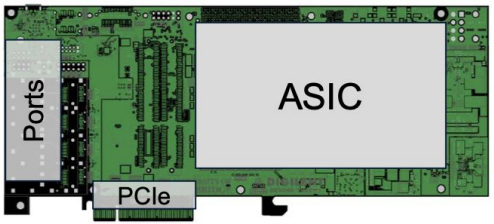
SmartNIC - the next leap in networking

Marcelo Caggiani Luizelli (UNIPAMPA)
Francisco Vogt (UNICAMP)
Guilherme Mendes Vieira de Matos (UFSCar)
Weverton Cordeiro (UFRGS)
Alberto Egon Schaeffer Filho (UFRGS)
Marcos Schwarz (RNP)
Fábio Luciano Verdi (UFSCar)
Gergely Pongracz (Ericsson Research)
Gianni Antichi (Politecnico di Milano)
Christian Esteve Rothenberg (UNICAMP)

@ IEEE NetSoft 2024

SmartNICs



Agenda

- Introduction
- The Need for SmartNICs
- SmartNIC Architecture
- Hands-on #1
- Break
- Commercial & Market Perspectives
- Performance & Benchmarking
- Hands-on #2, #3
- Challenges and Future Trends
- Conclusion & Closing Remarks

Introduction

- Historical Context and Evolution
- Definition and Explanation of SmartNICs
- Key Components and Functionalities

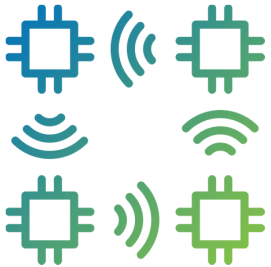
Understanding SmartNICs

Historical Context and Evolution

What are NICs designed for?

- A Network Interface Card (NIC) connects one or more Ethernet networks.

- It is responsible for: receiving incoming network packets and transmitting them to the host bus and/or to the wire (and vice-versa).



Understanding SmartNICs

Historical Context and Evolution

Evolution of NICs:

- **1980s:** simple devices designed to connect computers to Ethernet networks and transmit data packets. Most NICs were based on coaxial cables and 10 Mbps Ethernet network technologies.
- **1990s:** NICs evolved to support faster Ethernet technologies, such as 100 Mbps and 1 Gbps. Wireless NICs also paved the way for mobile networks.
- **2000s:** With the rise of server virtualization and the need for increased network performance in data centers. Hence, the need to support virtualization features like SR-IOV became common.
- **2010s:** With the growth of cloud computing, data center demands became greater - i.e., 10 Gbps, 25 Gbps, and 40 Gbps became common standards.
- **2020s:** Nowadays, NICs support 200 Gbps and beyond. Emerging technologies like SmartNICs integrate programmable processing capabilities to enhance performance and flexibility. In 2024 2024, the IEEE P802.3df Task Force approved the 800 Gigabit Ethernet (800G, 800GbE) standard.

Understanding SmartNICs

Definition and Explanation of SmartNICs

What are SmartNICs?

- The term "**SmartNIC**" suggests that it goes beyond the capabilities of standard NIC (Network Interface Card) devices, as it entails more than just added functionality.
- The "**Smart**" in SmartNIC implies a level of intelligence, enabling it to perform complex tasks efficiently.
- SmartNIC can also be referred as **DPU** (Data Processing Unit) or **IPU** (Infrastructure Processing Unit).
  - However, DPUs can be designed to be a SmartNIC or a standalone appliance

ET3212A DPU appliance

ET3424A DPU appliance

Understanding SmartNICs

Definition and Explanation of SmartNICs

What are the benefits?

- It is a specialized network card designed for **hardware acceleration** of **network / infrastructure tasks.**
  - For example: DPI, compression engine, etc

- It optimizes CPU resources, achieves high network speeds, and enhances efficiency.

Understanding SmartNICs

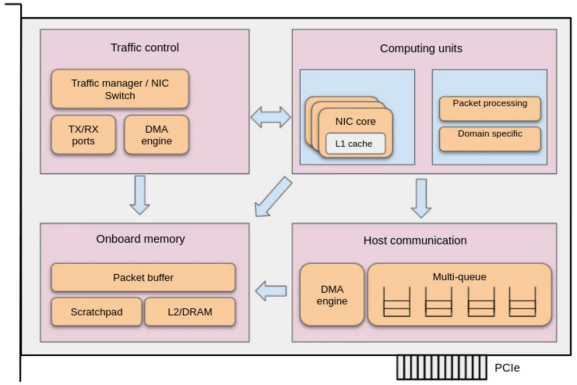
Key Components and Functionalities

The computational resources can be composed of **one or more** of the following categories:

- Classical x86 CPUs like Arm cores
- Purpose-built cores
- Networking processing units (NPUs)
- Field Programmable Gate Arrays (FPGAs)

These blocks are (mostly) highly specialized and run communication jobs more efficiently than CPUs.

Some blocks are programmable according to changing needs and keep up with network protocols as they evolve.



SmartNIC architecture

https://blogs.nvidia.com/blog/what-is-a-smartnic/
https://semiengineering.com/the-rise-of-smartnics/

The need for SmartNICs

- Challenges in Modern Network Environments
- Scalability, Latency and Bandwidth Demands
- Use Cases and Industry Benefiting from SmartNICs

The Need for SmartNICs

Challenges in Modern Network Environments

Considerations to integrate SmartNICs in existing network infrastructure, we can think about this topics:

- **High connectivity** - SmartNICs are best suited for big enterprises that support large bandwidths to handle high network traffic. These organizations have high-speed networks of more than 100 Gbps.

- **Ultra low latency** - Networks with requirements to perform high-speed computing use InfiniBand smartNIC infrastructure. In such networks, remote direct memory access (RDMA) enables direct access over another computer's memory to provide ultra low latency due to the noninvolvement of the kernel.

https://www.techtarget.com/searchnetworking/tip/SmartNICs-and-the-need-for-evolving-network-infrastructure#:~:text=A%20smartNIC%20is%20an%20intelligent,server%20at%20multiple%20networking%20layers.

https://www.techtarget.com/searchnetworking/tip/SmartNICs-and-the-need-for-evolving-network-infrastructure#:~:text=A%20smartNIC%20is%20an%20intelligent,server%20at%20multiple%20networking%20layers.

The Need for SmartNICs

Challenges in Modern Network Environments

- **Complex network architecture**
  - SmartNICs are applicable on the host side for large-scale cloud providers.
  - Network architectures, such as software-defined networking (SDN), network functions virtualization (NFV), enable smartNIC implementation for hardware acceleration.
- **Network switch compatibility**
  - Each SmartNIC vendor offers unique features, compatibility, reconfigurability
  - Some SmartNICs might not be compatible with network infrastructure
  - SmartNIC for existing network switches depends on interfaces, protocols, programmability, configuration and use cases.

https://www.techtarget.com/searchnetworking/tip/SmartNICs-and-the-need-for-evolving-network-infrastructure#:~:text=A%20smartNIC%20is%20an%20intelligent,server%20at%20multiple%20networking%20layers.

The Need for SmartNICs

Challenges in Modern Network Environments

- **SDN integration -** Integrating a SmartNIC infrastructure in SDN networks can offload SDN-centralized controller tasks for effective management of the network.

- **NFV capabilities (Network Functions Virtualization)** - NFVs can leverage a SmartNIC infrastructure as multiple VMs operate within a shared physical infrastructure, eliminating the necessity for dedicated hardware.
    - Examples of NFV acceleration encompass: Virtual eXtensible LAN (VX-LAN), micro segmentation, and load balancing.

https://www.techtarget.com/searchnetworking/tip/SmartNICs-and-the-need-for-evolving-network-infrastructure#:~:text=A%20smartNIC%20is%20an%20intelligent,server%20at%20multiple%20networking%20layers.

The Need for SmartNICs

Use cases and Industry Benefiting from SmartNICs

Use cases:

- **Data Centers and Cloud Computing**: SmartNICs offload networking tasks from the host CPU, reducing latency. Example: virtualized environments, such as virtual switch processing.

- **Telecommunications**: SmartNICs support the efficient processing of network functions and protocols. They play a role in accelerating data transfer, improving quality of service (QoS), and handling encryption tasks in secure communication networks.

- **Finance and High-Frequency Trading**: The low-latency and high-throughput capabilities of SmartNICs contribute to faster data processing and more responsive trading systems.

The Need for SmartNICs

Use cases and Industry Benefiting from SmartNICs

Use cases:

- **Network Security and Cybersecurity**: SmartNICs with security offload capabilities are employed in network security applications. Example: encryption, decryption, and packet inspection.
- **AI and Machine Learning**: SmartNICs play a role in accelerating AI and machine learning workloads. By offloading certain processing tasks from the CPU, SmartNICs contribute to faster model training and inference in distributed computing environments.
- **Healthcare**: SmartNICs assist in managing and processing large volumes of medical data efficiently. Example: medical images, patient records, etc.
- **Government and Defense**: SmartNICs are used in government and defense applications to ensure secure and efficient communication, accelerating critical network functions

The Need for SmartNICs

Use cases and Industry Benefiting from SmartNICs

Use cases:

- **Content Delivery Networks (CDNs)**: CDNs leverage SmartNICs to improve the delivery of content: (i) reducing latency, and (ii) enhancing content distribution performance.
- **Online Gaming**: SmartNICs contribute to a better online gaming experience by reducing latency and improving the responsiveness of gaming servers. They assist in offloading networking tasks, allowing CPUs to focus on game-related computations.
- **Enterprise Networking**: Enterprises benefit from SmartNICs in improving the overall efficiency of their networks. SmartNICs enhance data transfer rates, reduce latency, and contribute to better overall network performance in enterprise environments.
- **Edge Computing**: In edge computing scenarios, SmartNICs can process and filter data locally, reducing the need to send large volumes of data to centralized cloud servers. This is especially beneficial for applications requiring real-time responsiveness.

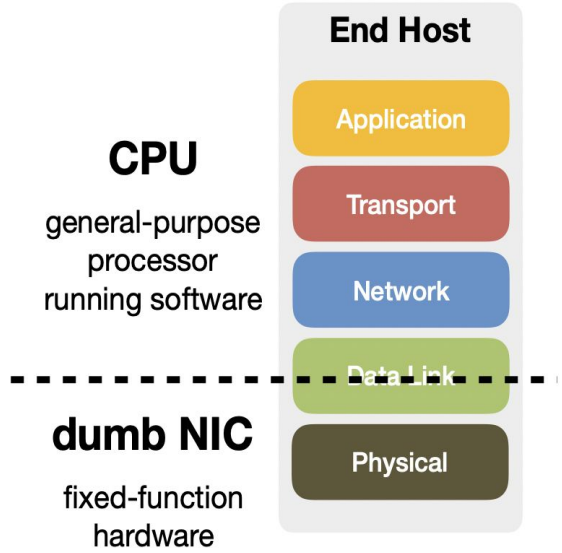
SmartNIC Architecture

- Functional blocks
- Comparison with Traditional NICs
- (NICs + CPUs) vs (SmartNICs)
- Types of implementation options
- Representative architectures
- Portable Network Architecture (PNA)

SmartNIC Architecture

Comparison with Traditional NICs

Traditional NICs



A problem known to exist is the lack of programmability/flexibility in regular NIC hardware which is built for a number of specific predefined use cases and at heart still serves the functionality as an interface from the host CPU to the network. This leaves much of the workload to the CPU.

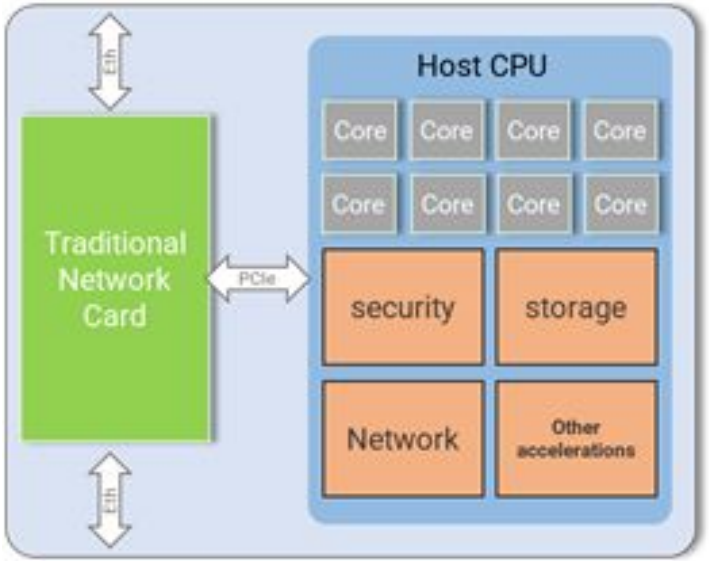
https://www.smallake.kr/wp-content/uploads/2021/11/NET-2021-05-1_05.pdf

https://www.cs.cornell.edu/~mt822/docs/smartNICs_lecture.pdf

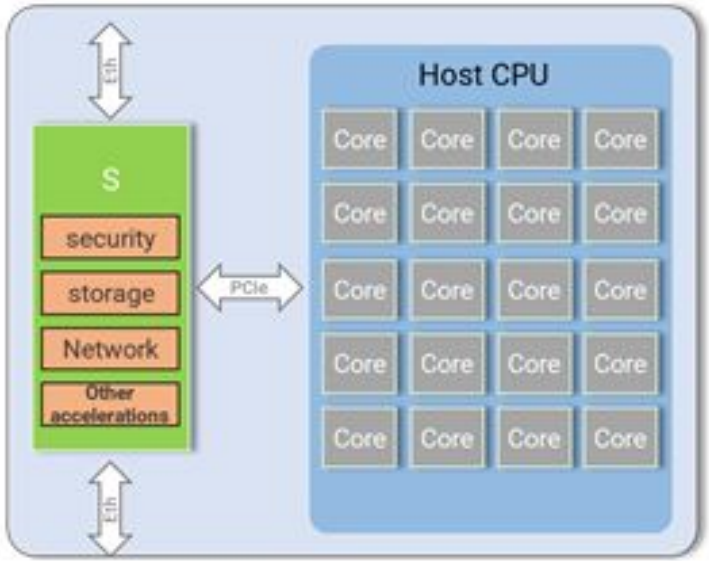
SmartNIC Architecture

Comparison with Traditional NICs

NIC + CPU



SmartNIC + CPU



Asterfusion`s SmartNIC Character and Architecture: https://cloudswit.ch/blogs/asterfusion-arm-based-dpu-smartnic-solution/

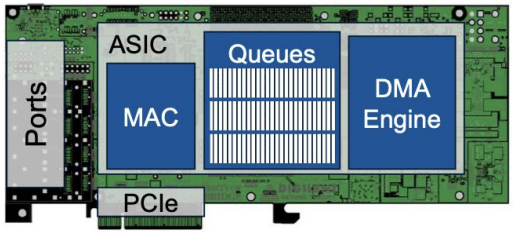
SmartNIC Architecture

Comparison with Traditional NICs

Basically…

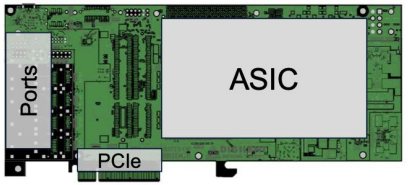
NICs

- Get packets from the network to the host
- Get packets from the host to the network
- DMA – manage getting packets to/from host over the interconnect
- Manage queues
- … and a few offloading by default (checksum, TCP/UDP offloading, etc)



SmartNICs

- Virtualization
- OVS offloading
- Tunneling offloading
- Storage offloading (e.g. Ceph)
- **Programmable data planes**
- Data transport acceleration
- NVMe, RoCE, DPDK…
- Security IPSec, MACSec
- Network function virtualization (NFV)
- **Application-specific acceleration**
- Deep-packet inspection (L4-L7)



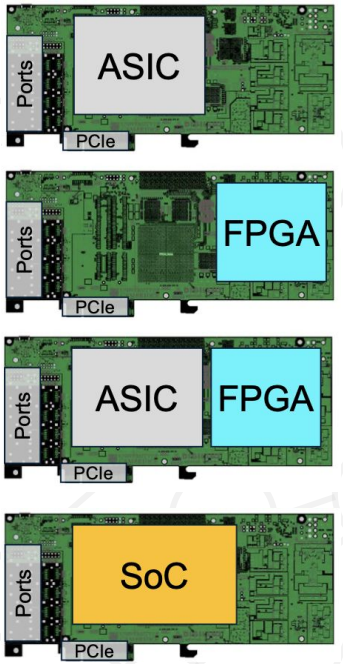
https://www.cl.cam.ac.uk/teaching/1819/P51/lecture2.pdf

SmartNIC Architecture

Types of implementation options

A SmartNIC can offload functions like networking, storage, and security. You can choose from:

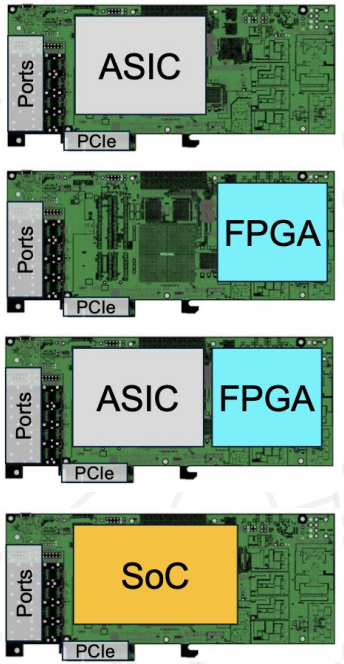
- ASIC based (e.g., Intel, Netronome)
- FPGA based (e.g., Xilinx, Intel, Achronix)
- SoC (e.g., ASIC w/ CPU) based (e.g., Nvidia/Mellanox)
- Combination (ASIC + FPGA based (inline or not, e.g., Intel/Xilinx)



