

Getting started with the steering

📅 26 Marzo 2026

👤 Simone Tiberi
✉ tiberi@epigenesys.com

Outline

- 1 What is the steering
- 2 Receive Side Scaling
- 3 Receive Packet Steering
- 4 Packet fanout
- 5 From user-space to kernel-space

Steering
Simone Tiberi


[What is the steering](#)
[Receive Side Scaling](#)
[Receive Packet Steering](#)
[Packet fanout](#)
AF_PACKET family
[From user-space to kernel-space](#)
socket
bind
setsockopt
NAPI data-path

Epigenesys s.r.l. 1 / 24

Steering
Simone Tiberi


[What is the steering](#)
[Receive Side Scaling](#)
[Receive Packet Steering](#)
[Packet fanout](#)
AF_PACKET family
[From user-space to kernel-space](#)
socket
bind
setsockopt
NAPI data-path

Epigenesys s.r.l. 2 / 24

What is the steering

With **steering** we refer to several different techniques that allows you to efficiently distribute traffic in order to optimize the analysis.

💡 Remark

These techniques can be applied at different depths in the receive data-path.

Outline

- 1 What is the steering
- 2 Receive Side Scaling**
- 3 Receive Packet Steering
- 4 Packet fanout
- 5 From user-space to kernel-space

Steering

Simone Tiberi



What is the steering

Receive Side Scaling

Receive Packet Steering

Packet fanout

AF_PACKET family

From user-space to kernel-space

socket

bind

setsockopt

NAPI data-path

Steering

Simone Tiberi



What is the steering

Receive Side Scaling

Receive Packet Steering

Packet fanout

AF_PACKET family

From user-space to kernel-space

socket

bind

setsockopt

NAPI data-path

Receive Side Scaling (1)

The earliest spot at which steering can be applied is inside the NIC (i.e., at [hardware level](#)).

At this stage, we talk about **RSS** (i.e., **R**ecieve **S**ide **S**caling).

Conditio sine qua non to have RSS enabled in a NIC is [multi-queue](#) support. To check if this condition is satisfied you can run the following command:

```
$ ethtool -l <iface>
```

Receive Side Scaling (2)

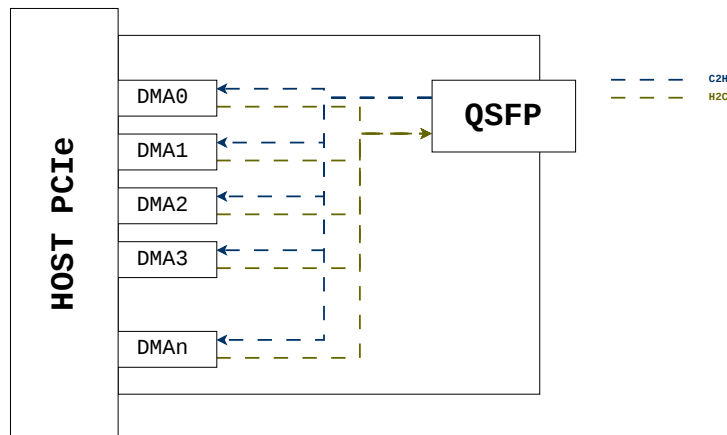
```
# ethtool -l enp3s0f1np1
Channel parameters for enp3s0f1np1:
Pre-set maximums:
RX:                n/a
TX:                n/a
Other:             n/a
Combined:          63
Current hardware settings:
RX:                n/a
TX:                n/a
Other:             n/a
Combined:          63
```



Receive Side Scaling (3)

A multi-queue NIC is characterized by having with multiple **descriptor queues** that maps 1:1 with DMA engines that allows:

- **C2H** (i.e., card-to-host) data migration
- **H2C** (i.e., host-to-card) data migration

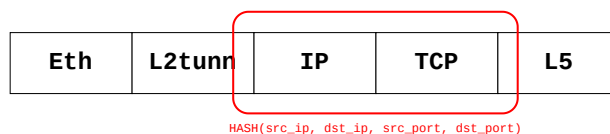


Receive Side Scaling (4)

When RSS is enabled, traffic distribution among queues is ruled by **flow key** evaluation (i.e., ~~no round-robin~~).

