



## Flow Steering

# Table of contents

Flow Steering Support

---

Flow Domains and Priorities

---

Ethtool

---

Accelerated Receive Flow Steering (aRFS)

---

Flow Steering Dump Tool

---

Flow steering is a new model which steers network flows based on flow specifications to specific QPs. Those flows can be either unicast or multicast network flows. In order to maintain flexibility, domains and priorities are used. Flow steering uses a methodology of flow attribute, which is a combination of L2-L4 flow specifications, a destination QP and a priority. Flow steering rules may be inserted either by using ethtool or by using InfiniBand verbs. The verbs abstraction uses different terminology from the flow attribute (`ibv_flow_attr`), defined by a combination of specifications (`struct ibv_flow_spec_*`).

## Flow Steering Support

All flow steering features are enabled in the supported adapter cards.

## Flow Domains and Priorities

Flow steering defines the concept of domain and priority. Each domain represents a user agent that can attach a flow. The domains are prioritized. A higher priority domain will always supersede a lower priority domain when their flow specifications overlap. Setting a lower priority value will result in a higher priority.

In addition to the domain, there is a priority within each of the domains. Each domain can have at most  $2^{12}$  priorities in accordance with its needs.

The following are the domains at a descending order of priority:

- **User Verbs** allows a user application QP to be attached to a specified flow when using `ibv_create_flow` and `ibv_destroy_flow` verbs
- `ibv_create_flow`

```
struct ibv_flow *ibv_create_flow(struct ibv_qp *qp, struct
ibv_flow_attr
*flow)
```

### Input parameters:

- `struct ibv_qp` - the attached QP.

- `struct ibv_flow_attr` - attaches the QP to the flow specified. The flow contains mandatory control parameters and optional L2, L3 and L4 headers. The optional headers are detected by setting the size and `num_of_specs` fields:

`struct ibv_flow_attr` can be followed by the optional flow headers structs:

```
struct ibv_flow_spec_eth
struct ibv_flow_spec_ipv4
struct ibv_flow_spec_tcp_udp
struct ibv_flow_spec_ipv6
```

For further information, please refer to the `ibv_create_flow` man page.

- `ibv_destroy_flow`

```
int ibv_destroy_flow(struct ibv_flow *flow_id)
```

#### **Input parameters:**

`ibv_destroy_flow` requires struct `ibv_flow` which is the return value of `ibv_create_flow` in case of success.

#### **Output parameters:**

Returns 0 on success, or the value of `errno` on failure.

For further information, please refer to the `ibv_destroy_flow` man page.

## **Ethtool**

Ethtool domain is used to attach an RX ring, specifically its QP to a specified flow. Please refer to the most recent ethtool man page for all the ways to specify a flow.

#### **Examples:**

- `ethtool -U eth5 flow-type ether dst 00:11:22:33:44:55 loc 5 action 2`

All packets that contain the above destination MAC address are to be steered into rx-ring 2 (its underlying QP), with priority 5 (within the ethtool domain)

- `ethtool -U eth5 flow-type tcp4 src-ip 1.2.3.4 dst-port 8888 loc 5 action 2`

All packets that contain the above destination IP address and source port are to be steered into rx- ring 2. When destination MAC is not given, the user's destination MAC is filled automatically.

- `ethtool -U eth5 flow-type ether dst 00:11:22:33:44:55 vlan 45 m 0xf000 loc 5 action 2`

All packets that contain the above destination MAC address and specific VLAN are steered into ring 2. Please pay attention to the VLAN's mask 0xf000. It is required in order to add such a rule.

- `ethtool -u eth5`

Shows all of ethtool's steering rule

When configuring two rules with the same priority, the second rule will overwrite the first one, so this ethtool interface is effectively a table. Inserting Flow Steering rules in the kernel requires support from both the ethtool in the user space and in kernel (v2.6.28).

## Accelerated Receive Flow Steering (aRFS)

Receive Flow Steering (RFS) and Accelerated Receive Flow Steering (aRFS) are kernel features currently available in most distributions. For RFS, packets are forwarded based on the location of the application consuming the packet. aRFS boosts the speed of RFS by adding support for the hardware. By using aRFS (unlike RFS), the packets are directed to a CPU that is local to the thread running the application.

aRFS is an in-kernel-logic responsible for load balancing between CPUs by attaching flows to CPUs that are used by flow's owner applications. This domain allows the aRFS mechanism to use the flow steering infrastructure to support the aRFS logic by implementing the `ndo_rx_flow_steer`, which, in turn, calls the underlying flow steering mechanism with the aRFS domain.

### To configure RFS:



Configure the RFS flow table entries (globally and per core).

**Note:** The functionality remains disabled until explicitly configured (by default it is 0).

- The number of entries in the global flow table is set as follows:

**Note**

```
/proc/sys/net/core/rps_sock_flow_entries
```

- The number of entries in the per-queue flow table are set as follows:

**Note**

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

**Example:**

```
# echo 32768 > /proc/sys/net/core/rps_sock_flow_entries
# NUM_CHANNELS=`ethtool -l ens6 | grep "Combined:" | tail -1 | awk
'{print $2}'`
# for f in `seq 0 $((NUM_CHANNELS-1))`; do echo 32768 >
/sys/class/net/ens6/queues/rx-$f/rps_flow_cnt; done
```

**To Configure aRFS:**



The aRFS feature requires explicit configuration in order to enable it. Enabling the aRFS requires enabling the 'ntuple' flag via the ethtool.

For example, to enable ntuple for eth0, run:

```
ethtool -K eth0 ntuple on
```

aRFS requires the kernel to be compiled with the `CONFIG_RFS_ACCEL` option. This option is available in kernels 2.6.39 and above. Furthermore, aRFS requires Device Managed Flow Steering support.

### Note

RFS cannot function if LRO is enabled. LRO can be disabled via `ethtool`.

## Flow Steering Dump Tool

The `mlx_fs_dump` is a python tool that prints the steering rules in a readable manner. Python v2.7 or above, as well as `pip`, `anytree` and `termcolor` libraries are required to be installed on the host.

### Running example:

```
./ofed_scripts/utils/mlx_fs_dump -d /dev/mst/mt4115_pciconf0
FT: 9 (level: 0x18, type: NIC_RX)
+-- FG: 0x15 (MISC)
    |-- FTE: 0x0 (FWD) to (TIR:0x7e) out.ethtype:IPv4
out.ip_prot:UDP out.udp_dport:0x140
    +-- FTE: 0x1 (FWD) to (TIR:0x7e) out.ethtype:IPv4
out.ip_prot:UDP out.udp_dport:0x13f
...
```

For further information on the `mlx_fs_dump` tool, please refer to [mlx\\_fs\\_dump Community post](#).

**Notice**  
This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality. NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete. NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document. NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk. NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs. No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA. Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices. THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product. **Trademarks** NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

© Copyright 2025, NVIDIA. PDF Generated on 05/05/2025