SmartNIC Architecture

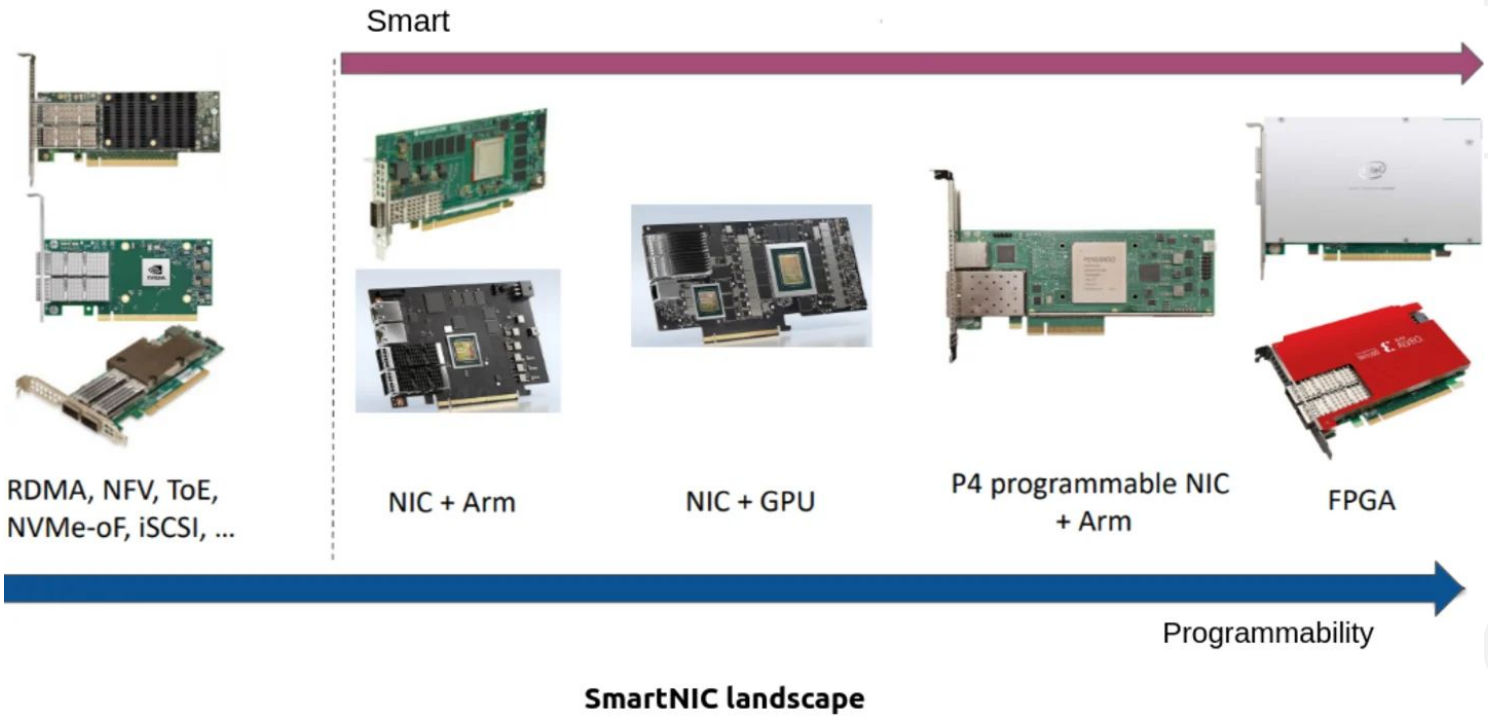
Types of implementation options

- **ASICs**: ASICs (application-specific integrated circuits) incorporate many software-programmable microprocessor cores. This solution can unburden the CPU of tasks like standardized security and storage protocols.
- **FPGA**: A SmartNIC based on FPGA visibly accelerates network functions compared to implementations based solely on software. FPGA-based is more flexible than other because enable people to program almost every function.
- **SoC:** SmartNICs with SoC (system-on-chip) onboard can link a minimum of one or more CPUs with a standard NIC. A SmartNIC with SoC onboard will be helpful if you want to move software components written primarily for generic CPUs.



SmartNIC Architecture

Types of implementation options



SmartNIC landscape

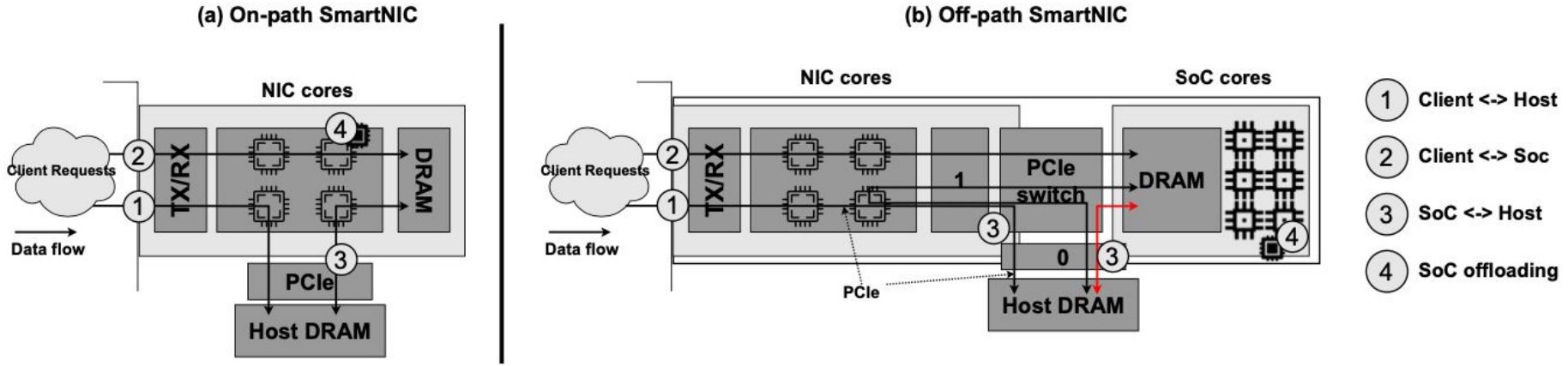
https://ubuntu.com/blog/data-centre-networking-smartnics

SmartNIC Architecture

On-path vs Off-path

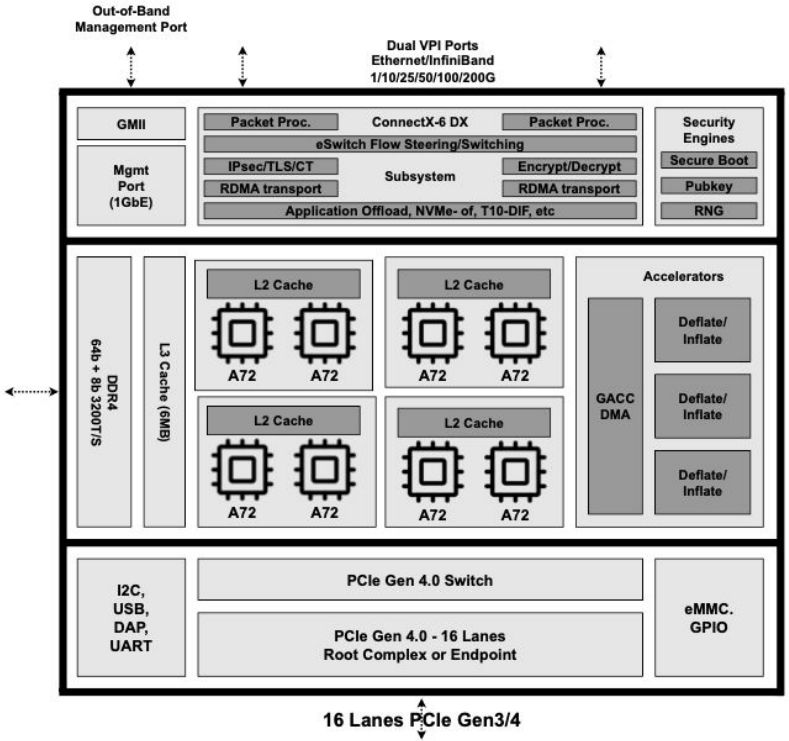
Run-to-completion model: SmartNICs typically employ an alternative processing approach in comparison to an ASIC switch (e.g., Tofino), wherein a packet is directed to a specific processing engine.

Packet latency can vary based on the program structure, including factors such as the number of Match/Action tables and their match types



SmartNIC Architecture

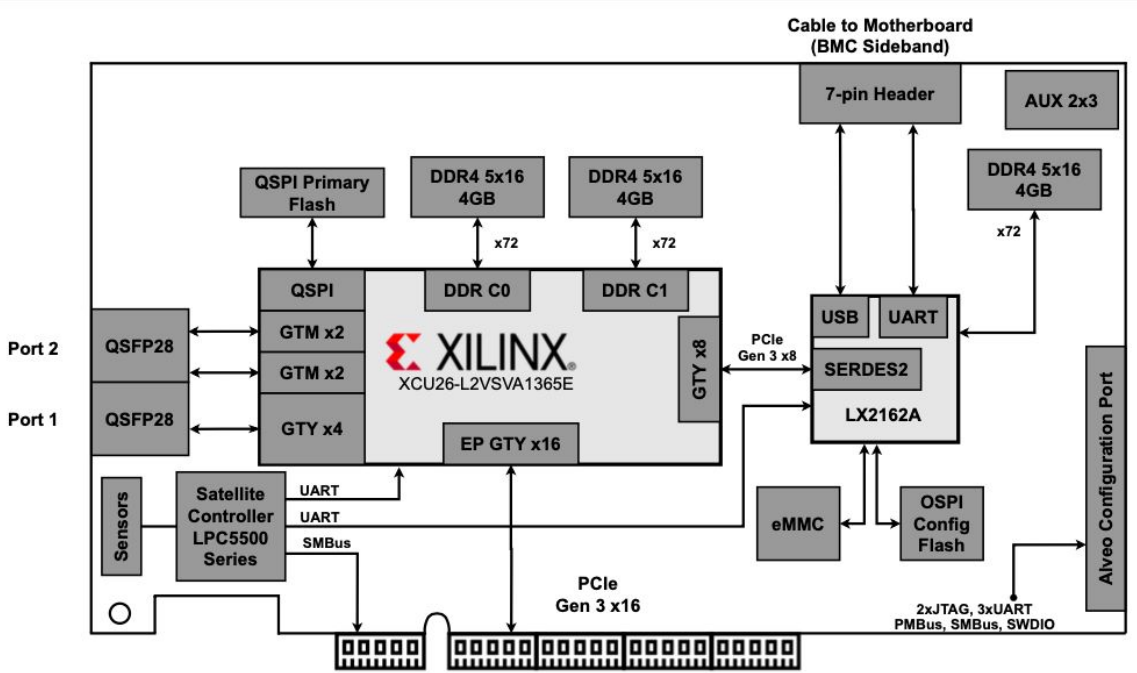
Representative architecture: SoC-based (NVIDIA BlueField)



https://ubuntu.com/blog/data-centre-

SmartNIC Architecture

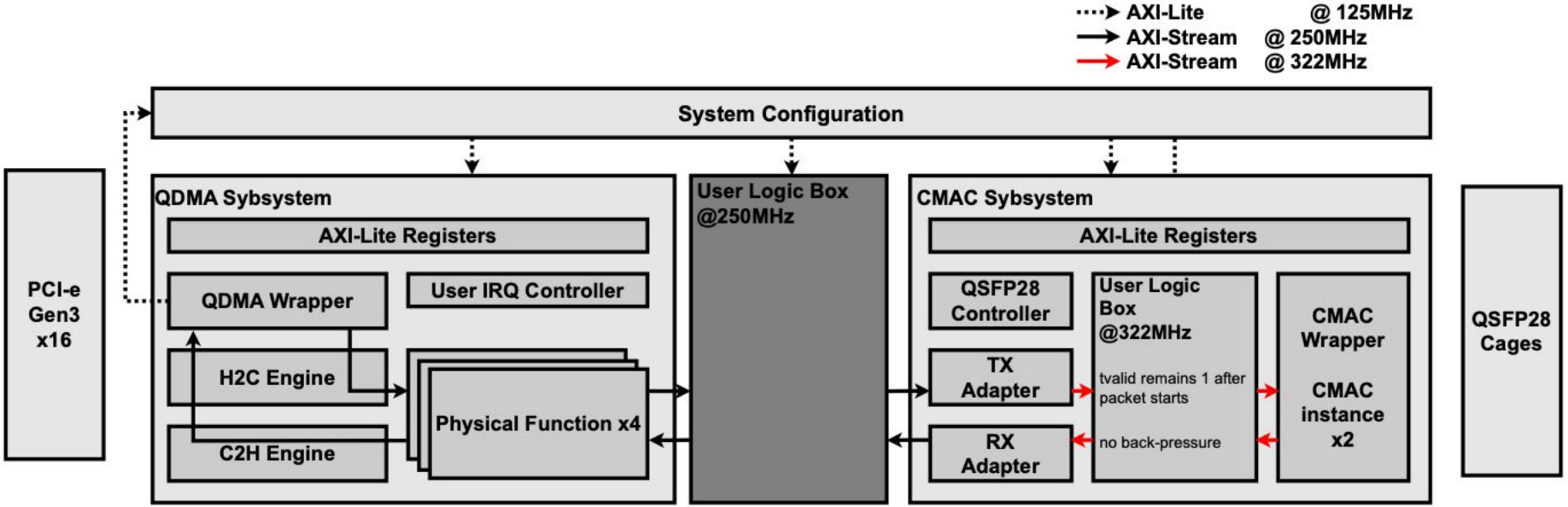
Representative architecture: FPGA-based (Xilinx)



https://ubuntu.com/blog/data-centre-networking-smartnics

SmartNIC Architecture

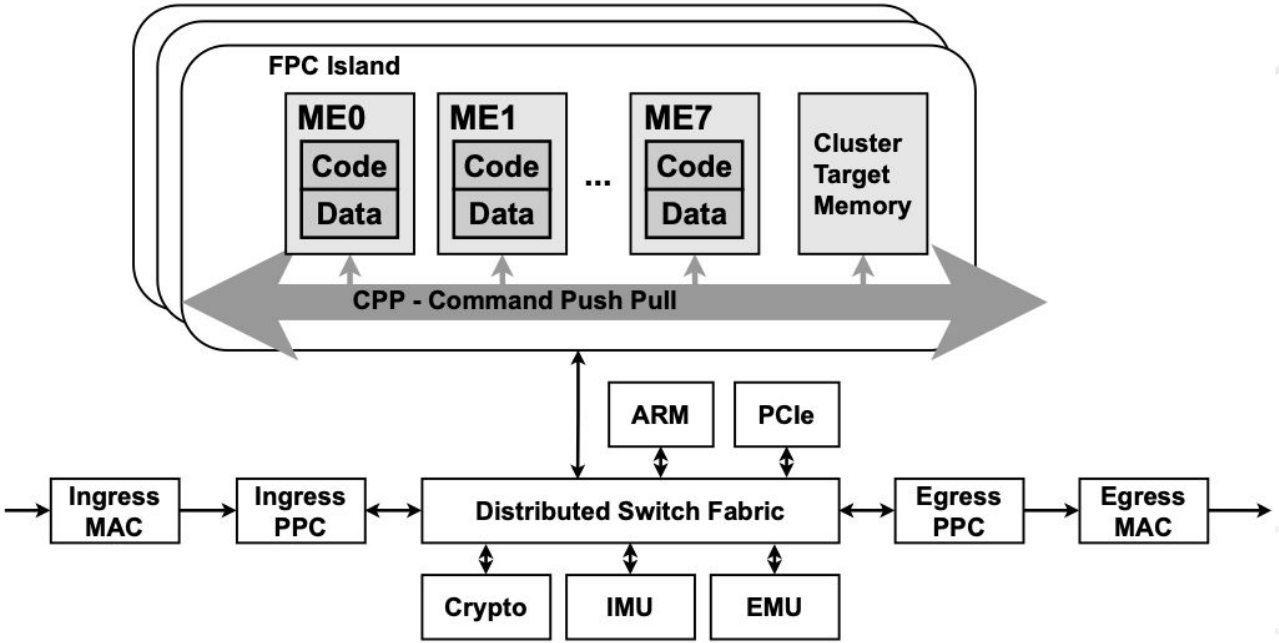
Representative architecture: FPGA-based (OpenNIC project)



https://ubuntu.com/blog/data-centre-networking-smartnics

SmartNIC Architecture

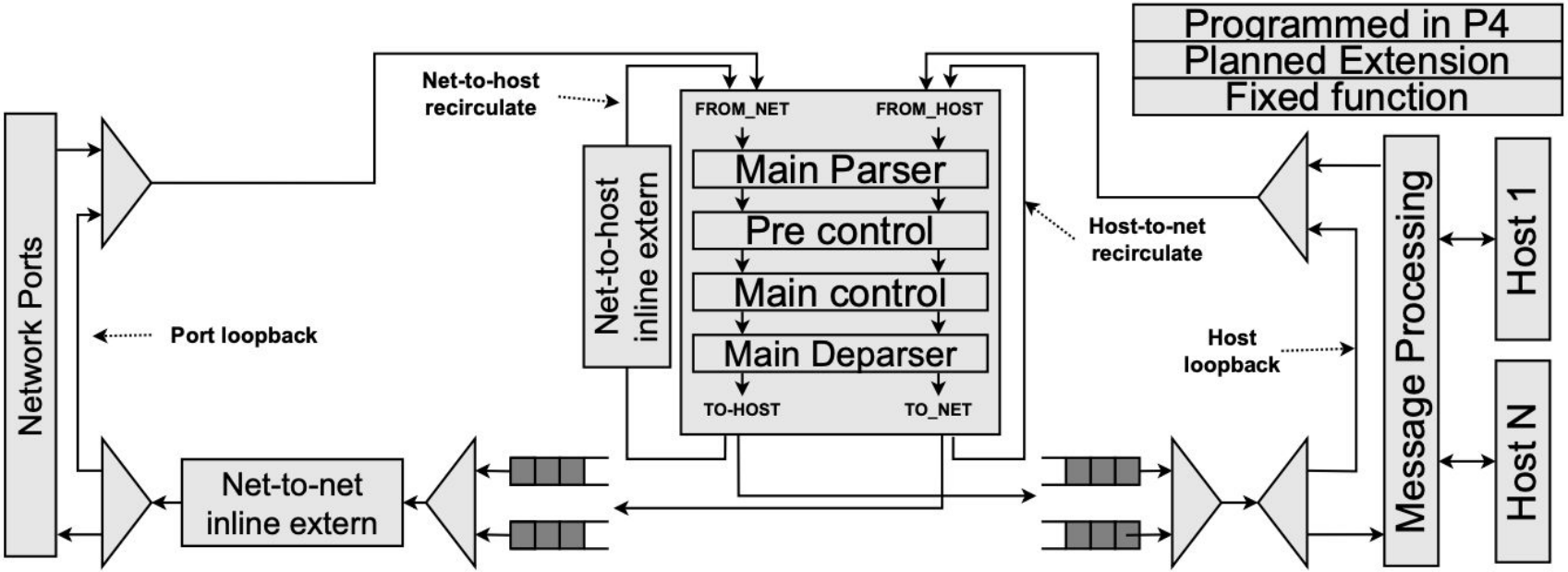
Representative architecture: ASIC-based (Netronome)



https://ubuntu.com/blog/data-centre-networking-smartnics

SmartNIC Architecture

Portable NIC Architecture (PNA)