Remark

Most of the time we would like to have **symmetric** flow key, in order to *put in the same bucket* both requests and responses.



Receive Side Scaling (5)

Each time an RSS-enable NIC receives a packet:

- 1 it extracts useful fields for key computation
- 2 if the key has to be symmetric, it should shuffle them by inferring some form of partial ordering
- 3 it computes an hash (e.g. `toeplitz`) of the interesting fields
- 4 it exploits the n least significant bits of the hash to index an **indirection table**

💡 Remark

An indirection table is a table with 2^n entries, each of which contains the identifiers of the host DMA engines.



Receive Side Scaling (6)

Indirection table can be inspected/configured via `ethtool`

```
$ ethtool --show-rxfh-indir enp3s0f1np1
RX flow hash indirection table for enp3s0f1np1 with 63 RX
ring(s):
  0:      0      1      2      3      4      5      6      7
  8:      8      9     10     11     12     13     14     15
 16:     16     17     18     19     20     21     22     23
  .       .       .       .       .       .       .       .
  .       .       .       .       .       .       .       .
  .       .       .       .       .       .       .       .
 240:     51     52     53     54     55     56     57     58
 248:     59     60     61     62      0      1      2      3
RSS hash key:
2a:71:1d:3d:ac:71:f6:cf:b5:4f:40:f1:8c:ca:41:43:e1:43:a3
:99:e4:a7:89:ed:d3:d6:c0:ff:cc:db:a8:5c:f7:ec:4b:a1
:97:14:89:b2
RSS hash function:
toeplitz: on
xor: off
crc32: off
```

```
# ethtool --set-rxfh-indir enp3s0f1np1 equal 32
# #it distributes equally traffic among 32 cores
```



Outline

- 1 What is the steering
- 2 Receive Side Scaling
- 3 **Receive Packet Steering**
- 4 Packet fanout
- 5 From user-space to kernel-space

Steering
Simone Tiberi

What is the steering
Receive Side Scaling
Receive Packet Steering
Packet fanout
AF_PACKET family
From user-space to kernel-space
socket
bind
setsockopt
NAPI data-path
Epigenesys s.r.l. 11 / 24

Receive Packet Steering (1)

In case RSS is not available for the NIC, traffic analysis still remains *distributable* among DMA engines/CPU cores.

In fact, later on in the data-path (i.e., at [driver level](#)), Linux kernel provides an alternative approach for the distribution: the so called **RPS** (i.e., **R**eceive **P**acket **S**teering).

In this case, once the **flow key** is computed, the packet is directly placed in the CPU-*i* backlog queue.

Steering
Simone Tiberi

What is the steering
Receive Side Scaling
Receive Packet Steering
Packet fanout
AF_PACKET family
From user-space to kernel-space
socket
bind
setsockopt
NAPI data-path
Epigenesys s.r.l. 12 / 24

💡 Remark

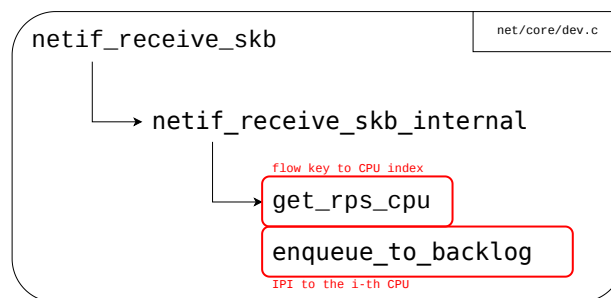
Flow key computation can still be provided by the hardware (i.e., *offload*) although it does not have the support for RSS.

