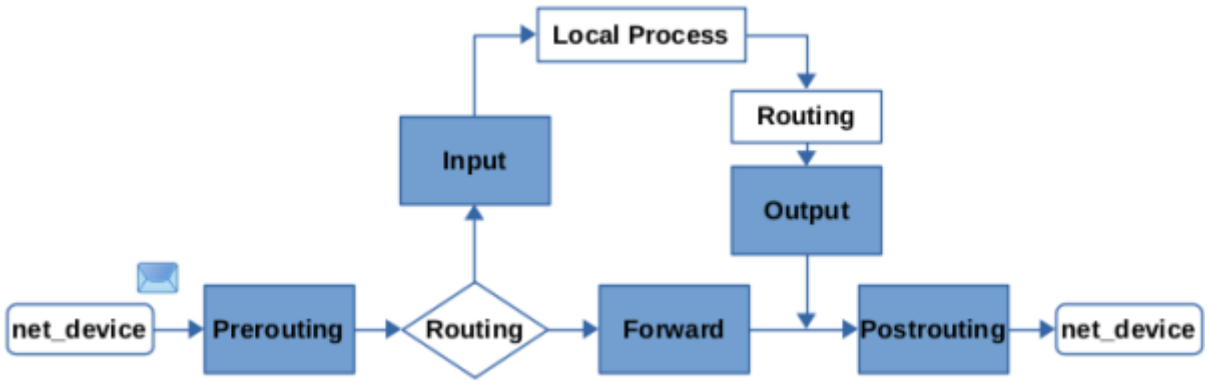
NFTables

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).

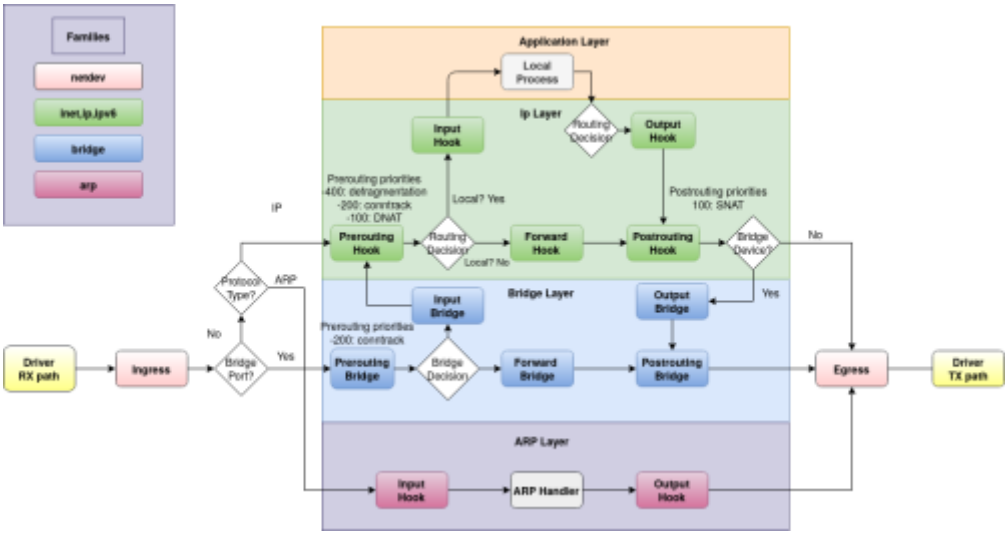
Address Families

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:

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:



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\;}
```

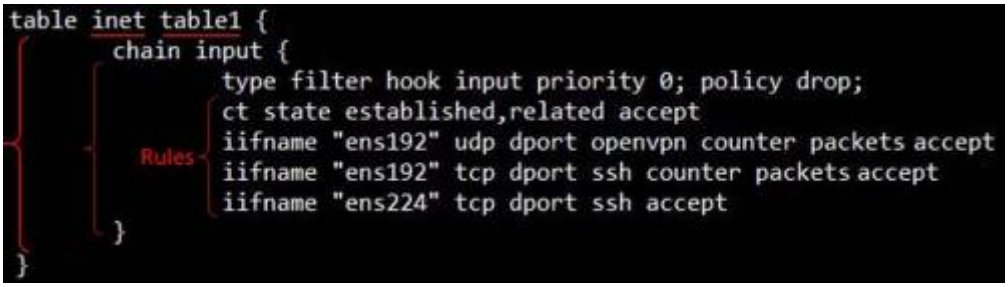
Variables

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.



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.

Chain chain-name { type <type> hook <hook> priority <priority> ; policy <policy> ; }

Links

- wiki.nftables.org