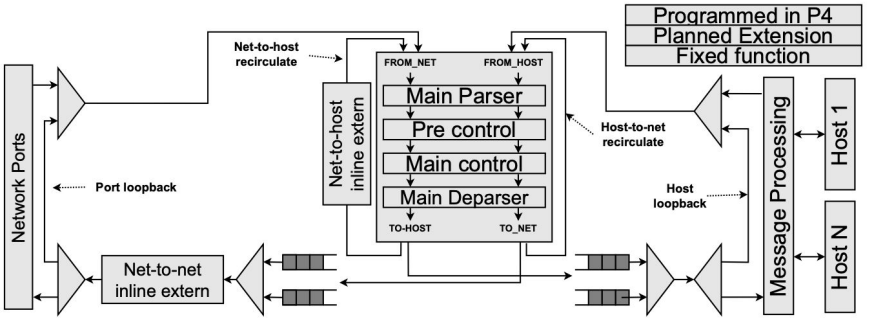
https://ubuntu.com/blog/data-centre-networking-smartnics

SmartNIC Architecture

Portable NIC Architecture (PNA)

- PNA is a design framework aimed at encapsulating the fundamental characteristics of a wide array of NIC devices

- It constitutes a P4 architecture delineating the organization and shared functionalities inherent to NIC devices. PNA consists of two primary elements:
  a. A configurable pipeline engineered to accommodate diverse packet pathways traversing between different ports on the device, such as network interfaces or the host system to which it connects.
  b. A library of types (e.g., intrinsic and standard metadata) and P4 externs (e.g., counters, meters, and registers)

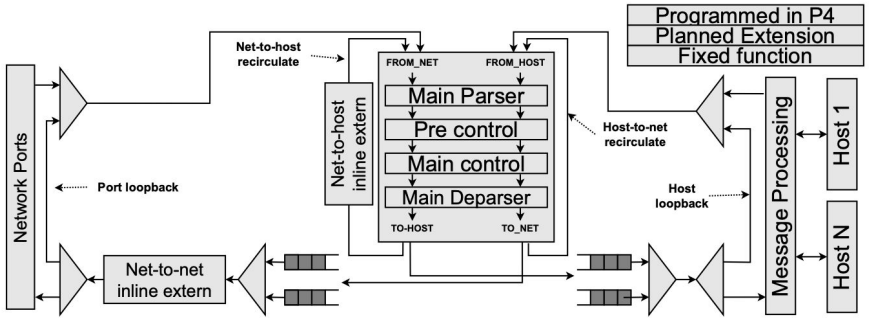


https://ubuntu.com/blog/data-centre-networking-smartnics

SmartNIC Architecture

PNA: few differences from P4 lang

- Although the PNA architecture relies on the P4 language definition, some new features are still not defined in the base P4 language.
- In particular, there are mach-action table properties that are not included in the base P4 language specification.
  a. The add_on_miss table property
  b. table entry timeout notification
- **How to use PNA?** The only P4 backend compiler that adheres to the PNA architecture is the **p4c-dpdk**.
- The p4c-dpdk backend translates the P4-16 programs to DPDK API to configure the DPDK software switch (SWX) pipeline. DPDK introduced the SWX pipeline in the DPDK 20.11 LTS release.



SmartNIC Architecture

Which technology for SmartNICs?

SMARTNESS

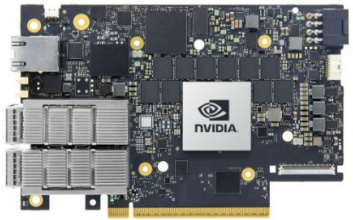
ASIC

Hardware designed for specific application

FPGA

Programmable hardware

SOC



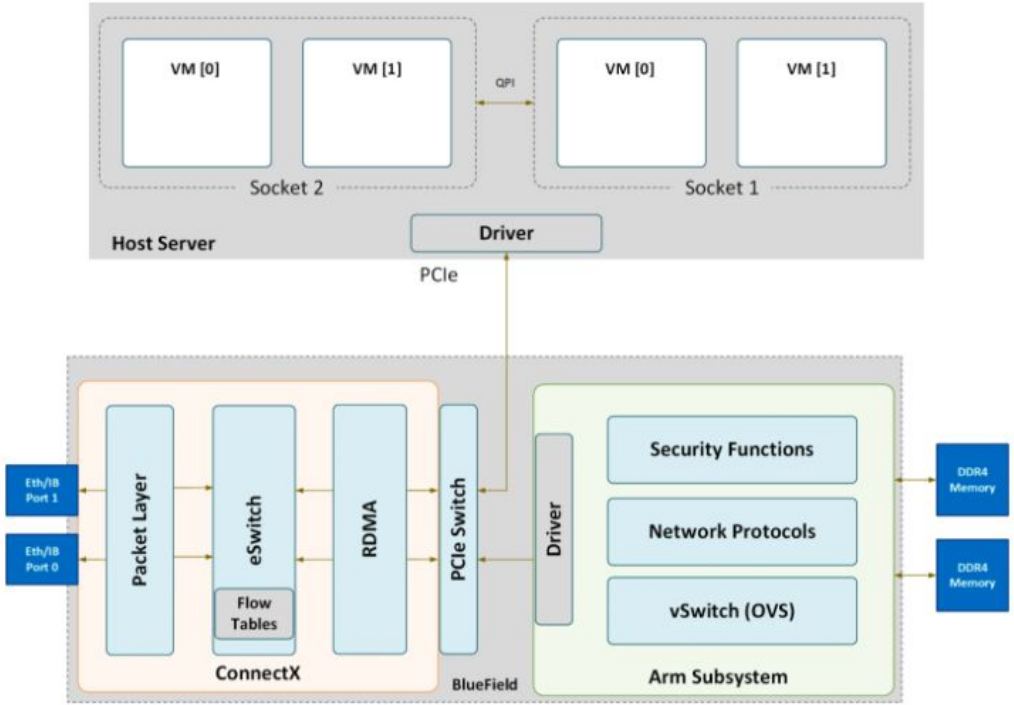
CPU inside the NIC

Good	Performance	**Bad**
Bad	Productivity	**Good**

Hands-on #1

- understanding how physical ports and virtual ports are interconnected in the BlueField

BlueField: packet processing overview



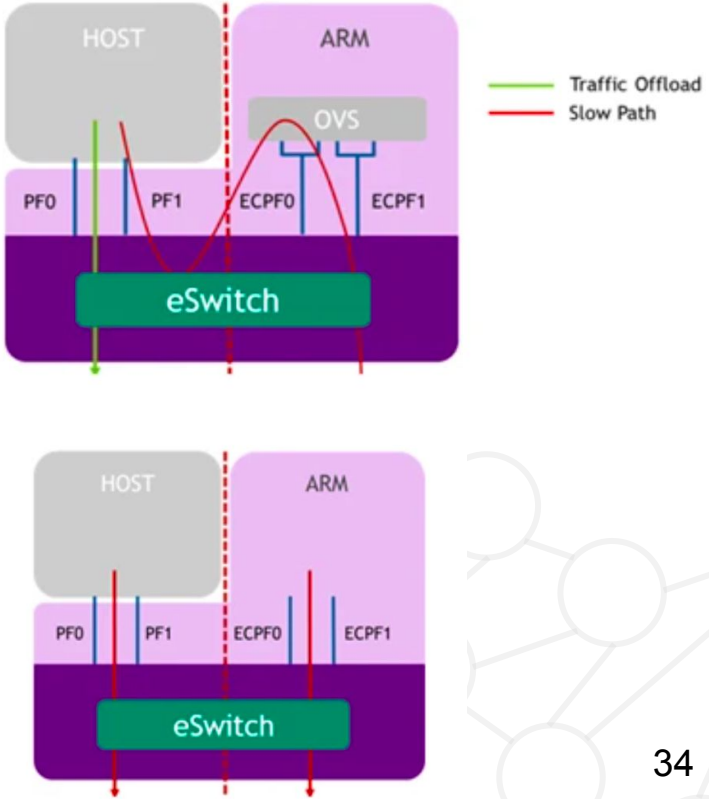
BlueField: operation modes

The NVIDIA BlueField DPU has three modes of operation:

***Embedded function (ECPF) (or DPU mode)** ownership where the embedded Arm system controls the NIC resources and data path (default)

Restricted mode which is an extension of the ECPF ownership with additional restrictions on the host side

Separated host mode (symmetric model)



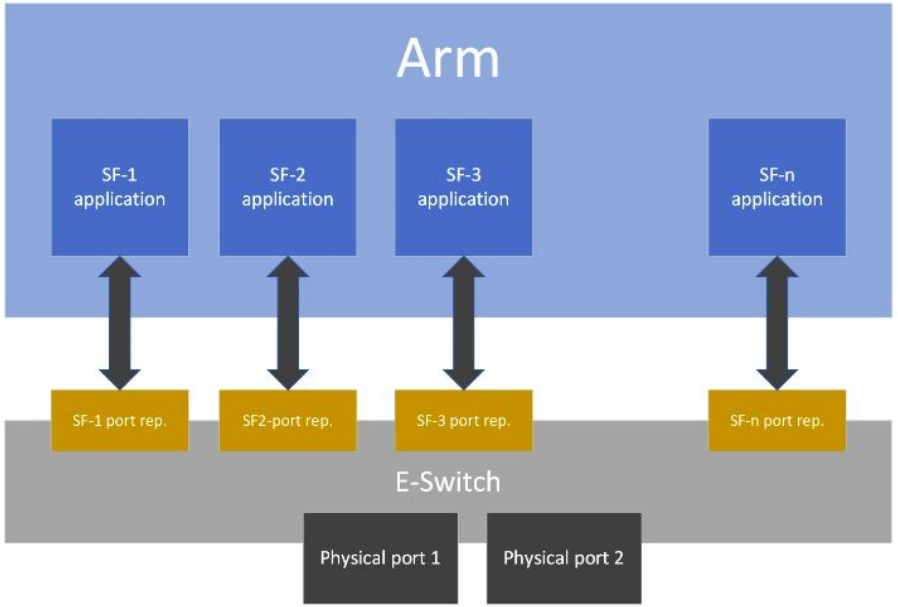
BlueField: Scalable Functions Overview

The core interconnection of Offloaded application is done using **Scalable Function (SF)**

A SF is a lightweight function that has a parent PCIe function on which it is deployed.

An mlx5 SF has its own function capabilities and its own resources.

Dedicated queues (txq, rxq, cq, eq) which are neither shared nor stolen from the parent PCIe function



BlueField: Scalable Functions Overview

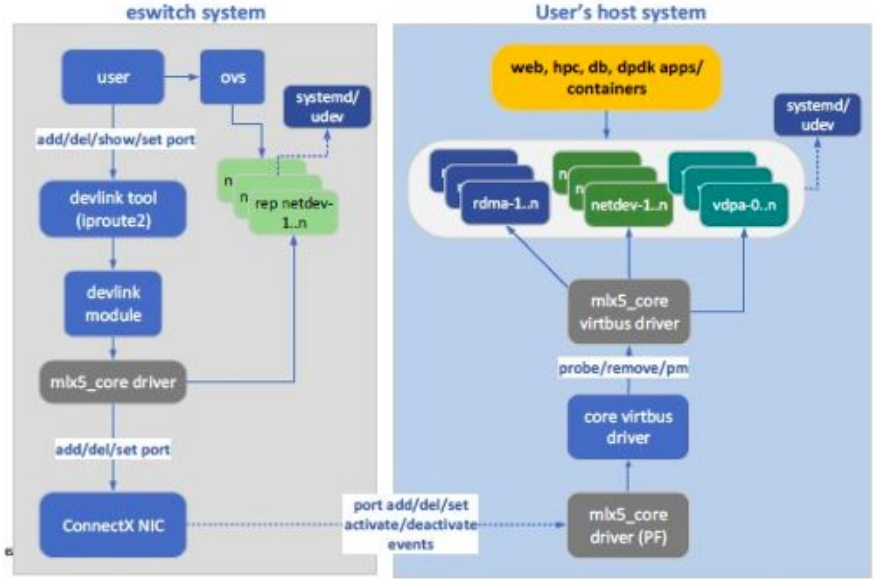
devlink dev show

mlnx-sf -a show

```
root@localhost:/home/ubuntu# mlnx-sf -a show

  SF Index: pci/0000:03:00.0/229409
    Parent PCI dev: 0000:03:00.0
    Representor netdev: en3f0pf0sf10
    Function HWADDR: 00:00:00:00:00:01
    Function trust: on
    Auxiliary device: mlx5_core.sf.2
      netdev: enp3s0f0s10
      RDMA dev: mlx5_2

  SF Index: pci/0000:03:00.1/294945
    Parent PCI dev: 0000:03:00.1
    Representor netdev: en3f1pf1sf10
    Function HWADDR: 00:00:00:00:00:02
    Function trust: on
    Auxiliary device: mlx5_core.sf.3
      netdev: enp3s0f1s10
      RDMA dev: mlx5_3
```

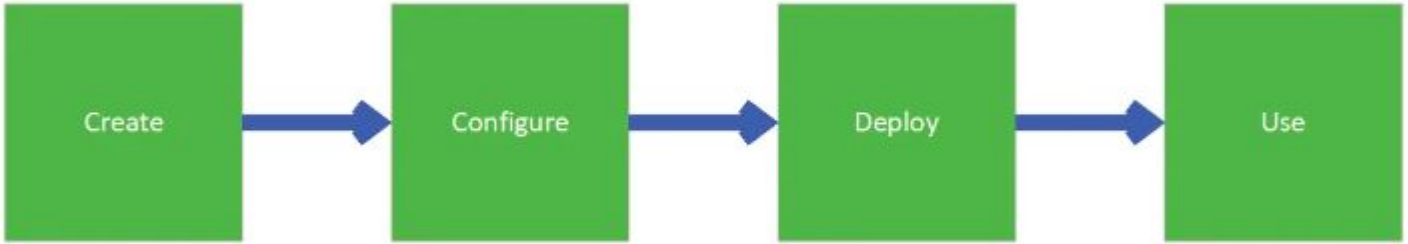


SFs are managed using mlxdevm tool.

/opt/mellanox/iproute2/sbin/mlxdevm

BlueField: Scalable Functions Config.

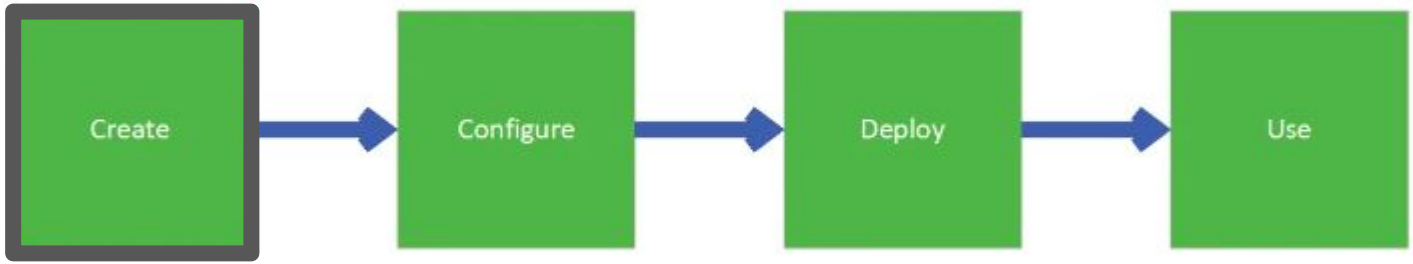
To use a subfunction, a 4-step setup sequence must be followed first:



SFs are managed using mlxdevm tool.