Receive Packet Steering (2)

Pros vs Cons:

- 👍 it can be exploited by **any** NIC
- 👍 SW flow key computation algorithm can be easily expanded (i.e., via eBPF filters)
- 👎 it is based on IPI (i.e., it is **heavy**)

Kernel implementation:



Receive Packet Steering (3)

RPS has to be enabled at compile time via Kconfig (i.e., `CONFIG_RPS`).

By default, even if `CONFIG_RPS=y`, RPS it has to be configured in order to be actually exploited.

Configuration files are located in `sysfs`:

```
/sys/class/net/<dev>/queues/rx-<n>/rps_cpus
```



Outline

- 1 What is the steering
- 2 Receive Side Scaling
- 3 Receive Packet Steering
- 4 Packet fanout**
- 5 From user-space to kernel-space

Steering
Simone Tiberi

What is the steering
Receive Side Scaling
Receive Packet Steering
Packet fanout
AF_PACKET family
From user-space to kernel-space
socket
bind
setsockopt
NAPI data-path
Epigenesys s.r.l. 15 / 24

Packet fanout - AF_PACKET family

TCP-IP's L3/4 management is done in the `net` subsystem.

As you *should* know from user-space app development 😊 when you open a socket, you must provide a *family*.

The family of choice for packet analysis is **AF_PACKET**.

Steering
Simone Tiberi

What is the steering
Receive Side Scaling
Receive Packet Steering
Packet fanout
AF_PACKET family
From user-space to kernel-space
socket
bind
setsockopt
NAPI data-path
Epigenesys s.r.l. 16 / 24

Packet fanout - AF_PACKET family

You can easily understand that:

💡 Remark

The deeper the analysis, the slower the *per-packet* analysis.

For example:

- simply audit packets (like `tcpdump` does) ⇒ **fast**
- reconstruct L4 flows (like `wireshark` does) ⇒ **slower**
- deep inspect L5 payload ⇒ **even slower**

Steering

Simone Tiberi



What is the steering

Receive Side Scaling

Receive Packet Steering

Packet fanout

AF_PACKET family

From user-space to kernel-space

socket

bind

setsockopt

NAPI data-path

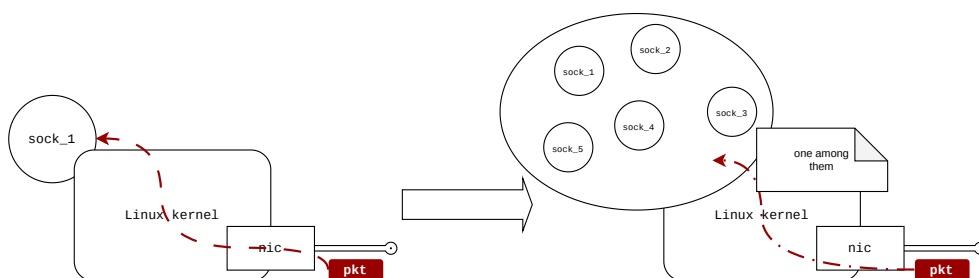
Epigenesys s.r.l. 17/24

Packet fanout - AF_PACKET family

Luckily, Linux Kernel provides you another way to distribute traffic: the **fanout** approach.

The fanout logic is based on the concept of **group** as unit of measurement of traffic reception

When a group is defined, each incoming packet is delivered to the group (not anymore to the specific socket).



Steering

Simone Tiberi



What is the steering

Receive Side Scaling

Receive Packet Steering

Packet fanout

AF_PACKET family

From user-space to kernel-space

socket

bind

setsockopt

NAPI data-path

Epigenesys s.r.l. 18/24

Packet fanout - AF_PACKET family

Linux kernel provides you several different way to distribute packets among sockets in the same group based on:

