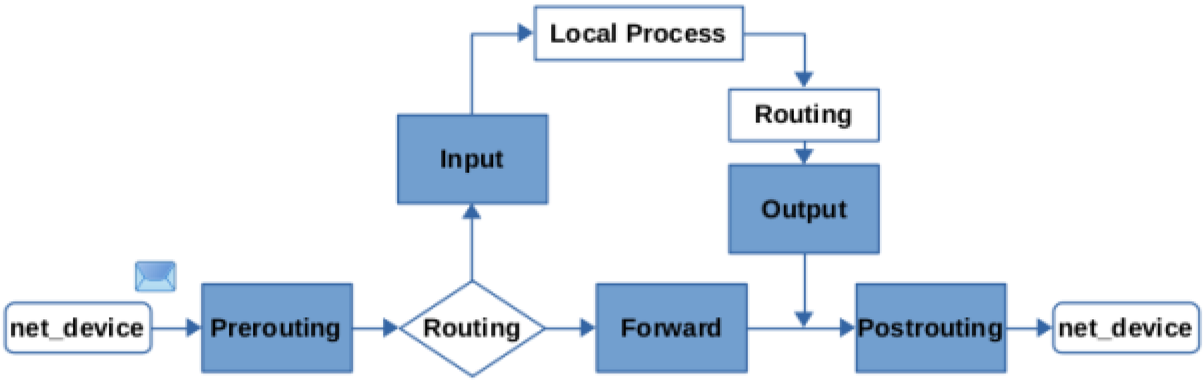
NFTables

Hooks

Every packet that enters a system, whether incoming or outgoing will trigger some hooks as it traverses through the Linux kernel's networking stack. Those five hooks have been present in the Linux kernel for a very long time. Linux kernel allows rules that are associated with these hooks to interact with the network traffic. Nftables has five hooks including prerouting, input, output, post routing, forward, and ingress.



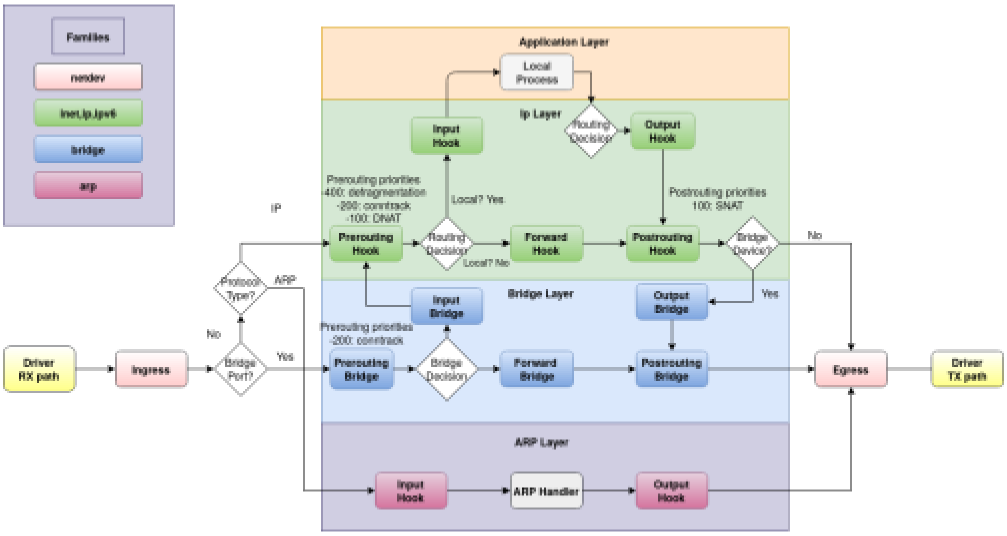
When traffic flow goes into a local machine, first, it faces the prerouting hook and then input hook. Next, the traffic generated by the local machine's processes follows the output hook and then the postrouting hook as shown in the next figure. The packets destined to your network but are not addressed to the local node will face the forward hook after following prerouting and then postrouting path. Ingress hook, however, as a new hook in nftables, is a hook that is placed before all the hooks behind the prerouting hook and can filter traffic on layer 2 OSI model. With this hook, therefore, early filtering policies can be defined (2019).