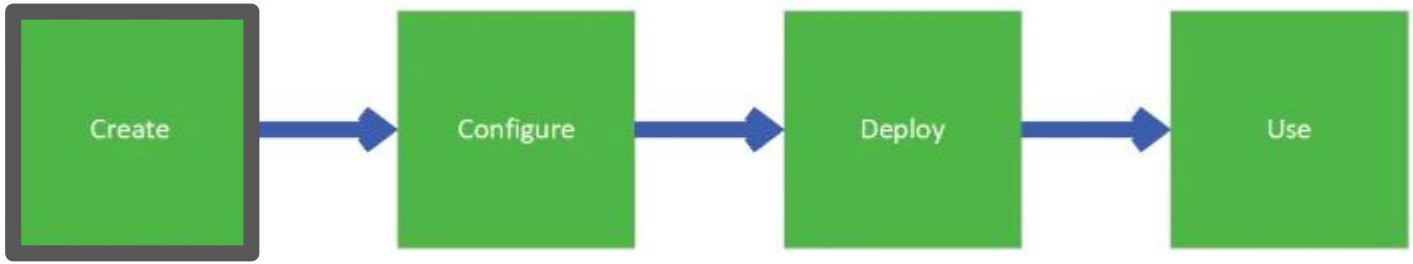
/opt/mellanox/iproute2/sbin/mlxdevm

BlueField: Scalable Functions Config. (creating)



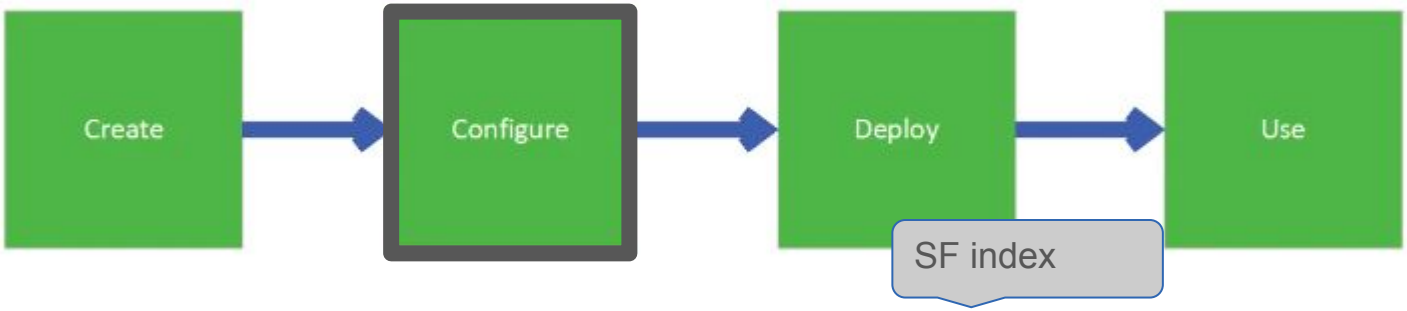

```
# /opt/mellanox/iproute2/sbin/mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum 0 sfnum 10
# /opt/mellanox/iproute2/sbin/mlxdevm port add pci/0000:03:00.1 flavour pcisf pfnum 1 sfnum 10
```

BlueField: Scalable Functions Config. (creating)



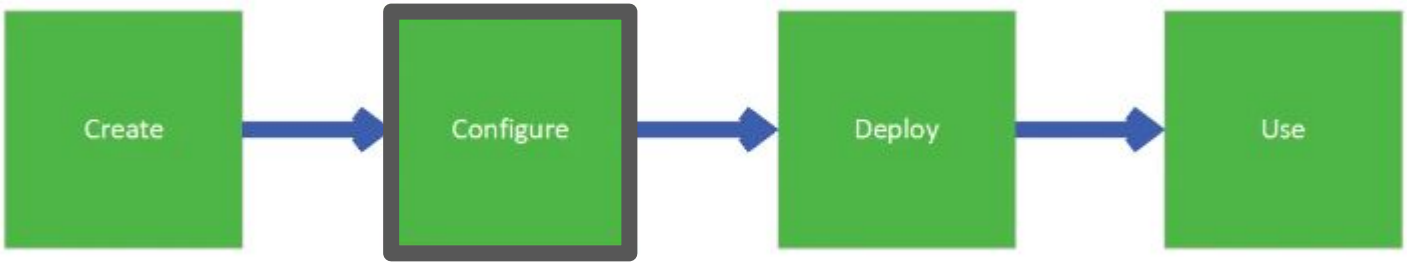

```
# mlnx-sf -a show
```

BlueField: Scalable Functions Config. (configure)



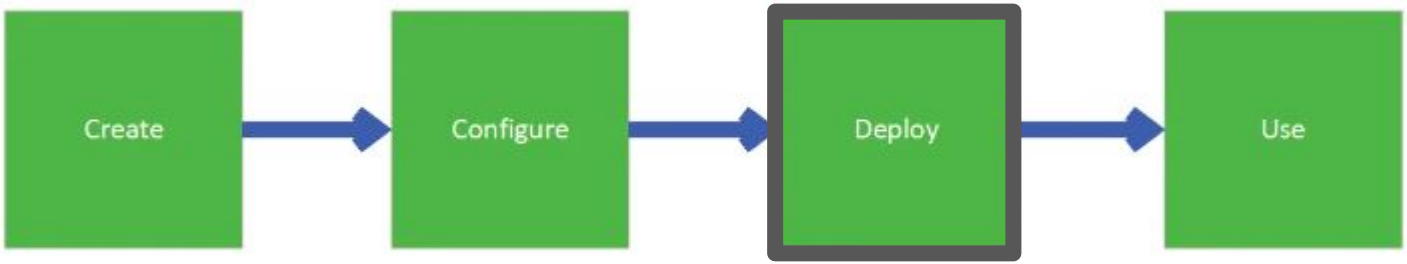

```
# /opt/mellanox/iproute2/sbin/mlxdevm port function set pci/0000:03:00.0/229409
 hw_addr 00:00:00:00:00:01 trust on state active
# /opt/mellanox/iproute2/sbin/mlxdevm port function set pci/0000:03:00.1/294945
 hw_addr 00:00:00:00:00:02 trust on state active
```

BlueField: Scalable Functions Config. (configure)




```
# mlnx-sf -a show
```

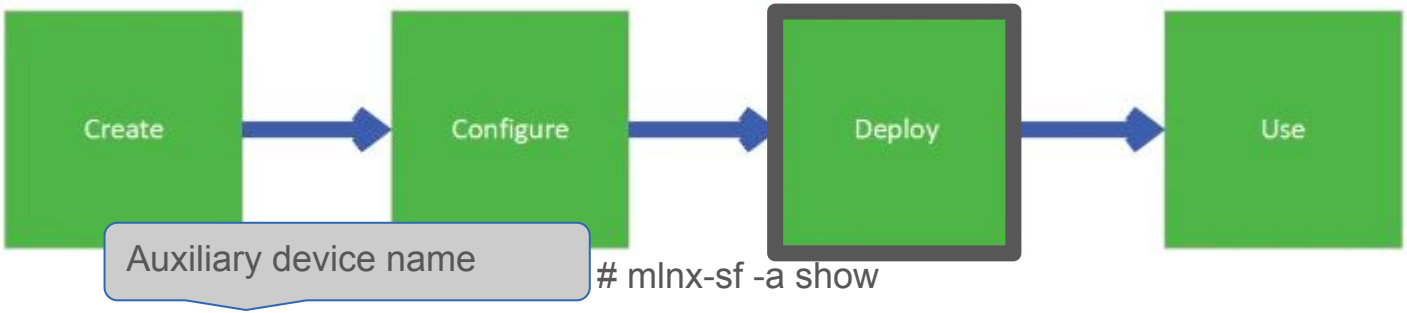
BlueField: Scalable Functions Config. (deploy)




```
# echo mlx5_core.sf.2  > /sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/unbind
# echo mlx5_core.sf.2  > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind

# echo mlx5_core.sf.3  > /sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/unbind
# echo mlx5_core.sf.3  > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind
```

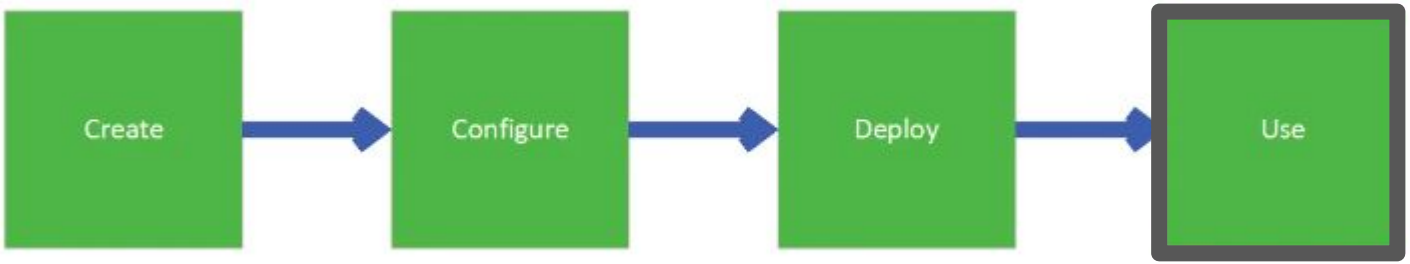
BlueField: Scalable Functions Config. (deploy)




```
# echo mlx5_core.sf.2  > /sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/unbind
# echo mlx5_core.sf.2  > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind

# echo mlx5_core.sf.3  > /sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/unbind
# echo mlx5_core.sf.3  > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind
```

BlueField: Scalable Functions Config. (use)



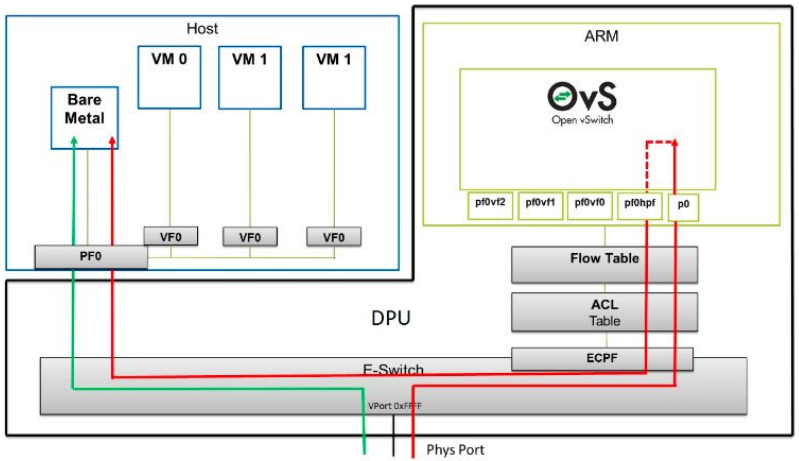

```
# 1) Using with netdev in the Kernel
# 2) Using with NVIDIA DOCA application (or DPDK app)
```

BlueField: vSwitch/representor model in DPU

BlueField uses netdev representors to map each one of the host side physical and virtual functions:

1. Serve as the tunnel to pass traffic for the vSwitch or application running on the Arm cores to the relevant PF/VF on the Arm side.

2. Serve as the channel to configure the embedded switch



For each one of the VFs created on the host side a corresponding representor would be created on the Arm side. The naming convention for the representors is as follows:

- Uplink representors: p<port_number>
- PF representors: pf<port_number>hpf
- VF representors: pf<port_number>vf<function_number>
- SF representors:

BlueField: vSwitch/representor model in DPU

To show how OVS is configured

```
# ovs-vsctl show
```

```
root@localhost:/home/ubuntu# ovs-vsctl show
e350e1eb-a1f2-4eb8-b9ad-8bb2a3a7f86a
    Bridge ovsbr2
        Port p1
            Interface p1
        Port pf1hpf
            Interface pf1hpf
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port en3f1pf1sf10
            Interface en3f1pf1sf10
    Bridge ovsbr1
        Port en3f0pf0sf10
            Interface en3f0pf0sf10
        Port pf0hpf
            Interface pf0hpf
        Port p0
            Interface p0
        Port ovsbr1
            Interface ovsbr1
                type: internal
    ovs_version: "2.17.7-e054917"
```

https://github-wiki-see.page/m/Mellanox/scalablefunctions/wiki

BlueField: vSwitch offloading with ASAP

The virtual switch running on the Arm cores allows us to pass all the traffic to and from the host functions through the Arm cores while performing all the operations supported by OVS.

ASAP2 allows us to offload the datapath by programming the NIC embedded switch and avoiding the need to pass every packet through the Arm cores. The control plane remains the same as working with standard OVS.

```
# ovs-ofctl add-flow bridge_name matching_rules, actions=set_of_actions
# ovs-ofctl add-flow ovsbr1 in_port=p0, actions=output:p1
# ovs-ofctl add-flow ovsbr1 in_port=p0,tcp,nw_dst=1.1.1.1,actions=output:pf0hpf
# ovs-ofctl add-flow ovsbr1 tcp,nw_src=1.1.1.1, nw_dst=2.2.2.2,actions=drop
cpu consumption
```

https://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt

Break 1: vSwitch offloading

1- Configure two scalable functions in the DPU side and make the netdev representors to communicate through the eSwitch

- Goal 1: understand how physical ports and virtual ports are interconnected in the BlueField
- Goal 2: add and configure virtual ports (SFs) in the BlueField
- Goal 3: configure vSwitch offloading
- Goal 4: test connectivity with iperf/ping

Access our GitHub:
https://github.com/smartness2030/netsoft24-smartnic-tutorial

Break 1: vSwitch offloading (step-by-step)

1) create SF

/opt/mellanox/iproute2/sbin/mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum 0 **sfnum 10**

2) configure SF

/opt/mellanox/iproute2/sbin/mlxdevm port function set **pci/0000:03:00.0/229408** hw_addr **00:00:00:00:00:11** trust on state active

3) deploy (unbind/bind driver)

echo **mlx5_core.sf.2** > /sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/unbind

echo **mlx5_core.sf.2** > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind

4) add netdev representor to OVS

ovs-vsctl add-port ovsbr1 **en3f0pf0sf10**

5) set an IP in a different subnet (each one should choose a unique subnet 12.12.[1-50].0/24

ifconfig **enp3s0f0s10 12.12.12.10/24** up

6) add ovs-flow rules

ovs-ofctl add-flow ovsbr1 ip,nw_src=**12.12.12.10**,in_port=**en3f0pf0sf10**,actions=output:**p0**

ovs-ofctl add-flow ovsbr1 ip,nw_dst=**12.12.12.10**,in_port=**p0**,actions=output:**en3f0pf0sf10**

7) test connectivity

ping -I **enp3s0f0s10 12.12.12.20**

iperf -c **12.12.12.20**

Commercial & Market Perspectives

- SmartNICs Ecosystem
- Market Update
- Features of Selected SmartNICs
- CAPEX/OPEX Perspectives

SmartNIC Ecosystem

SMARTNESS

We surveyed most of the SmartNIC vendors to identify:

1. **Available models**
2. **Architectures** (ASIC, FPGA, SoC, or a combination of)
3. **Programming suite** (P4, open-source, vendor-specific, etc)
4. **Data link & Bandwidth** (100Gbit/s, 200Gbit/s, 400Gbit/s, ...)
5. **PCIe** requirement (PCIe 3.0, 4.0, 5.0)
6. **Specific characteristics** of the architectures (e.g., # of cores in SoCs, LUTS in FPGA)

SmartNIC Ecosystem

FPGA-based SmartNICs

Vendor	Model	Release Year	Architecture	Bandwidth	PCIe	Programming Suite	Features
Achronix	Speedster7t FPGA series	2023	FPGA	400 Gbit/s	PCIe 5.0	Achronix Tool Suite (Verilog/VHDL)	Achronix FPGA 692K LUTs
Napatech	NT200A02	2021	FPGA	100Gbit/s	PCIe 3.0	Vivado Suite	Xilinx UltraScale+
Xilinx	u280	2018	FPGA	100Gbit/s	PCIe 4.0	Vivado Suite	Xilinx UltraScale+ 1079K LUTS
Xilinx	**u55c**	**2020**	**FPGA**	**100Gbit/s**	**PCIe 4.0**	**Vivado Suite**	**Xilinx UltraScale+ 1079K LUTS**
Intel	N5010	-	FPGA	4x 100Gbit/s	PCIe 4.0	Intel Open FPGA Stack (Intel OFS)	2073K LUTS Intel Stratix 10 DX
Intel	N6000-PL	2021	FPGA	200Gbit/s	PCIe 4.0	-	Intel Agilex 7 FPGAs F-Series 1437k LUTS

SmartNIC Ecosystem

SoC-based SmartNICs

SMARTNESS

Vendor	Model	Release Year	Architecture	Bandwidth	PCIe	Programming Suite	Features
Asterfusion	Helium EC2002P	2021	SoC	200Gbit/s	PCIe 3.0/PCIe 4.0	DPDK/VPP Development Kit	24 ARM 1.8Ghz + Dedicated acellerator + ASIC
Fungible (now Microsoft)	F1 DPU	2019	SoC	800Gbit/s	4x PCIe 4.0	P4 alike	MIPS-64, 9-stage, dual-issue, 4xSMT, FPU/SIMD unit
Kalray	K200/K200-LP	2021	SoC	100Gbit/s	PCIe 4.0	P4 alike	5 cluster of 16 cores -- MPPA -- specific cores -- 600 to 1.2 Ghz
Marvell	LiquidIO™ III	2022	SoC	2x 50Gbit/s	PCIe 4.0	Standard DPDK, VPP, SPDK	Up to 36 cores ARM V8 at 2.2GHz
NVDIA	**BlueField-2**	**2020**	**SoC**	**2x 100Gbit/s**	**PCIe 4.0**	**DOCA framework, DPDK, C**	**8 Armv8 A72 cores**
NVIDIA	**BlueField-3**	**2022**	**SoC**	**up to 400Gbit/s**	**PCIe 5.0**	**DOCA framework, DPDK, C**	**16 Armv8.2+ A78 Hercules cores**
Broadcom	Stingray	2020	SoC	100Gbit/s	PCIe 4.0	DPDK	cluster of four 3 GHz dual-core Arm v8 A72 complexes

SmartNIC Ecosystem

(SoC+FPGA)-based SmartNICs

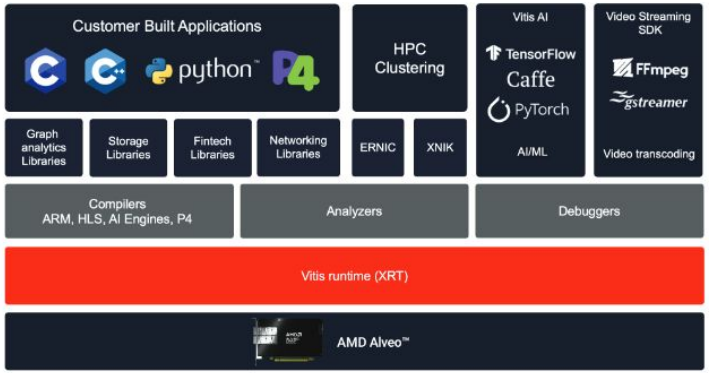
Vendor	Model	Release Year	Architecture	Bandwidth	PCIe	Programming Suite	Features
Intel	**Intel FPGA IPU F2000X-PL Platform**	**2021**	**SoC + FPGA**	**200Gbit/s**	**4x PCIe 4.0**	**Infrastructure Programmer Development Kit (IPDK) Storage Performance Development Kit (SPDK) P4 Programmable**	**Intel Agilex 7 FPGA F-Series 2300 k LUTS + 8-core Intel Xeon-D SoC**
Xilinx	**SN1000**	**2021**	**SoC+FPGA**	**200Gbit/s**	**PCIe 4.0**	**Vivado, Vitis, P4, DPDK**	**AMD XCU26 FPGA UltraScale+, 16-core Arm**
Netronome	Agilio CX	2015	ASIC/NPU	40Gbit/s	PCIe 3.0	P4, Micro-C / Agilio Software	48/60 NFP-4000 flow processor