- flow key (i.e., `PACKET_FANOUT_HASH`)
- round-robin logic (i.e., `PACKET_FANOUT_LB`)
- CPU core that have managed the IRQ (i.e., `PACKET_FANOUT_CPU`)
- backup logic¹ (i.e., `PACKET_FANOUT_ROLLOVER`)
- pseudo-random generator (i.e., `PACKET_FANOUT_RND`)
- queue-mapping (i.e., `PACKET_FANOUT_QM`)
- cBPF filter (i.e., `PACKET_FANOUT_CBPF`)
- eBPF filter (i.e., `PACKET_FANOUT_EBPF`)

¹Once the current is backlogged the next is involved

Outline

- 1 What is the steering
- 2 Receive Side Scaling
- 3 Receive Packet Steering
- 4 Packet fanout
- 5 From user-space to kernel-space

Steering
Simone Tiberi



[What is the steering](#)
[Receive Side Scaling](#)
[Receive Packet Steering](#)
[Packet fanout](#)
AF_PACKET family
[From user-space to kernel-space](#)
socket
bind
setsockopt
NAPI data-path

Epigenesys s.r.l. 19 / 24

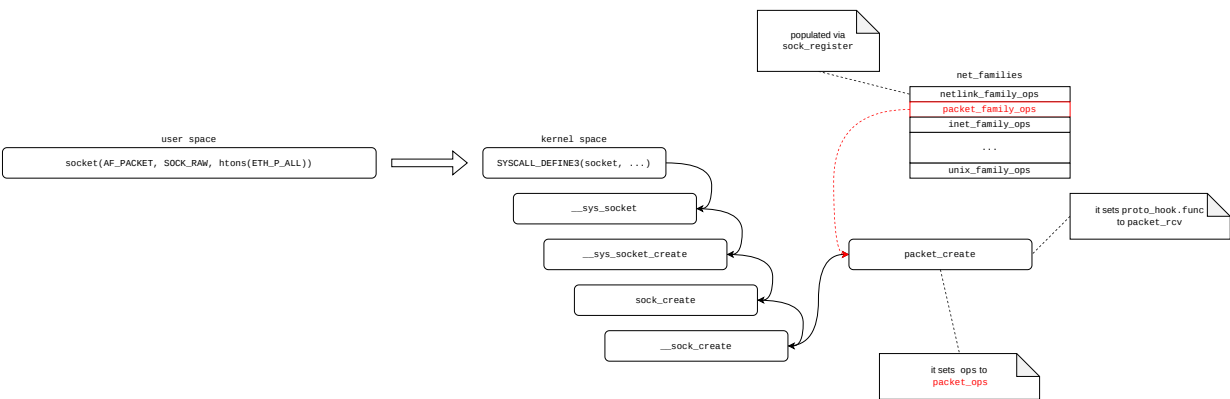
Steering
Simone Tiberi



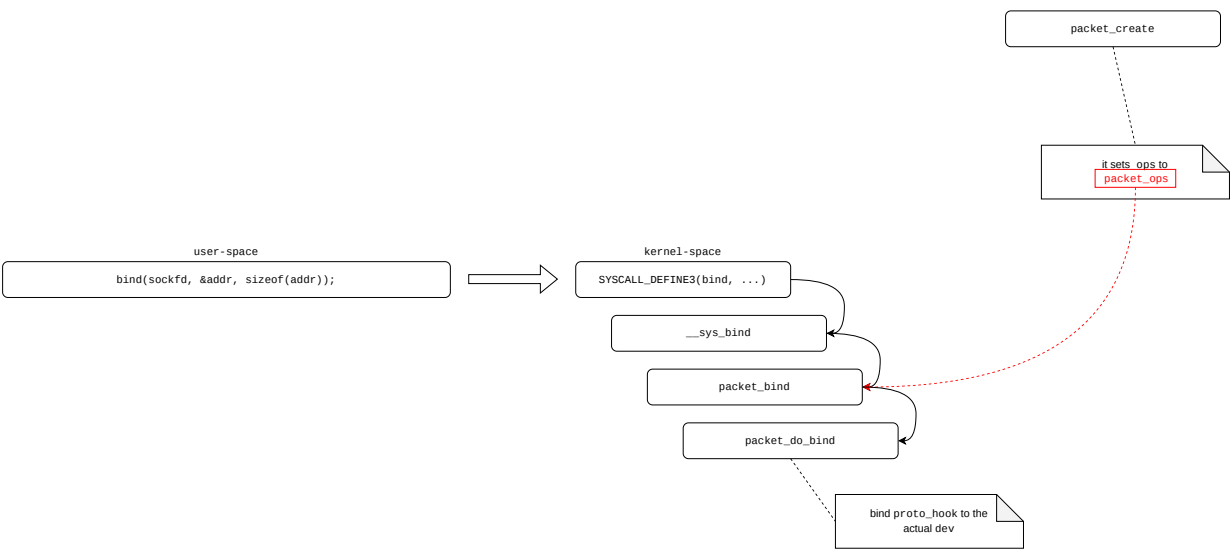
[What is the steering](#)
[Receive Side Scaling](#)
[Receive Packet Steering](#)
[Packet fanout](#)
AF_PACKET family
From user-space to kernel-space
socket
bind
setsockopt
NAPI data-path

