FW centric devices, NIC customization
Firmware Centric Device

Configurable Firmware

• Modern devices are complex
  • More functionality <> More HW units and engines
  • More Processing power <> More capabilities
  • More NVM capacity <> More Configuration options!

• Modern Device <> FW centric device
• Some Modern Devices have ARM cores and can run entire Linux OS.
• Even some early NICs had MIPs cores running Linux OS
Customization Nightmare

Configurable Firmware

Multitude of features
• Multi port
• Multi function
• E-switch
• Virtualization
• Traffic shaping and rate limiting
• Multi protocol/stack (ethernet, rdma, vdpa, virtio, nvme, etc..)
• Standard offloads (Checksum, GRO, TSO, etc..)
• Modes of operation: eBPF, XDP, XSK, HDS, RDMA.
• Crypto offloads engines (TLS, IPSec, MACsec)
• TC offloads, tunneling and full pipeline offloads.
• Vendor specific optimizations

• These features simply don’t work in harmony magically out of the box.
• User specific demands per feature
ConnecTX (4 and later)

Some Stats

- 500K Lines of “core” code.
- ~600 NV config parameters
- Forward compatible devices
  - FW/driver backward compatible

![FW LOC and NV params](chart.png)
Highly configurable

Yet Not Flexible

• Sadly, although highly configurable, these devices aren’t as flexible.

• Human intervention is required, and recommended :).  
• Exact combination of features to boot with
• Most efficient configuration
• Vendor specific optimizations and parameters.  
  • ConnectX device has ~600 NV Firmware parameters  
  • ~25% growth year over year.

• standardization can only go so far.
Vendor Specific Toolbox :'(

I don't like them too!

• To address some of the issues vendors had to extend there basic Firmware and NVM tools

• Require direct user space access to PCI
• Require proprietary kernel modules and long list of toolchain

• Not welcome in production
• Long turnaround cycles
• Secure boot and secure kernel, no solution!

Real life stories :
- How do I increase your HW GRO timeout ?
- Booting to other OS to make configuration changes
- Personal Laptop i2c on production machines
- Forgetting a debug fw running in production
- Down time due to bad custom fw, just to enable a simple flag!
Types of dials and knobs

- Functionality <> enable/disable/select.
- Performance <> parameters values, and rangers.
- Verbosity/Debugability (RAS feature) <> trigger, monitor, capture, report

Categorization:

1) non-volatile device configuration and firmware update - static and preserved across reboots
2) Volatile device global firmware configuration – runtime.
3) Volatile per-function firmware configuration (PF/VF/SF) – runtime.
4) RAS features for FW - capture crash/fault data, read back logs, trigger device diagnostic modes, report device diagnostic data, device attestation
upstream APIs

Mainstream utilities (Devlink)


- devlink goal was always to provide a healthy mix of standards based multi-vendor APIs side by side

- Devlink [vendor specific] params
- Devlink health
- Devlink resource
- Devlink port functions *
(1 & 2) Non-Volatile and Volatile NIC customization

Devlink parameters


• SET:
  • devlink dev param set DEV name PARAMETER value VALUE cmode { runtime | driverinit | permanent }

• READ:
  • devlink dev param show

• Reload driver:
  • devlink dev reload pci/0000:01:00.0

• Vendor Specific params are marked
  • driver-specific

• Problems:
  • very hard to agree what’s driver-specific what’s generic
  • Almost everything starts as driver-specific
  • Only a small subset of NIC FW customization exist today in devlink!
(3) Volatile Per function configuration (PF/VF/SF)

**Devlink port functions**

**Virtualization mess**

- Virtualization environments where physical, virtual and synthetic functions are spawned on demand
- VFs are created via sriov sub-system
- SFs are managed by eswitch
- These functions, are represented via function ports (port flavor) in devlink
- VF/SF customization required and life cycle management
- High scale HW resource limitations
- Missing devlink params infrastructure per function port
RAS feature for FW

Trigger, monitor, capture, report

• Devlink provides many mechanisms but still the picture is incomplete!

• Devlink DPIPE – very specific to the offloading pipeline!
• Devlink health – very specific to driver faulty flows, although very rich in terms of RAS
• Devlink region – dump the whole configuration space or pre-defined registers
• Devlink resource – control limits of driver-registered HW resources

• Not really utilized or implemented by many drivers
• All of the above is READ only, the NIC debug state is immutable, without any external tools!
• Missing set-ability to enable disable hw dumps tracing and diagnostic mechanism
Vendor extensions

Dynamic approach

• High configurability issues isn't specific to NICs and already had been addressed in many products

• Ethtool module eeprom and register dump* vendor specific parsing

• Nvme-cli : https://man.archlinux.org/man/nvme.1
  • Nvme-fw-log(1), nvme-smart-log(1), nvme-fw-activate(1), nvme-fw-commit(1)
  • Plugins/Vendor extension commands:
    • nvme-intel-xyz

• Kubernetes: Container Network Interface (CNI) Specification plugins
  • plugin is a program that applies a specified network configuration.
  • https://github.com/containernetworking/cni/blob/spec-v1.0.0/SPEC.md#summary

• OpenStack
  • “OpenStack SDK is implemented as an extensible core, upon which vendor extensions can be plugged in”

• Libvirt ?
To summarize

- NIC attestation is still immature even with all

- Embargo on devlink parameters should be lifted! It’s an unpopulated wonderland!

- Kernel Admins, developers, testers, and curious HW explorers will welcome such ability to easily customize the HW!
Questions?