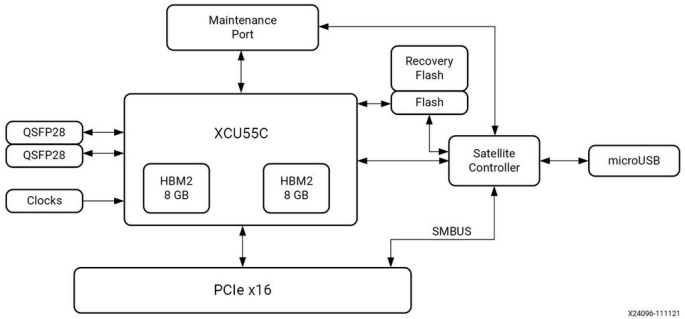
FPGA-based SmartNICs

Xilinx u55c

SMARTNESS

Features	AMD Alveo™ U55C Card
Architecture	Virtex UltraScale+
LUTs	1,304K
Registers	2,607K
DSP Slices	9,024
Form Factor	Single Slot, Full Height, Half Length
Micro-USB	Yes
HBM Memory	16GB
HBM Bandwidth	460 GB/s
N/W Interface	2 x QSFP28 ports
PCI Express	PCIe Gen3 x 16, 2 x PCIe Gen 4x8
Thermal Solution	Passive
Power (TDP)	115W
Tool Support	Vitis™





https://www.xilinx.com/content/dam/xilinx/publications/product-briefs/xilinx-alveo-u55c-product-brief.pdf

SoC-based SmartNICs

NVIDIA BlueField2

NETWORK AND HOST INTERFACES

Network Interfaces
- Ethernet - Dual ports of 10/25/50/100Gb/s, or a single port of 200Gb/s
- InfiniBand - Dual ports of EDR / HDR100, or single port of HDR

PCI Express Interface
- 8 or 16 lanes of PCIe Gen 4.0
- PCIe switch bi-furcation with 8 downstream ports

ARM/DDR SUBSYSTEM

Arm Cores
- Up to 8 Armv8 A72 cores (64-bit) pipeline
- 1MB L2 cache per 2 cores
- 6MB L3 cache with plurality of eviction policies

DDR4 DIMM Support
- Single DDR4 DRAM controller
- 16GB / 32GB of on-board DDR4
- ECC error protection support

HARDWARE ACCELERATIONS

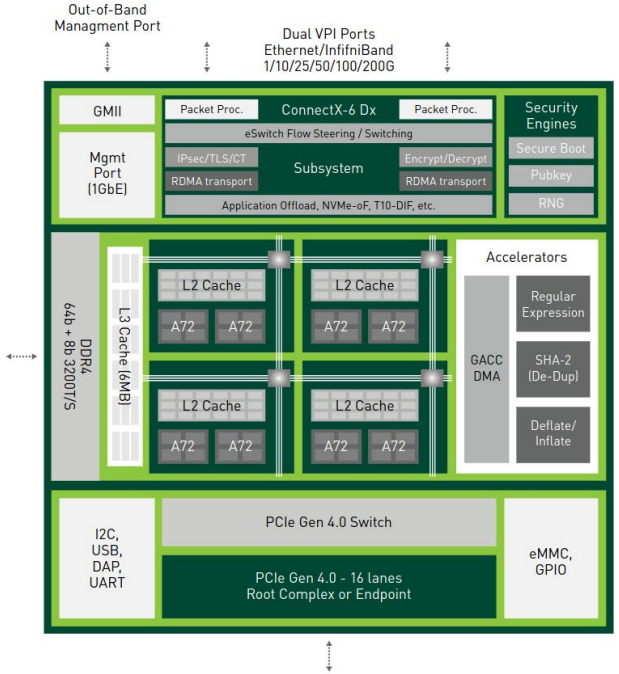
Security
- Secure boot with hardware root-of-trust
  - Secure firmware update
  - Cerberus compliant
- Regular expression (RegEx) acceleration
- IPsec/TLS data-in-motion encryption
- AES-GCM 128/256-bit key
- AES-XTS 256/512-bit data-at-rest encryption
- SHA 256-bit hardware acceleration
- Hardware public key accelerator
  - RSA, Diffie-Hellman, DSA, ECC, EC-DSA, EC-DH
- True random number generator (TRNG)

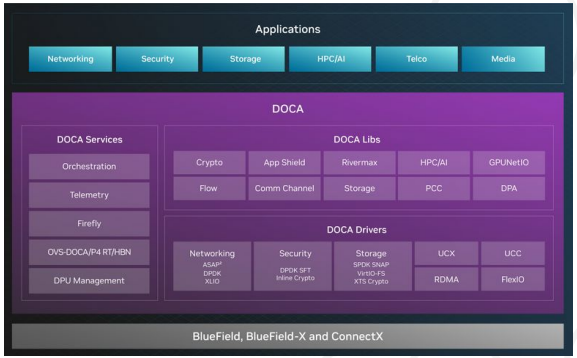
Storage
- BlueField SNAP - NVMe™ and VirtIO-blk
- NVMe-oF™ acceleration
- Compression and decompression acceleration
- Data hashing and deduplication

Networking
- RoCE, Zero Touch RoCE
- Stateless offloads for:
  - TCP/UDP/IP
  - LSO/LRO/checksum/RSS/TSS/HDS
  - VLAN insertion/stripping
- SR-IOV
- VirtIO-net
- Multi-function per port
- VMware NetQueue support
- Virtualization hierarchies
- 1K ingress and egress QoS levels

Boot Options
- Secure boot (RSA authenticated)
- Remote boot over Ethernet
- Remote boot over iSCSI
- PXE and UEFI

Management
- 1GbE out-of-band management port
- NC-SI, MCTP over SMBus, and MCT over PCIe
- PLDM for Monitor and Control DSP0248
- PLDM for Firmware Update DSP026
- I2C interface for device control and configuration
- SPI interface to flash
- eMMC memory controller
- UART
- USB





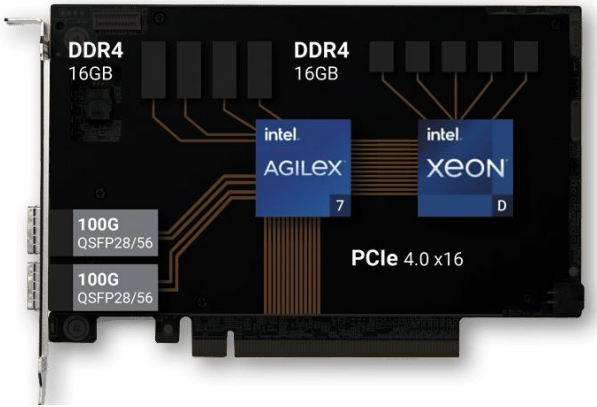
https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf
https://network.nvidia.com/files/doc-2020/pb-bluefield-2-dpu.pdf

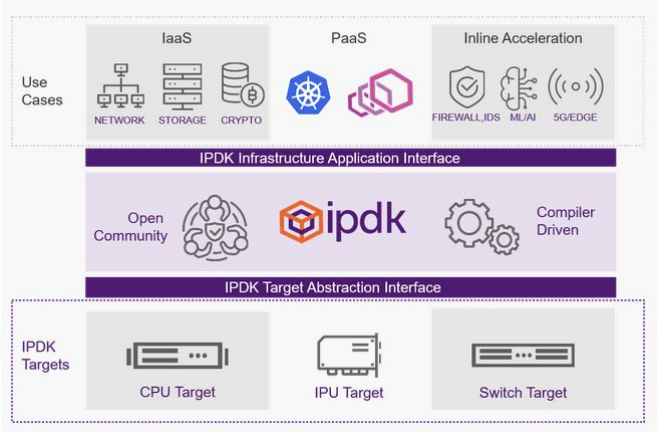
SoC+FPGA-based SmartNICs

Intel IPU F2000X-PL

SMARTNESS

FPGA	Intel Agilex® FPGA F-Series AGFC023 • 2.3M LEs, 782.4K ALMs • 246 Mb on-chip memory • 1,640 digital signal processing (DSP) blocks • Hardened crypto • 4x4 GB FPGA memory (error correcting code (ECC), 40 bit, 2,666 MTps)
CPU	Intel® Xeon® D-1736 SoC • 8 cores, 16 threads • 2.3 GHz, 3.4 GHz turbo frequency • 15 MB cache • 2x8 GB DDR4 Intel Xeon SoC memory (ECC, 72 bit, 2,900 MTps) • 64 GB NVMe* in the M.2 slot on the IPU for the SoC operating system
PCI Express Interfaces	• PCIe* 4.0 x16 (16 GTps) between the CPU and host • PCIe 4.0 x16 (16 GTps) between the FPGA and CPU
Front Panel Network Interface	• 2-port QSFP28/56 • 2x100GBASE-LR4/SR4/CR4 • 2x10/25GBASE-LR/SR/CR (QSFP/SFP adapter) • 8x10/25GBASE-SR/CR (breakout cable) • Dedicated 1G management port (RJ45)
Mechanical, Thermal, and Power	• Full-height, ¾ length, single-slot PCIe form factor (note: partners may develop versions with different form factors) • Maximum power consumption for standard hardware configuration: 150 W
Board Management Controller (BMC)	• Ethernet, USB (front panel), UART (internal) connectivity • Secure FPGA image update • Wake-on-LAN • MCTP over SMBus • MCTP over PCIe VDM • Dedicated NC-SI RBT internal port • PLDM for monitor and control (DSP0248) • PLDM for FRU (DSP0257)
SoC Operating System	• Fedora Linux* • UEFI basic input/output system (BIOS) • PXE boot support • Full shell access via SSH and UART
Tool Support	• Intel Acceleration Stack for Intel Xeon CPU with FPGAs • Intel Quartus® Prime Design Software • Data Plane Development Kit (DPDK) • Storage Performance Development Kit (SPDK) • Infrastructure Programmer Development Kit (IPDK)
Environment and Approvals	• EU, US, and APCJ regulatory approvals • Thermal, shock, and vibration tested • UL marked, RoHS, and REACH compliant • Temperature range: -5°C to +45°C • ASHRAE class A2
Application Stack Acceleration Framework	• Framework for embedding customer Accelerator Functional Units (AFU) implementing workload acceleration/offload in FPGA • Six AFUs supported • Throughput up to 200 Gbps • Look-aside and inline AFU configurations • Pre-integrated AFUs for host VirtIO-net direct memory access (DMA), SoC VirtIO-net DMA, and packet processor with foundational network interface card (NIC) functions.





https://www.intel.com/content/www/us/en/products/details/network-io/ipu/f2000x-pl-platform.html
https://ipdk.io/documentation/Recipes/PaaSOffloadKubernetes/

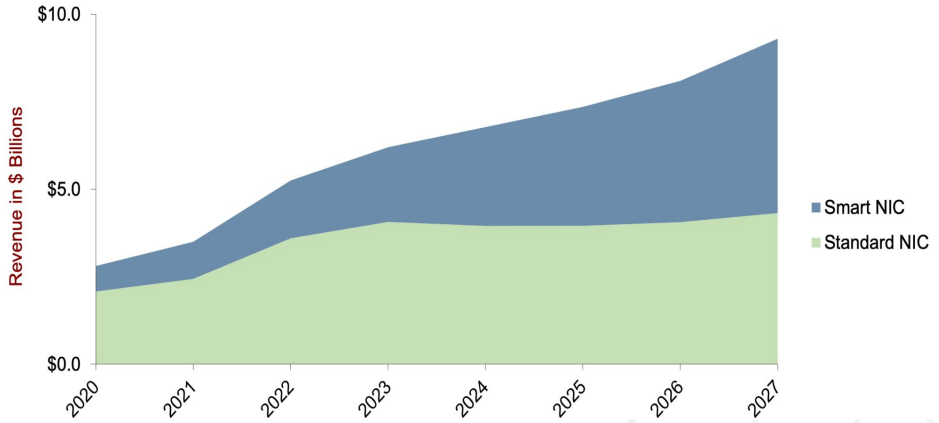
Market Perspectives

SmartNIC Market Update

DPU Vendors & Product lines Summarize

Logo	Vendor	Product Line	Core Processor
Achronix Data Acceleration	Achronix	Speedster7t FPGA series	FPGA
AMD	AMD	(Xilinx,Pensando) Alveo series、Elba、Capri	FPGA, SoC
Asterfusion	Asterfusion	Helium SmartNIC	SoC ARM+ASIC+Dedicated Accelerator
aws	AWS	Nitro system	SOC
Microsoft Azure	Azure	Catapult	GP SoC
BROADCOM	Broadcom	Stingray	SoC:ARM+ASIC
FUNGIBLE	Fungible	FI DPU	NP SoC
intel	Intel	IPU	FPGA+X86 SoC、FPGA+ARM SoC
KALRAY THE POWER OF MORE	Kalray	K200/K200-LP	MPPA DPU Processor
MARVELL	Marvell	OCTEON 10 DPU	SoC:ARM+ASIC
napa:tech;	Napatech	NT200A02 SmartNIC	FPGA
METRONOME	Metronome	Agilio Series	
NVIDIA	Nvidia	BlueField DPU	SoC:ARM+ASIC+Dedicated Accelerator/ GPU Dedicated Accelerator
Silicom Connectivity Solutions	Silicom	N5010 , N5110A, P425G2SNxIAONIC	FPGA

Ethernet Adapter Market Opportunity

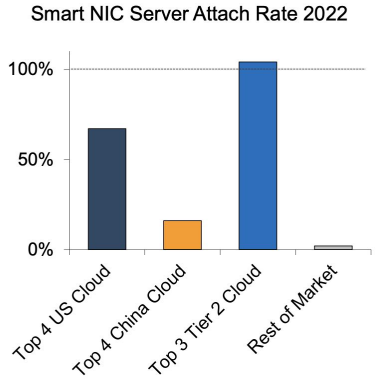


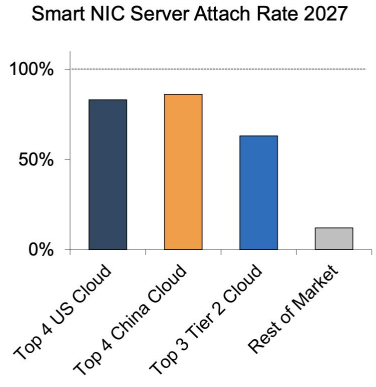
https://smartnicssummit.com/proceeding_files/a0q5f000003HkUt/20230614_PLEN_Fung.PDF

Market Perspectives

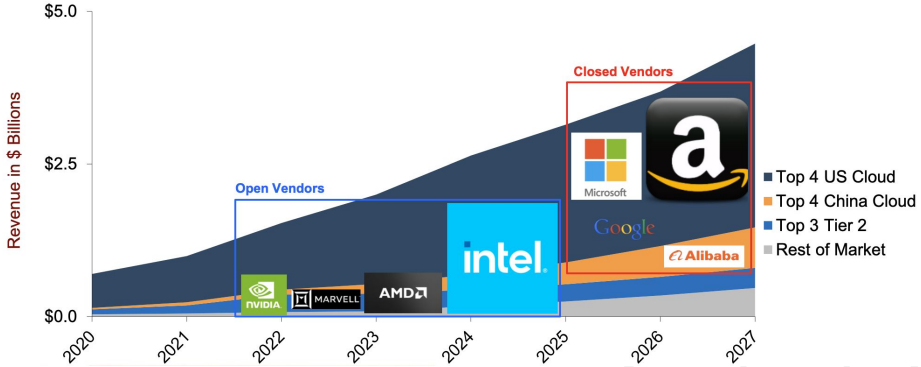
SmartNIC Market Update

SmartNIC Adoption





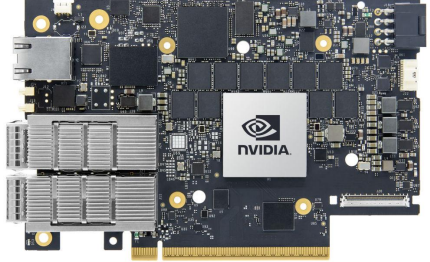
SmartNIC Market Oportunity



https://smartnicssummit.com/proceeding_files/a0q5f000003HkUt/20230614_PLEN_Fung.PDF

Commercial Products

Features of Selected SmartNICs



SMARTNESS

KEY SOFTWARE-DEFINED, HARDWARE-ACCELERATED APPLICATIONS

Cloud Networking
Cloud overlay, SDN acceleration, NAT, load balancer, NFV, video streaming

Storage
NVMe™ over Fabrics (NVMe-oF™), NVMe/TCP™, elastic storage, hyper converged infrastructure (HCI), encryption, data integrity, data deduplication, de-compression, erasure coding/RAID

Security
Distributed next-generation firewall, IDS/IPS, root of trust, micro-segmentation, DDOS prevention

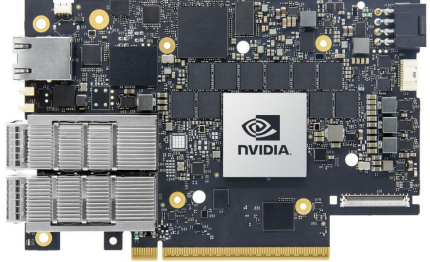
HPC / AI
Cloud-native supercomputing, multi-tenancy and security, communication accelerations

Telco and Edge
Cloud RAN, virtualized edge gateways, VNF acceleration, edge microservers

https://lenovopress.lenovo.com/lp1809-thinksystem-nvidia-bluefield-3-qsfp112-2-port-200gb-dpu-adapter
https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf

Commercial Products

Features of Selected SmartNICs



SMARTNESS

NVIDIA BlueField-3 QSFP112 2-Port 200Gb DPU Adapter

Network and Host Interfaces

Network Interfaces

- Ethernet - 1, 2, 4 ports with up to 400 Gb/s connectivity
- InfiniBand - Single port of NDR (400Gb/s), or dual ports of NDR200 / HDR (200Gb/s)