Epigenesys s.r.l. 20 / 24

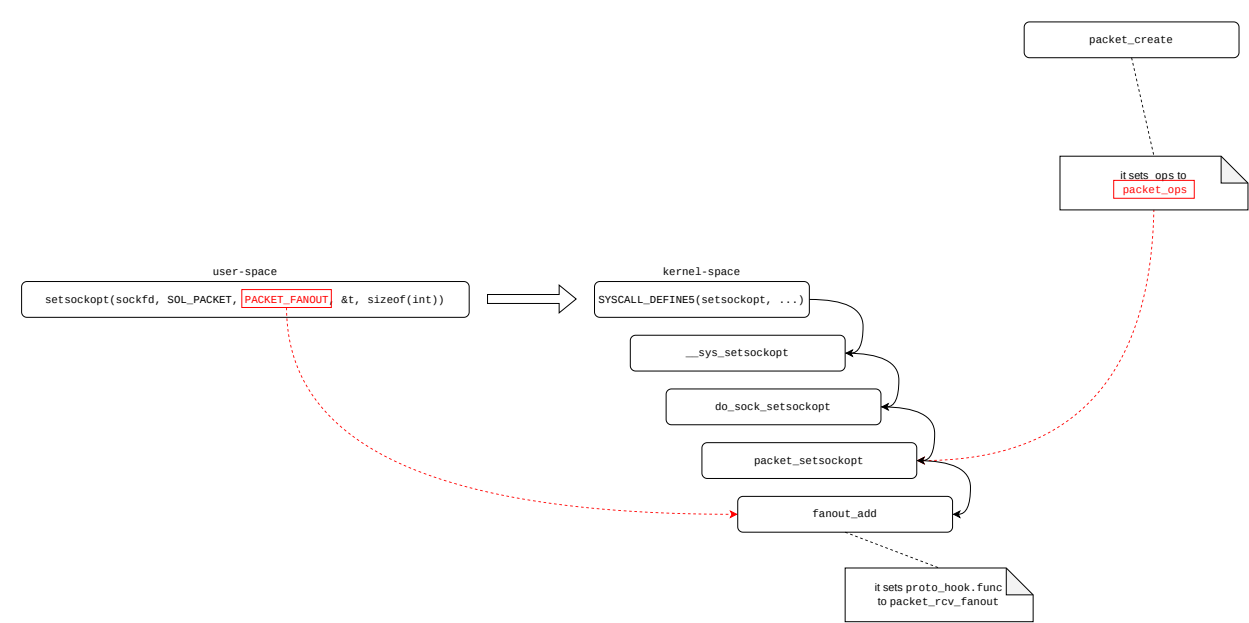
From user-space to kernel-space - socket



From user-space to kernel-space - bind



From user-space to kernel-space - setsockopt



From user-space to kernel-space - NAPI data-path

