Netronome Flow Processor (NFP) Kernel Drivers

Copyright (c) 2019, Netronome Systems, Inc.

Contents

Overview

This driver supports Netronome's line of Flow Processor devices, including the NFP4000, NFP5000, and NFP6000 models, which are also incorporated in the company's family of Agilio SmartNICs. The SR-IOV physical and virtual functions for these devices are supported by the driver.

Acquiring Firmware

The NFP4000 and NFP6000 devices require application specific firmware to function. Application firmware can be located either on the host file system or in the device flash (if supported by management firmware).

Firmware files on the host filesystem contain card type (*AMDA-** string), media config etc. They should be placed in */lib/firmware/netronome* directory to load firmware from the host file system.

Firmware for basic NIC operation is available in the upstream *linux-firmware.git* repository.

Firmware in NVRAM

Recent versions of management firmware supports loading application firmware from flash when the host driver gets probed. The firmware loading policy configuration may be used to configure this feature appropriately.

Devlink or ethtool can be used to update the application firmware on the device flash by providing the appropriate *nic_AMDA*.nffw* file to the respective command. Users need to take care to write the correct firmware image for the card and media configuration to flash.

Available storage space in flash depends on the card being used.

Dealing with multiple projects

NFP hardware is fully programmable therefore there can be different firmware images targeting different applications.

When using application firmware from host, we recommend placing actual firmware files in application-named subdirectories in */lib/firmware/netronome* and linking the desired files, e.g.:

```
$ tree /lib/firmware/netronome/
/lib/firmware/netronome/
â”œâ”€â”€ bpf
â”‚Â Â  â”œâ”€â”€ nic_AMDA0081-0001_1x40.nffw
â”‚Â Â  â””â”€â”€ nic_AMDA0081-0001_4x10.nffw
â”œâ”€â”€ flower
â”‚Â Â  â”œâ”€â”€ nic_AMDA0081-0001_1x40.nffw
â”‚Â Â  â””â”€â”€ nic_AMDA0081-0001_4x10.nffw
â”œâ”€â”€ nic
â”‚Â Â  â”œâ”€â”€ nic_AMDA0081-0001_1x40.nffw
â”‚Â Â  â””â”€â”€ nic_AMDA0081-0001_4x10.nffw
â”œâ”€â”€ nic_AMDA0081-0001_1x40.nffw -> bpf/nic_AMDA0081-0001_1x40.nffw
â””â”€â”€ nic_AMDA0081-0001_4x10.nffw -> bpf/nic_AMDA0081-0001_4x10.nffw

3 directories, 8 files
```

You may need to use hard instead of symbolic links on distributions which use old *mkinitrd* command instead of *dracut* (e.g. Ubuntu).

After changing firmware files you may need to regenerate the initramfs image. Initramfs contains drivers and firmware files your system may need to boot. Refer to the documentation of your distribution to find out how to update initramfs. Good indication of stale initramfs is system loading wrong driver or firmware on boot, but when driver is later reloaded manually everything works correctly.

Selecting firmware per device

Most commonly all cards on the system use the same type of firmware. If you want to load specific firmware image for a specific card, you can use either the PCI bus address or serial number. Driver will print which files it's looking for when it recognizes a NFP device:

```
nfp: Looking for firmware file in order of priority:
nfp:  netronome/serial-00-12-34-aa-bb-cc-10-ff.nffw: not found
nfp:  netronome/pci-0000:02:00.0.nffw: not found
nfp:  netronome/nic_AMDA0081-0001_1x40.nffw: found, loading...
```

In this case if file (or link) called *serial-00-12-34-aa-bb-5d-10-ff.nffw* or *pci-0000:02:00.0.nffw* is present in */lib/firmware/netronome* this firmware file will take precedence over *nic_AMDA** files.

Note that *serial-** and *pci-** files are **not** automatically included in initramfs, you will have to refer to documentation of appropriate tools to find out how to include them.