PCI Express Interface

- 32 lanes of PCIe Gen 5.0
- PCIe switch bi-furcation of up to 16 downstream ports
- Non-transparent bridging (NTB) support

Compute and Memory

Arm CPU Cores

- Up to 16 Armv8.2+ A78 Hercules cores (64-bit)
- 8MB L2 cache
- 16MB LLC system cache

Programmable Datapath Accelerator

- 16 cores, 256 threads
- Programmability through DOCA
- Heavy multi-threading applications acceleration

DDR DIMM Support

- Dual DDR5 5600MT/s DRAM controllers
- 16GB on-board DDR5
- ECC error protection support

Hardware Accelerations

Security

- Secure boot with Public key accelerator (PKA) root-of-trust
  - Secure firmware update
  - Flash encryption
  - Cerberus compliant
  - Functional isolation layer
  - Regular expression (RegEx) matching processor
  - MACsec/IPsec/TLS data-in-motion encryption
    - AES-GCM 128/256-bit key
  - AES-XTS 256/512-bit data-at-rest encryptio
  - Connection tracking for stateful firewall
  - Public key accelerator (PKA)
    - RSA, Diffie-Hellman, DSA, ECC, EC-DSA, EC-DH
  - True random number generator (TRNG)

https://lenovopress.lenovo.com/lp1809-thinksystem-nvidia-bluefield-3-qsfp112-2-port-200gb-dpu-adapter
https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf

Commercial Products

Features of Selected SmartNICs

SMARTNESS

NVIDIA BlueField-3 QSFP112 2-Port 200Gb DPU Adapter

Storage

- BlueField SNAP - Elastic block storage - NVMe™ and VirtIO-blk
- NVMe-oF™ and NVMe/TCP™ acceleration
- Decompression engine
- Erasure coding for RAID implementation
- M.2 / U.2 connectors for direct attached storage

HPC/AI Accelerations

- HPC / AI All-to-All engine
- NVIDIA GPUDirect
- NVIDIA GPUDirect Storage (GDS)

Networking

- RoCE, Zero Touch RoCE
- ASAP² - Accelerated Switch and Packet Processing® for SDN and VNF acceleration
- Single Root I/O Virtualization (SR-IOV)
- VirtIO acceleration
- Overlay network acceleration
  - VXLAN, GENEVE, NVGRE
- Programmable flexible parser: user defined classification
- Connection tracking (L4 firewall)
- Flow mirroring, sampling and statistics
- Header rewrite
- Hierarchical QoS
- Stateless TCP offloads

Advanced Timing and Synchronization

- IEEE 1588v2 (any profile)
- G.8273.2 Class C
- PTP hardware clock (PHC)
- Line rate hardware timestamp
- SyncE
- G.8262.1 (eEEC)
- Configurable PPS In and PPS Out
- Time triggered scheduling
- Time-based SDN acceleration

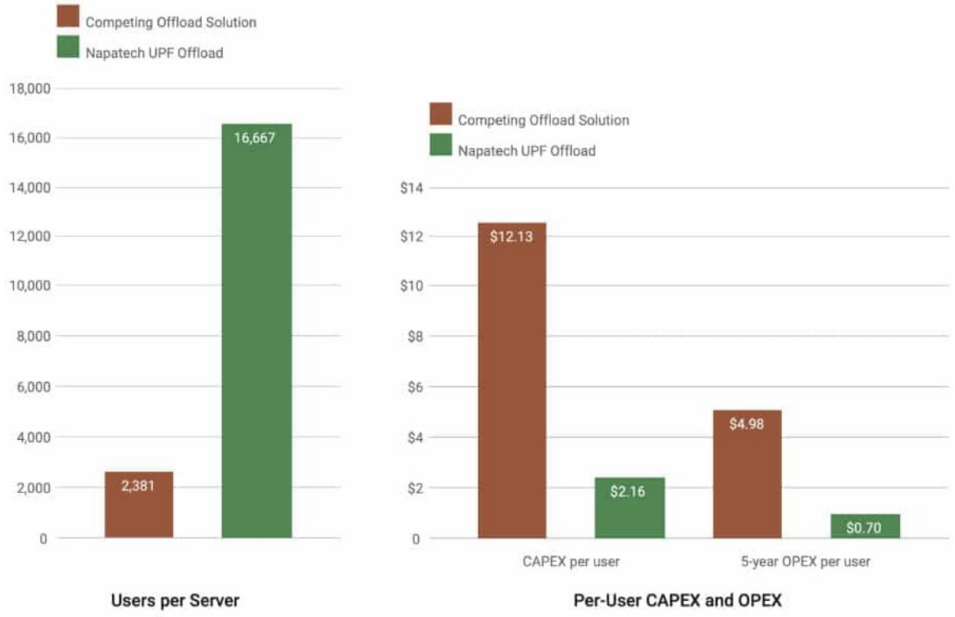
https://lenovopress.lenovo.com/lp1809-thinksystem-nvidia-bluefield-3-qsfp112-2-port-200gb-dpu-adapter
https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf

CAPEX/OPEX Perspectives

Tailoring SmartNICs for Specific Environments

The example illustrated above reflects the analysis of a **metro edge data center running a 5G Packet Core** to support **50,000 5G users**.

- Operational expenses (OPEX) were calculated over a five-year period and included the cost of power as well as server OPEX.
- Servers were assumed to be industry standard platforms with two PCIe slots configured with NT200 SmartNICs.



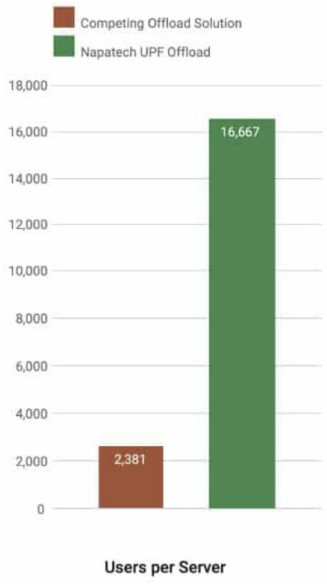
https://www.napatech.com/support/resources/solution-descriptions/smartnic-offload-solution-for-5g-user-plane-function/

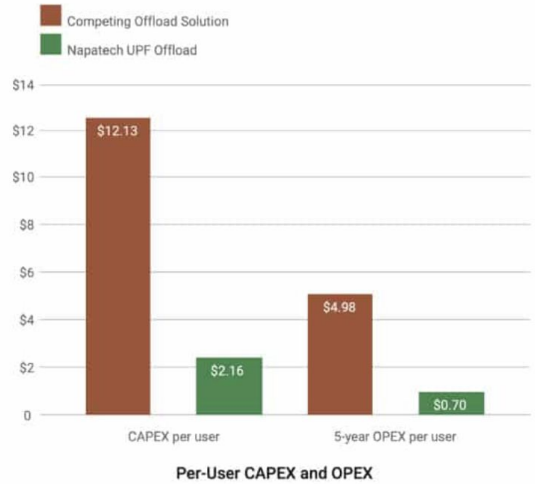
CAPEX/OPEX Perspectives

Tailoring SmartNICs for Specific Environments

Based on the full set of assumptions, available on request, the above scenario yields the following benefits:

- **16,667** users supported per server when the Napatech UPF Offload solution is used, compared to **2,381** for a competing offload solution, **an increase of 7x**;
- Per-user CAPEX of **$2.16** for the Napatech UPF Offload solution is used, compared to **$12.13** for a competing offload solution, **a reduction of 82%**;
- Per-user 5-year OPEX of **$0.70** for the Napatech UPF Offload solution is used, compared to **$4.98** for a competing offload solution, **a reduction of 86%.**





https://www.napatech.com/support/resources/solution-descriptions/smartnic-offload-solution-for-5g-user-plane-function/

SMARTNESS

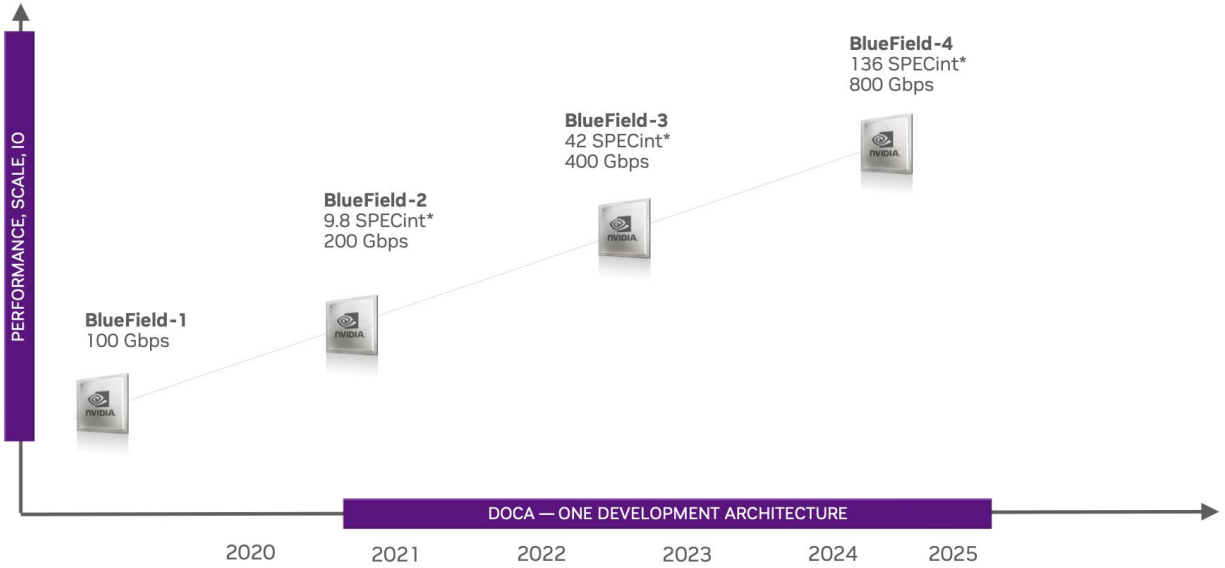
Performance and Benchmarking

- Metrics and Benchmarks to Assess Improvements

Performance and Benchmarking

Metrics and Benchmarks to Assess Improvements

Evolution of NVIDIA DPU BlueField

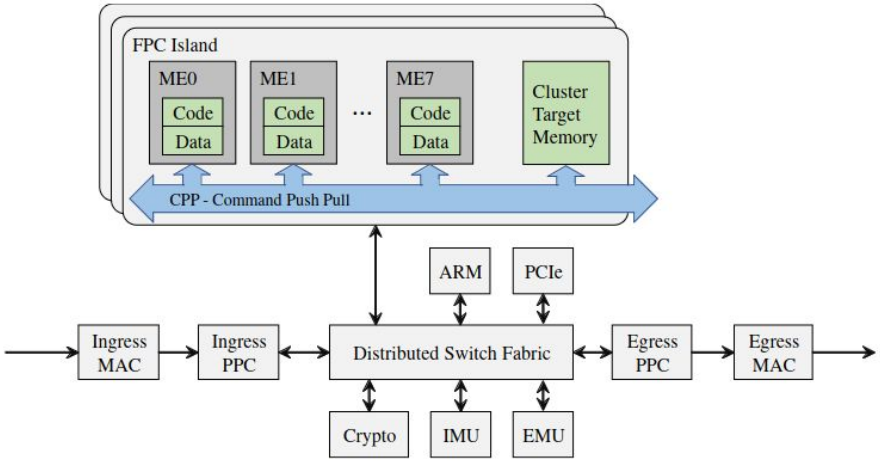


SmartNIC (DPU) Storage Solutions and Use Cases
Rob Davis VP Storage Technology, NVIDIA Networking Platforms
https://smartnicssummit.com/proceeding_files/a0q5f000003HkUt/20230614_B-102_Davis.PDF

Performance and Benchmarking

Metrics and Benchmarks to Assess Improvements

Evaluation of Netronome



An overview of the Netronome SmartNIC architecture.

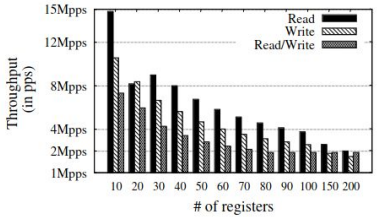
The Actual Cost of Programmable SmartNICs: diving into the existing limits.
https://doi.org/10.1007/978-3-030-75100-5_17

Performance and Benchmarking

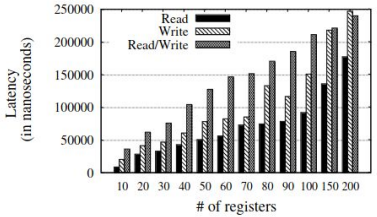
Metrics and Benchmarks to Assess Improvements

Evaluation of Netronome

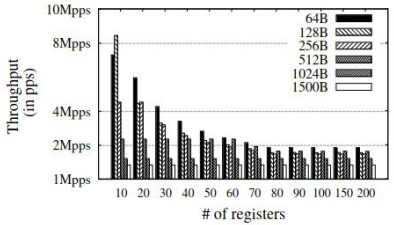
Evaluation of register access costs:



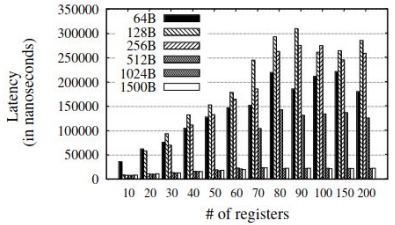
(a) Measured throughput for different register operations (packet size 64B).



(b) Measured latency for different register operations (packet size 64B).



(c) Measured throughput for different packet sizes.



(d) Measured latency for different packet sizes.

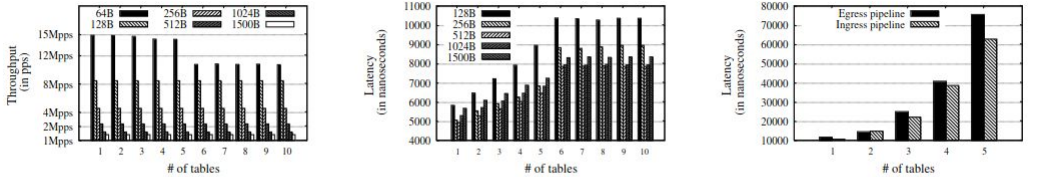
The Actual Cost of Programmable SmartNICs: diving into the existing limits.
https://doi.org/10.1007/978-3-030-75100-5_17

Performance and Benchmarking

Metrics and Benchmarks to Assess Improvements

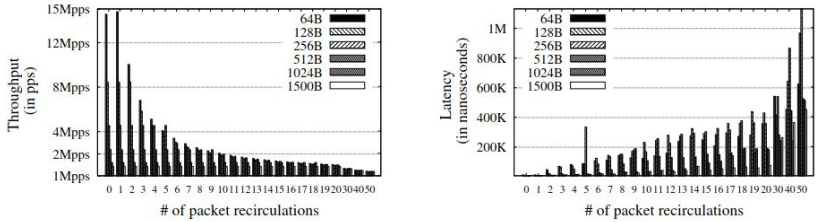
Evaluation of Netronome

Tables access:



(a) Measured throughput. (b) Measured latency. (c) Measured latency in the egress/ingress pipeline.

Recirculation



(a) Measured throughput. (b) Measured latency.

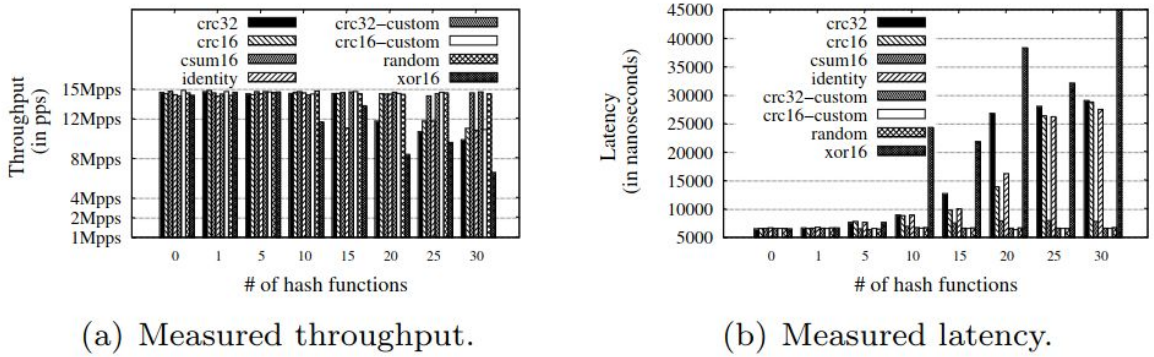
The Actual Cost of Programmable SmartNICs: diving into the existing limits.
https://doi.org/10.1007/978-3-030-75100-5_17

Performance and Benchmarking

Metrics and Benchmarks to Assess Improvements

Evaluation of Netronome

Evaluation of hash function costs:



The Actual Cost of Programmable SmartNICs: diving into the existing limits.
https://doi.org/10.1007/978-3-030-75100-5_17

Performance and Benchmarking

Metrics and Benchmarks to Assess Improvements

Evaluation of Netronome

To achieve line-rate packet processing:

- The number of register operations (up to 10 operations)
- The number of multiple match+action tables user in the pipeline (up to 5)
- The number of cryptography operations (up to 10)

Performance Characteristics of the BlueField-2 SmartNIC
https://www.osti.gov/servlets/purl/1783736

Hands-on #2 and #3

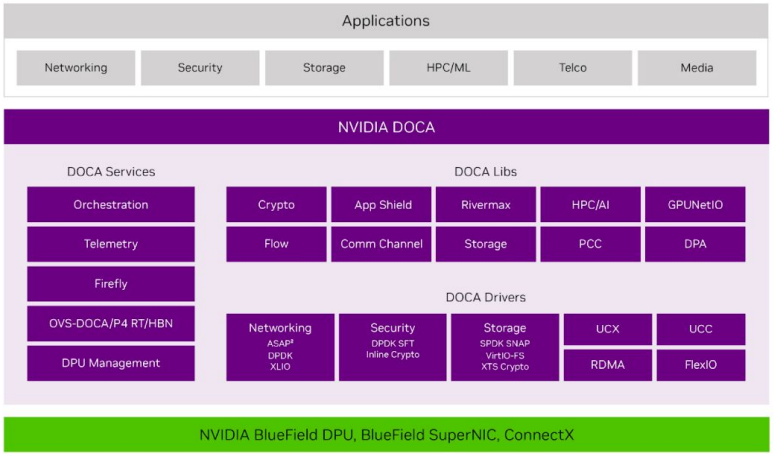
Understanding the DOCA Flow API

Nvidia DOCA Programming: Overview

NVIDIA DOCA brings together a wide range of powerful APIs, libraries, and frameworks for programming and accelerating modern data center infrastructures.

provides libraries for networking and data processing programmability that leverage NVIDIA® BlueField® DPU and NVIDIA® ConnectX® NIC hardware accelerators.

built on top of DOCA Core, which provides a unified software framework for DOCA libraries, to form a processing pipeline or workflow build of one or many DOCA libraries.