Address Families

Address families determine the type of incoming and outgoing packets processed by nftables. For each address family, the Linux kernel contains specific hooks at different stages of the packet processing paths, which invoke nftables to decide either allow or drop a packet only if relevant rules for these hooks such as input or output are defined. These address families are as follows:

Family	Description
ip:	IPv4 address family.
ip6:	IPv6 address family.
inet:	Supports Both IPv4 and IPv6 address families.
arp:	ARP address family, handling IPv4 ARP packets at layer 2 OSI model.
bridge:	Bridge address family, handling packets traversing a bridge device at layer 2.
netdev:	Netdev address family, handling packets from ingress hook working before layer 3.

Table with netdev family, by means of ingress hook, allows early filtering traffic before they reach other filters below layer 3 on the OSI model. netdev family with ingress hook is an ideal stage to drop packets that result from DDOS attacks since this hook works very early in the packet path of networking. There are different hooks for different "family" types. The following schematic shows packet flows through Linux networking:



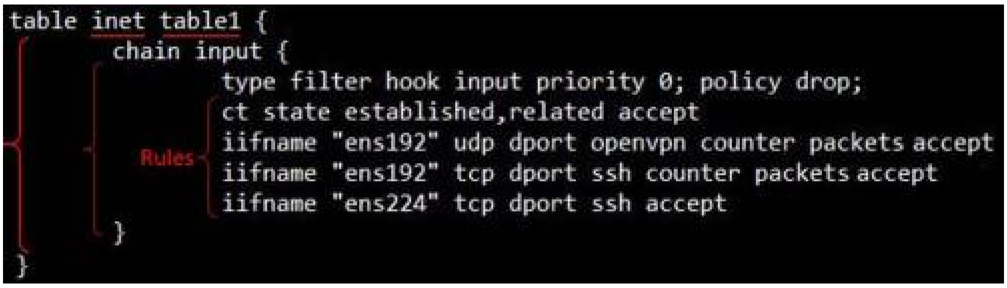
Variables

Repetition is bad. To simplify things, nftables supports variables. Instead of repeating an interface multiple times, you define it at the beginning of your configuration file. After that, you will be using the variable. Example: defining interfaces.

```
define ext_if = eth0
define int_if = eth1
define all_if = { $ext_if, $int_if }
```

Tables

Within the configuration of nftables, a table is at the top of the ruleset. It consists of chains, which are containers for rules. **Overview: Tables -> Chains -> Rules**. The following structure can be seen in the nftables configuration file as shown in the next figure. As it is clear from this example below, a table followed by an address family, in this case "inet", and then it is followed by its defined name that is "table1" with an open and a close curly bracket.



```
table inet table1 {
        chain input {
                type filter hook input priority 0; policy drop;
                ct state established,related accept
                iifname "ens192" udp dport openvpn counter packets accept
                iifname "ens192" tcp dport ssh counter packets accept
                iifname "ens224" tcp dport ssh accept
        }
}
```

Chains

Chains are a container of rules and are located inside a table. Chains have two types.

- A base chain is an entry point for packets from the networking stack, where a hook value is specified.
- A regular chain is a chain that is used for organization of chains and has no hook hence no control on packets. It may be used as a jump target for better organization.

Similar to a table, all operational activities can be done on a chain in addition to renaming a chain. Chains should be followed by a name and an open and a close curly bracket. They also come with a type, a hook, a priority, and a policy that must be defined when creating a chain as shown in the next figure.

```
table [<family>] <name> {

 chain <chain-name> {

   type <filter-type> hook <hook> priority <priority>; policy <policy>;

 } }
```

Chains Types

Type	Description
Filter	This is a standard chain type and supports all address families namely ARP, bridge, IP, IP6, and inet and hooks.
Route	It supports only IP and IPv6 address families and only output hook. If relevant parts of the IP header have changed, a new route lookup is performed.
Nat	It can perform Network Address Translation, and only supports IP and IPv6 address families. prerouting, input, output, postrouting hooks are also supported.