Firmware loading policy

Firmware loading policy is controlled via three HWinfo parameters stored as key value pairs in the device flash:

app_fw_from_flash
: Defines which firmware should take precedence, 'Disk' (0), 'Flash' (1) or the 'Preferred' (2) firmware. When 'Preferred' is selected, the management firmware makes the decision over which firmware will be loaded by comparing versions of the flash firmware and the host supplied firmware. This variable is configurable using the 'fw_load_policy' devlink parameter.

abi_drv_reset
: Defines if the driver should reset the firmware when the driver is probed, either 'Disk' (0) if firmware was found on disk, 'Always' (1) reset or 'Never' (2) reset. Note that the device is always reset on driver unload if firmware was loaded when the driver was probed. This variable is configurable using the 'reset_dev_on_drv_probe' devlink parameter.

abi_drv_load_ifc
: Defines a list of PF devices allowed to load FW on the device. This variable is not currently user configurable.

Statistics

Following device statistics are available through the `ethtool -S` interface:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\device_drivers\ethernet\netronome\[linux-master][Documentation][networking][device_drivers][ethernet][netronome]nfp.rst, line 140)

Unknown directive type "flat-table".

```
.. flat-table:: NFP device statistics
   :header-rows: 1
   :widths: 3 1 11

   * - Name
     - ID
     - Meaning

   * - dev_rx_discards
     - 1
     - Packet can be discarded on the RX path for one of the following reasons:

         * The NIC is not in promisc mode, and the destination MAC address
           doesn't match the interfaces' MAC address.
         * The received packet is larger than the max buffer size on the host.
           I.e. it exceeds the Layer 3 MRU.
         * There is no freelist descriptor available on the host for the packet.
           It is likely that the NIC couldn't cache one in time.
         * A BPF program discarded the packet.
         * The datapath drop action was executed.
         * The MAC discarded the packet due to lack of ingress buffer space
           on the NIC.

   * - dev_rx_errors
     - 2
     - A packet can be counted (and dropped) as RX error for the following
       reasons:

         * A problem with the VEB lookup (only when SR-IOV is used).
         * A physical layer problem that causes Ethernet errors, like FCS or
           alignment errors. The cause is usually faulty cables or SFPs.

   * - dev_rx_bytes
     - 3
     - Total number of bytes received.

   * - dev_rx_uc_bytes
     - 4
     - Unicast bytes received.

   * - dev_rx_mc_bytes
```

```
  - 5
  - Multicast bytes received.

* - dev_rx_bc_bytes
  - 6
  - Broadcast bytes received.

* - dev_rx_pkts
  - 7
  - Total number of packets received.

* - dev_rx_mc_pkts
  - 8
  - Multicast packets received.

* - dev_rx_bc_pkts
  - 9
  - Broadcast packets received.

* - dev_tx_discards
  - 10
  - A packet can be discarded in the TX direction if the MAC is
    being flow controlled and the NIC runs out of TX queue space.

* - dev_tx_errors
  - 11
  - A packet can be counted as TX error (and dropped) for one for the
    following reasons:

    * The packet is an LSO segment, but the Layer 3 or Layer 4 offset
      could not be determined. Therefore LSO could not continue.
    * An invalid packet descriptor was received over PCIe.
    * The packet Layer 3 length exceeds the device MTU.
    * An error on the MAC/physical layer. Usually due to faulty cables or
      SFPs.
    * A CTM buffer could not be allocated.
    * The packet offset was incorrect and could not be fixed by the NIC.

* - dev_tx_bytes
  - 12
  - Total number of bytes transmitted.

* - dev_tx_uc_bytes
  - 13
  - Unicast bytes transmitted.

* - dev_tx_mc_bytes
  - 14
  - Multicast bytes transmitted.

* - dev_tx_bc_bytes
  - 15
  - Broadcast bytes transmitted.

* - dev_tx_pkts
  - 16
  - Total number of packets transmitted.

* - dev_tx_mc_pkts
  - 17
  - Multicast packets transmitted.

* - dev_tx_bc_pkts
  - 18
  - Broadcast packets transmitted.
```

Note that statistics unknown to the driver will be displayed as `dev_unknown_stat$ID`, where `$ID` refers to the second column above.