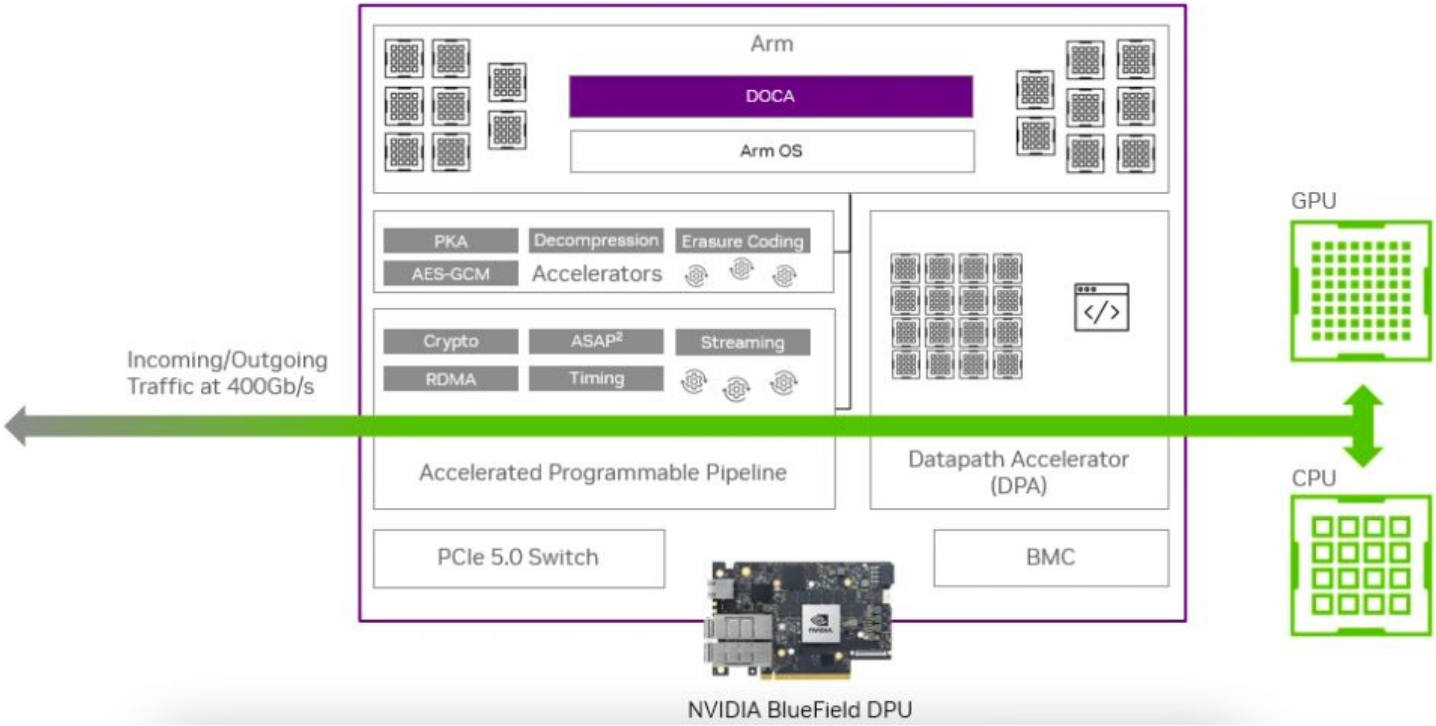


goal: learn the basics of DOCA flow, run a sample application, and offload a few features to the data path.

We are based on DOCA 2.7

https://docs.nvidia.com/doca/sdk/doca+sdk+architecture/index.html

Nvidia DOCA Programming: Overview



Nvidia DOCA Programming: Overview

Arm cores: optimized for control-path applications, general-purpose applications and single-flow performance

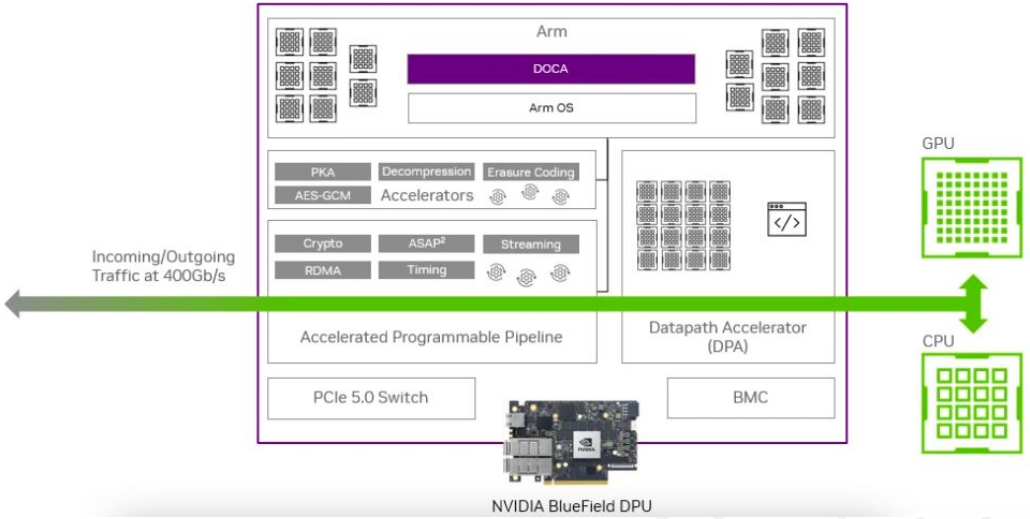
- 16 A78 Arm cores general-purpose processor
- Coherent Mesh architecture
- Last level cache (LLC)
- DDR5 memory subsystem
- Base OS and microservices

Accelerated programmable pipeline: optimized for high-performance packet processing applications and advanced packet handling

- Programmable 64-128 packet processor
- Multi-staged, highly parallelized
- Flow-based classification and action engine
- RDMA, crypto, time-based scheduling

Data-path accelerator: optimized for IO-intensive applications, high insertion rate, network flow processing, device emulation, and collective and DMA operations

- 16 hyper-threaded cores I/O and packet processor
- Real-time OS



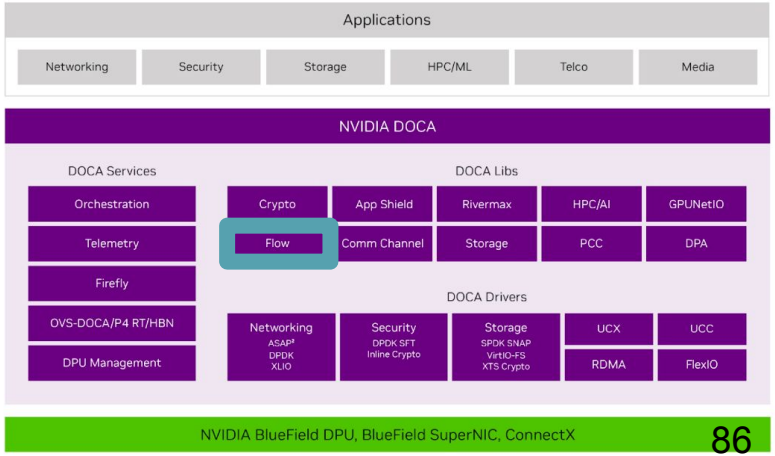
Doca Flow

DOCA Flow is the most fundamental API for building generic packet processing pipes in hardware.

The DOCA Flow library provides an API for building a set of pipes, where each pipe consists of match criteria, monitoring, and a set of actions. Pipes can be chained so that after a pipe-defined action is executed, the packet may proceed to another pipe.

Using DOCA Flow API, it is easy to develop hardware-accelerated applications that have a match on up to two layers of packets (tunneled).

1. MAC/VLAN/ETHERTYPE
2. IPv4/IPv6
3. TCP/UDP/ICMP
4. GRE/VXLAN/GTP-U/ESP/PSP
5. Metadata



Doca Flow

The execution pipe can include packet modification actions such as the following:

- Modify MAC address
- Modify IP address
- Modify L4 (ports)
- Strip tunnel
- Add tunnel
- Set metadata
- Encrypt/Decrypt

The execution pipe can also have monitoring actions such as the following:

- Count
- Policers

The pipe also has a forwarding target which can be any of the following:

- Software (RSS to subset of queues)
- Port
- Another pipe
- Drop packets

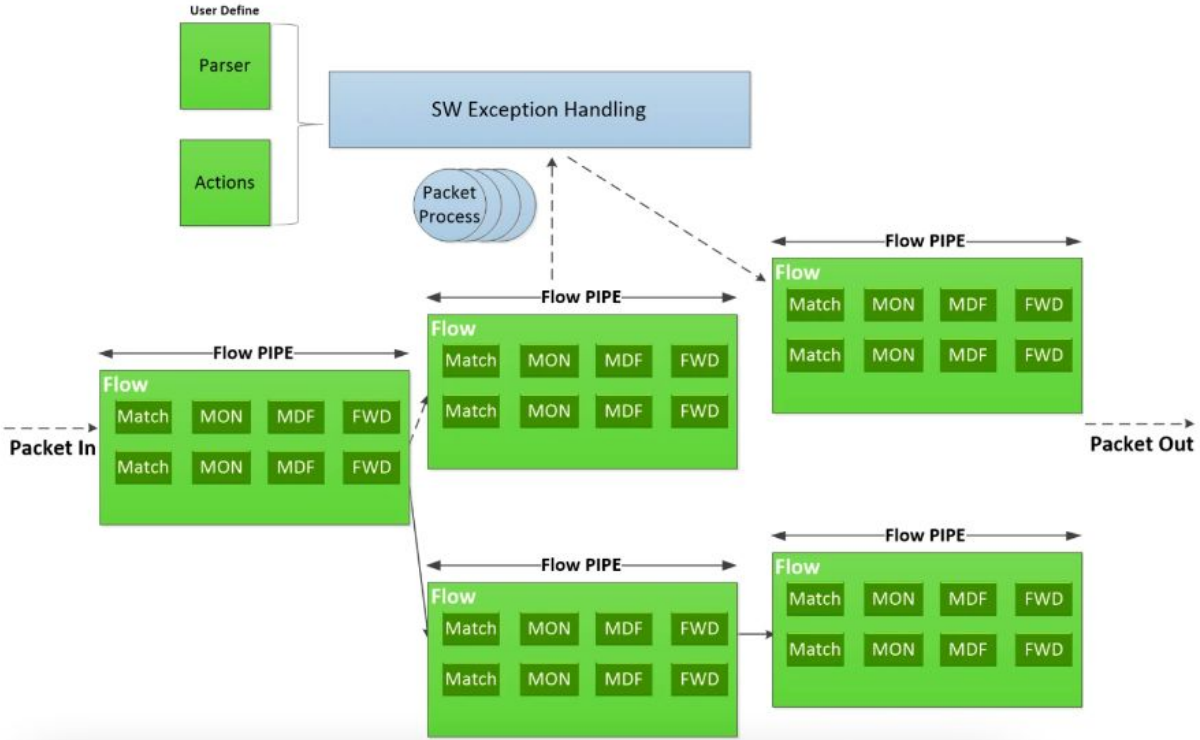
Doca Flow

A DOCA Flow-based application can run either on the host machine or on an NVIDIA BlueField DPU target.

Flow-based programs require an allocation of huge pages, hence the following commands are required:

```
# echo '1024' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
# sudo mkdir /mnt/huge
# sudo mount -t hugetlbfs nodev /mnt/huge
```

Doca Flow Architecture



Doca Flow Architecture

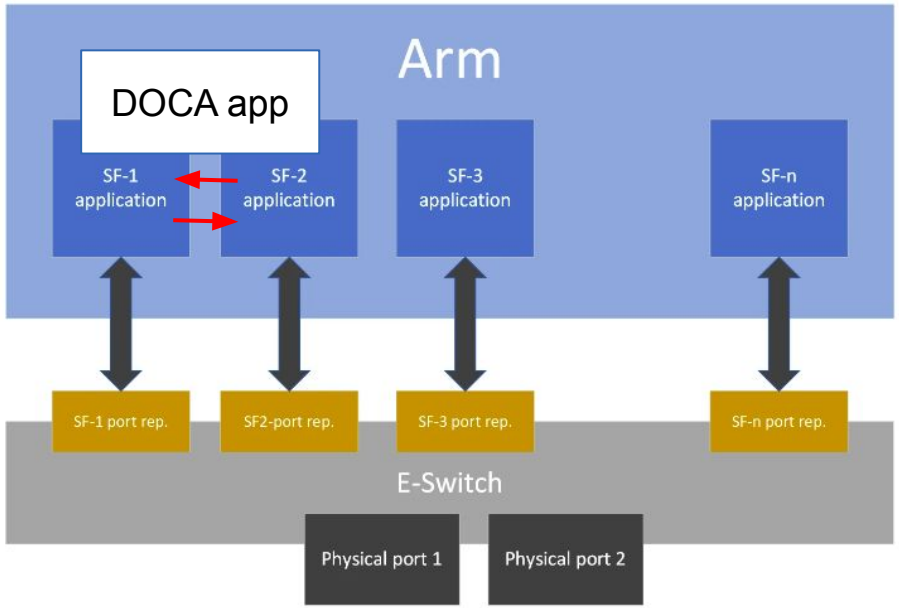
Features of DOCA Flow:

- User-defined set of matches parser and actions
- DOCA Flow pipes can be created or destroyed dynamically
- Packet processing is fully accelerated by hardware with a specific entry in a flow pipe
- Packets that do not match any of the pipe entries in hardware can be sent to Arm cores for exception handling and then reinjected back to hardware

The DOCA Flow pipe consists of the following components:

- Monitor (MON in the diagram) - counts, meters, or mirrors
- Modify (MDF in the diagram) - modifies a field
- Forward (FWD in the diagram) - forwards to the next stage in packet processing

DOCA Flow: hairpinning application



DOCA Flow: hairpinning application

```
doca_error_t
flow_hairpin(int nb_queues)
{
        int nb_ports = 2;
        struct doca_flow_resources resource = {0};
        uint32_t nr_shared_resources[DOCA_FLOW_SHARED_RESOURCE_MAX] = {0};
        struct doca_flow_port *ports[nb_ports];
        struct doca_flow_pipe *pipe;
        doca_error_t result;
        int port_id;

        result = init_doca_flow(nb_queues, "vnf,hws", resource,
                                nr_shared_resources);

        if (result != DOCA_SUCCESS) {
                DOCA_LOG_ERR("Failed to init DOCA Flow: %s",
                                doca_get_error_string(result));
                return result;
        }

        result = init_doca_flow_ports(nb_ports, ports, true);
        if (result != DOCA_SUCCESS) {
                DOCA_LOG_ERR("Failed to init DOCA ports: %s",
                             doca_get_error_string(result));
                doca_flow_destroy();
                return result;
        }
```

```
        for (port_id = 0; port_id < nb_ports; port_id++) {
                result = create_hairpin_pipe(ports[port_id],
                                             port_id, &pipe);

                if (result != DOCA_SUCCESS) {
                        stop_doca_flow_ports(nb_ports, ports);
                        doca_flow_destroy();
                        return result;
                }

                result = add_hairpin_pipe_entry(pipe,
                                                ports[port_id]);
                if (result != DOCA_SUCCESS) {
                        stop_doca_flow_ports(nb_ports, ports);
                        doca_flow_destroy();
                        return result;
                }
        }

        DOCA_LOG_INFO("Wait few seconds for packets to arrive");
        sleep(60);

        stop_doca_flow_ports(nb_ports, ports);
        doca_flow_destroy();
        return DOCA_SUCCESS;
}
```

DOCA Flow: hairpinning application

```
static doca_error_t
create_hairpin_pipe(struct doca_flow_port *port, int port_id,
struct doca_flow_pipe **pipe)
{
        struct doca_flow_match match;
        struct doca_flow_actions actions,
                                 *actions_arr[NB_ACTIONS_ARR];
        struct doca_flow_fwd fwd;
        struct doca_flow_pipe_cfg pipe_cfg;

        memset(&match, 0, sizeof(match));
        memset(&actions, 0, sizeof(actions));
        memset(&fwd, 0, sizeof(fwd));
        memset(&pipe_cfg, 0, sizeof(pipe_cfg));

        pipe_cfg.attr.name = "HAIRPIN_PIPE";
        pipe_cfg.attr.type = DOCA_FLOW_PIPE_BASIC;
        pipe_cfg.match = &match;
        actions_arr[0] = &actions;
        pipe_cfg.actions = actions_arr;
        pipe_cfg.attr.is_root = true;
        pipe_cfg.attr.nb_actions = NB_ACTIONS_ARR;
        pipe_cfg.port = port;
```

```
        match.outer.l4_type_ext = DOCA_FLOW_L4_TYPE_EXT_UDP;
        match.outer.l3_type = DOCA_FLOW_L3_TYPE_IP4;
        match.outer.ip4.src_ip = 0xffffffff;
        match.outer.ip4.dst_ip = 0xffffffff;

        match.outer.udp.l4_port.src_port = 0xffff;
        match.outer.udp.l4_port.dst_port = 0xffff;

        /* forwarding traffic to other port */
        fwd.type = DOCA_FLOW_FWD_PORT;
        fwd.port_id = port_id ^ 1;

        return doca_flow_pipe_create(&pipe_cfg, &fwd, NULL
                                                    , pipe);
}
```