Chains Hooks

A Hook in a chain refers to a specific stage that a packet is being processed through a Linux kernel based on defined rules. These hooks are ingress, prerouting, input, forward, output, and postrouting and are explaind in detail in the next section. Prerouting, input, forward, output, and postrouting hook can also support IP, IPv6, and inet address families. To support arp address family, input, output hooks can be used while for netdev family, ingress hook should be used.

Type	Description
Prerouting	All packets entering a node are processed by this hook. It is invoked before the routing process and is used for early filtering or changing packet attributes that affect routing.
Input	This hook are executed after the routing decision. Packets delivered to a local system are processed by this hook.
Forward	This hook also happens after the routing decision. Packets that are not directed to the local machine are processed by this hook.
Output	This hook controls the packets that are originated from processes in a local machine.
Postrouting	This hook is used for the packets leaving a local system after the routing decision.

Type	Description
Ingress	(only available at the netdev family): Since Linux kernel 4.2, traffic can be filtered before layer 3 and way before prerouting, after the packets are passed up from a NIC driver.

Priority

Nftables requires you to specify a priority value when creating a base chain. You can specify integer values, but the newer versions of Nftables also define placeholder names for several discrete priority values analog to the mentioned enums in Netfilter. The following table lists those placeholder names12).

Name	Priority Value
raw	-300
mangle	-150
conntrack13)	-200
dstnat	-100
filter	0
security	50
srcnat	100

When creating a base chain, you can e.g. specify priority filter which translates into priority 0. The following example creates a table named myfilter in the ip address family (IPv4). It then creates two base chains named foo and bar, registering them with the Netfilter IPv4 hook input, but each with different priority.

```
nft create table ip myfilter
nft create chain ip myfilter foo {type filter hook input priority 0\;}
nft create chain ip myfilter bar {type filter hook input priority 50\;}

# alternatively you could create the same chains using named priority
values:
nft create chain ip myfilter foo {type filter hook input priority filter\;}
nft create chain ip myfilter bar {type filter hook input priority
security\;}
```

Policies

Chains have to have their policies by which packets are treated to be either dropped or accepted by default. These policy values can be **“accept”**, which is the default policy, or **“drop”**. Accept policy means that all the network packets based on their locations defined by the hook should be accepted by default whereas drop policy means that by default all network packets must be dropped based on their locations defined by the hook in a chain and then based on defined rules inside a chain will be accepted or otherwise.

Rules

Rules are the actions that control the incoming and outgoing packets based on the defined hooks in a chain. If a rule inside a chain matches with a packet based on the stage derived from their hooks, the packet is dropped or accepted. A rule is evaluated from left to right in a way that when the first statement matches, it continues with the next parts of a rule, but if not, the next rule will be evaluated. The structure of a rule includes matches and statements which is as follows:

```
<matches> <statements>

For example:

iifname “interface name” Policy: <accept or drop>
```

Matches

Matches are those filters that enable a rule to filter certain packets. Some important matches with their possible formats are briefly as follows:

```
Ip saddr <ip source address>
Ip daddr <ip destination address>
tcp / udp dport <destination port>
tcp / udp sport < source port>
tcp flags <flags>
ICMP type <type>
iifname <input interface name>
oifname <output interface name>
protocol <protocol>
```

Statements

A statement is the defined action performed once a packet matches a match(es) defined by a rule. Statements comprise of verdict, log, and counter statements.

Verdict statements

The verdict statement alters the control flow in the rule set and issues policy decisions for packets. The valid verdict statements are:

Statement	Description
accept	Accept the packet and stop the remaining rules evaluation.
drop	Drop the packet and stop the remaining rules evaluation.
queue	Queue the packet to userspace and stop the remaining rules evaluation.
continue	Continue the ruleset evaluation with the next rule.
jump <chain>	Continue at the first rule of <chain>. It will continue to evaluate the next rules to finally return to the last position or a return statement is issued.
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

Statement	Description
goto <chain>	Similar to jump, but after finishing the rules in <chain>, the evaluation will continue to evaluate the next chains instead of waiting for a return to the last chain.

Links

- wiki.nftables.org

From:
https://wiki.oscardegroot.nl/ - **HomeWiki**

Permanent link:
https://wiki.oscardegroot.nl/doku.php?id=networking:nftables&rev=1693731251

Last update: **2023/09/03 08:54**