DOCA Flow: hairpinning application

```
static doca_error_t
add_hairpin_pipe_entry(struct doca_flow_pipe *pipe, struct
doca_flow_port *port)
{
        struct doca_flow_match match;
        struct doca_flow_actions actions;
        struct doca_flow_pipe_entry *entry;
        struct entries_status status;
        doca_error_t result;
        int num_of_entries = 1;

        /* example 5-tuple to forward */
        doca_be32_t dst_ip_addr = BE_IPV4_ADDR(8, 8, 8, 8);
        doca_be32_t src_ip_addr = BE_IPV4_ADDR(1, 2, 3, 4);
        doca_be16_t dst_port = rte_cpu_to_be_16(80);
        doca_be16_t src_port = rte_cpu_to_be_16(1234);

        memset(&status, 0, sizeof(status));
        memset(&match, 0, sizeof(match));
        memset(&actions, 0, sizeof(actions));
        match.outer.ip4.dst_ip = dst_ip_addr;
        match.outer.ip4.src_ip = src_ip_addr;
        match.outer.tcp.l4_port.dst_port = dst_port;
        match.outer.tcp.l4_port.src_port = src_port;

        result = doca_flow_pipe_add_entry(0, pipe, &match,
                        &actions, NULL, NULL, 0, &status, &entry);
        if (result != DOCA_SUCCESS)
                return result;

        result = doca_flow_entries_process(port, 0,
                                DEFAULT_TIMEOUT_US, num_of_entries);
        if (result != DOCA_SUCCESS)
                return result;

        if (status.nb_processed != num_of_entries || status.failure)
                return DOCA_ERROR_BAD_STATE;

        return DOCA_SUCCESS;
}
```

DOCA Flow: hairpinning application

SMARTNESS

Compiling and running

```
// build and compile
# meson build
# cd build
# ninja

// run the DOCA app passing the right SF interfaces
# ./doca_flow_hairpin -a auxiliary:mlx5_core.sf.4,dv_flow_en=2 -a
auxiliary:mlx5_core.sf.5,dv_flow_en=2 -- -l 60
```

Break 2: DOCA offloading

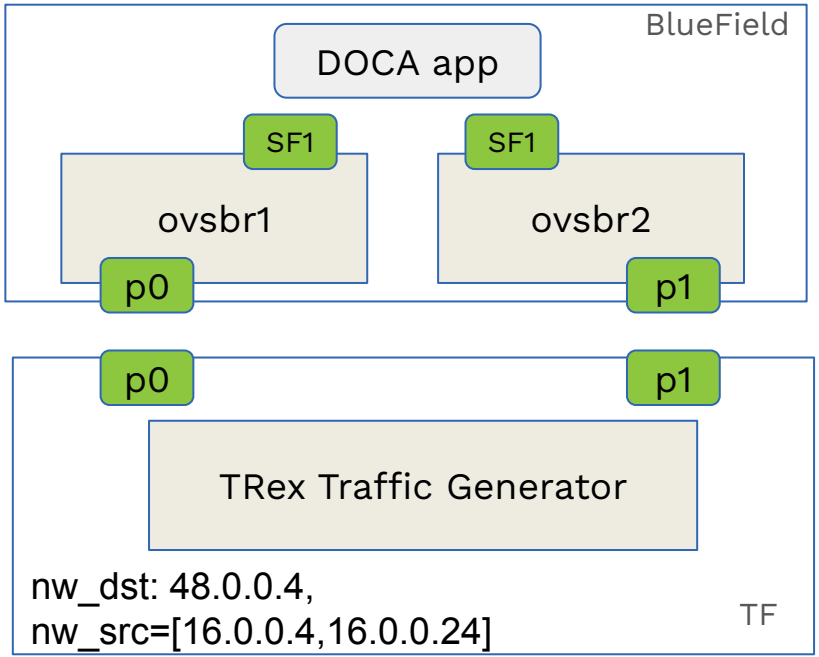
Compiling/running first DOCA application

- Goal 1: Compiling/running DOCA application.
- Goal 2: Deploy the application with specific src/dst IP
- Goal 3: Use traffic generator test application

Scan me!

Access our GitHub:
https://github.com/smartness2030/netsoft24-smartnic-tutorial

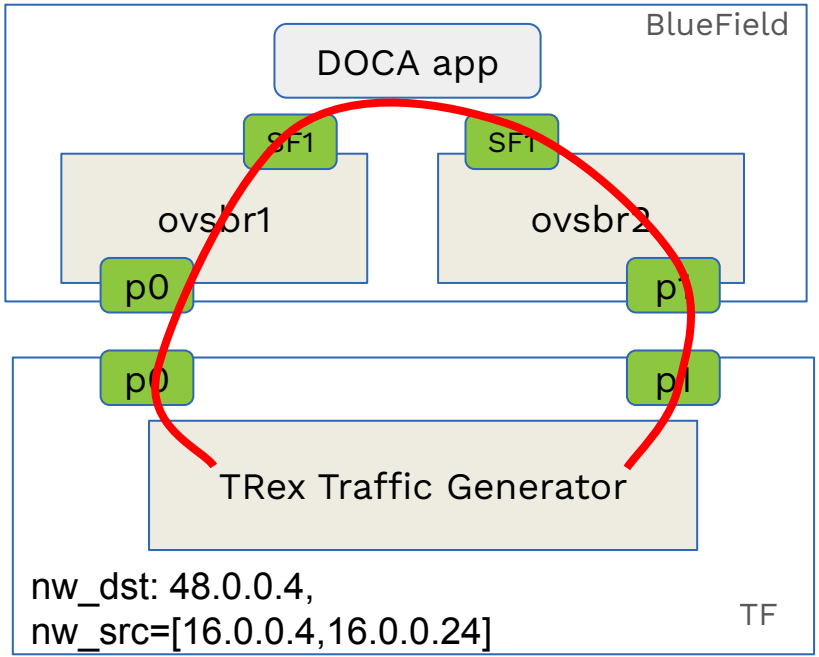
Break 2: DOCA offloading



Scan me!

Access our GitHub:
https://github.com/smartness2030/netsoft24-smartnic-tutorial

Break 2: DOCA offloading



Scan me!

Access our GitHub:
https://github.com/smartness2030/netsoft24-smartnic-tutorial

Break 2: DOCA offloading (step-by-step)

1) Let's **recreate two SFs.** Each SF is attached to a different OVS bridge.

2) Export DOCA path:

```
export PKG_CONFIG_PATH=:/opt/mellanox/doca/lib/aarch64-linux-gnu/pkgconfig:/opt/mellanox/dpdk/lib/aarch64-linux-gnu/pkgconfig:/opt/mellanox/flexio/lib/pkgconfig`
```

3) **Clone** this repository with the sample DOCA code.

```
git clone https://github.com/smartness2030/netsoft24-smartnic-tutorial.git
```

4) **Change the file code** flow_hairpin_vnf_sample.c so as to match in one of the subnets. Configure OVS flow rules to make the app receive packets.

5) **Compile**, and run:

```
meson build; cd build; ninja
```

6) **Add** OVS flow rules

```
sudo ovs-ofctl add-flow ovsbr1 ip,nw_dst=48.0.0.4,nw_src=16.0.0.4,in_port=p0,actions=output:en3f0pf0sf11
sudo ovs-ofctl add-flow ovsbr1 ip,nw_dst=48.0.0.4,nw_src=16.0.0.4,in_port=en3f0pf0sf11,actions=output:p0
sudo ovs-ofctl add-flow ovsbr2 ip,nw_dst=48.0.0.4,nw_src=16.0.0.4,in_port=en3f1pf1sf10,actions=output:p1
sudo ovs-ofctl add-flow ovsbr2 ip,nw_dst=48.0.0.4,nw_src=16.0.0.4,in_port=p1,actions=output:en3f1pf1sf10
```

7) **Execute** ./doca_flow_hairpin_vnf -a auxiliary:mlx5_core.sf.3,dv_flow_en=2 -a auxiliary:mlx5_core.sf.5,dv_flow_en=2 --file-prefix mylog -- -l 60

Break 3: DOCA offloading

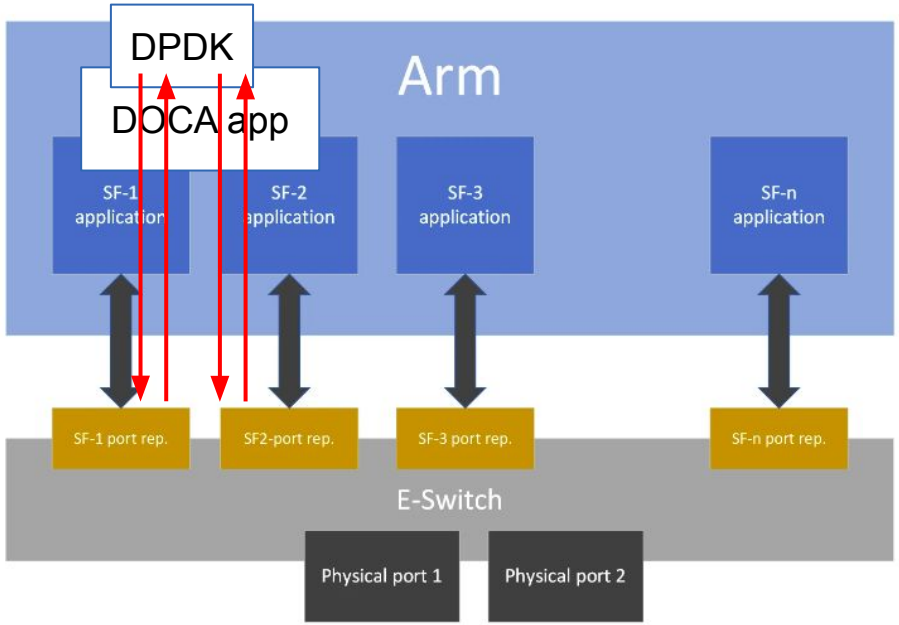
Extending the DOCA application to run on ARM processors

- Goal 1: Understand how to create a DOCA pipe that do packet processing in the ARM processor of the DPU
- Goal 2: Use traffic generator to test the application
- Goal 3: Count different flows src/dst using ARM cores

Scan me!

Access our GitHub:
https://github.com/smartness2030/netsoft24-smartnic-tutorial

DOCA Flow: RSS (DOCA_FLOW_FWD_RSS)



DOCA Flow: RSS (DOCA_FLOW_FWD_RSS)

```
doca_error_t
flow_hairpin(int nb_queues)
{
        int nb_ports = 2;
        struct doca_flow_resources resource = {0};
        uint32_t nr_shared_resources[DOCA_FLOW_SHARED_RESOURCE_MAX] = {0};
        struct doca_flow_port *ports[nb_ports];
        struct doca_flow_pipe *pipe;
        doca_error_t result;
        int port_id;

        result = init_doca_flow(nb_queues, "vnf,hws", resource,
                                nr_shared_resources);

        if (result != DOCA_SUCCESS) {
                DOCA_LOG_ERR("Failed to init DOCA Flow: %s",
                                doca_get_error_string(result));
                return result;
        }

        result = init_doca_flow_ports(nb_ports, ports, true);
        if (result != DOCA_SUCCESS) {
                DOCA_LOG_ERR("Failed to init DOCA ports: %s",
                             doca_get_error_string(result));
                doca_flow_destroy();
                return result;
        }
```

```
        for (port_id = 0; port_id < nb_ports; port_id++) {
                result = create_hairpin_pipe(ports[port_id],
                                             port_id, &pipe);

                if (result != DOCA_SUCCESS) {
                        stop_doca_flow_ports(nb_ports, ports);
                        doca_flow_destroy();
                        return result;
                }

                result = add_hairpin_pipe_entry(pipe,
                                                ports[port_id]);
                if (result != DOCA_SUCCESS) {
                        stop_doca_flow_ports(nb_ports, ports);
                        doca_flow_destroy();
                        return result;
                }
        }

        DOCA_LOG_INFO("Wait few seconds for packets to arrive");

        while(1){
                for (port_id = 0; port_id < nb_ports; port_id++)
                        process_packets(port_id);
        }

        stop_doca_flow_ports(nb_ports, ports);
        doca_flow_destroy();
        return DOCA_SUCCESS;
}
```

DOCA Flow: RSS (DOCA_FLOW_FWD_RSS)

```
static doca_error_t create_rss_meta_pipe(struct doca_flow_port
*port, struct doca_flow_pipe **pipe)
{
        struct doca_flow_match match;
        struct doca_flow_actions actions,
*actions_arr[NB_ACTIONS_ARR];
        struct doca_flow_fwd fwd;
        struct doca_flow_fwd fwd_miss;
        struct doca_flow_pipe_cfg *pipe_cfg;
        uint16_t rss_queues[1];
        doca_error_t result;

        memset(&match, 0, sizeof(match));
        memset(&actions, 0, sizeof(actions));
        memset(&fwd, 0, sizeof(fwd));
        memset(&fwd_miss, 0, sizeof(fwd_miss));

        /* set mask value */
        actions.meta.pkt_meta = UINT32_MAX;
        actions_arr[0] = &actions;

        /* 5 tuple match */
        match.parser_meta.outer_l4_type = DOCA_FLOW_L4_META_UDP;
        match.parser_meta.outer_l3_type = DOCA_FLOW_L3_META_IPV4;
        match.outer.l4_type_ext = DOCA_FLOW_L4_TYPE_EXT_UDP;
        match.outer.l3_type = DOCA_FLOW_L3_TYPE_IP4;
        match.outer.ip4.src_ip = 0xffffffff;
        match.outer.ip4.dst_ip = 0xffffffff;
        match.outer.tcp.l4_port.src_port = 0xffff;
        match.outer.tcp.l4_port.dst_port = 0xffff;
```

```
        result = doca_flow_pipe_cfg_create(&pipe_cfg, port);
        …

        result = set_flow_pipe_cfg(pipe_cfg, "RSS_META_PIPE",
DOCA_FLOW_PIPE_BASIC, true);
        …
        result = doca_flow_pipe_cfg_set_match(pipe_cfg, &match,
NULL);
        …
        result = doca_flow_pipe_cfg_set_actions(pipe_cfg,
actions_arr, NULL, NULL, NB_ACTIONS_ARR);
        …

        /* RSS queue - send matched traffic to queue 0  */
        rss_queues[0] = 0;
        fwd.type = DOCA_FLOW_FWD_RSS;
        fwd.rss_queues = rss_queues;
        fwd.rss_inner_flags = DOCA_FLOW_RSS_IPV4 | DOCA_FLOW_RSS_UDP;
        fwd.num_of_queues = 1;

        fwd_miss.type = DOCA_FLOW_FWD_DROP;

        result = doca_flow_pipe_create(pipe_cfg, &fwd, &fwd_miss, pipe);

        doca_flow_pipe_cfg_destroy(pipe_cfg);
        return result;
}
```

DOCA Flow: RSS (DOCA_FLOW_FWD_RSS)

```
static doca_error_t add_rss_meta_pipe_entry(struct doca_flow_pipe
*pipe, struct entries_status *status)
{
        struct doca_flow_match match;
        struct doca_flow_actions actions;
        struct doca_flow_pipe_entry *entry;
        doca_error_t result;

        /* example 5-tuple to drop */
        doca_be32_t dst_ip_addr = BE_IPV4_ADDR(48, 0, 0, 4);
        doca_be32_t src_ip_addr = BE_IPV4_ADDR(16, 0, 0, 4);
        doca_be16_t dst_port = rte_cpu_to_be_16(12);
        doca_be16_t src_port = rte_cpu_to_be_16(1025);

        memset(&match, 0, sizeof(match));
        memset(&actions, 0, sizeof(actions));

        match.outer.ip4.dst_ip = dst_ip_addr;
        match.outer.ip4.src_ip = src_ip_addr;
        match.outer.tcp.l4_port.dst_port = dst_port;
        match.outer.tcp.l4_port.src_port = src_port;

        /* set meta value */
        actions.meta.pkt_meta = 10;
        actions.action_idx = 0;

        result = doca_flow_pipe_add_entry(0, pipe, &match,
                        &actions, NULL, NULL, 0, status, &entry);
        …
}
```

```
 static void
process_packets(int ingress_port)
{
        struct rte_mbuf *packets[PACKET_BURST];
        int queue_index = 0;
        int nb_packets;
        int i;

        nb_packets = rte_eth_rx_burst(ingress_port, queue_index,
                                      packets, PACKET_BURST);

        /* do whatever we want with packets


        rte_eth_tx_burst(ingress_port,queue_index,
                                     packets,nb_packets);
}
```

Challenges and Future Trends

- Current Limitations and Open Issues
- Research and Development Trends

Challenges and Future Trends

There is no perfect solution, and SmartNICs struggle with their weaknesses too.

- **Standartization of SmartNICs.** Despite existing efforts towards defining a common NIC architecture (e.g. PNA) and programming framework such as P4 language, we are still far from having standard programming interfaces fully available in all SmartNICs.
- **Variable Performance.** Packet processing performance in SmartNIC is subject to significant variation across programs, traffic types, and table entries -- and, therefore, line-speed processing is not an automatic guarantee.
- **Limited Resource Orchestration.** Little has yet been done to provide a unified resource orchestration layer for SmartNICs, allowing to run multiple isolated applications on top of the same platform.
- **Limited Programmability.** Each architecture provides a set of programmable primitives and constraints.
- **Application offloading.** SmartNICs and hardware accelerations enable a plethora of system optimizations. However, it is still not clear which application to offload and, more importantly, which parts of the application to offload.

Challenges and Future Trends

Research and Development Trends

Below, we have identified some areas and questions that require further study:

- Offloading control-intensive complex workloads to SmartNICs a promising path?
- One can consider processors in a Smart NIC as extra heterogeneous cores in a system. What extra benefits do we get by putting these extra cores into the NIC (in contrast to putting them close to storage or CPU)?
- What database operations should be pushed to SmartNIC?

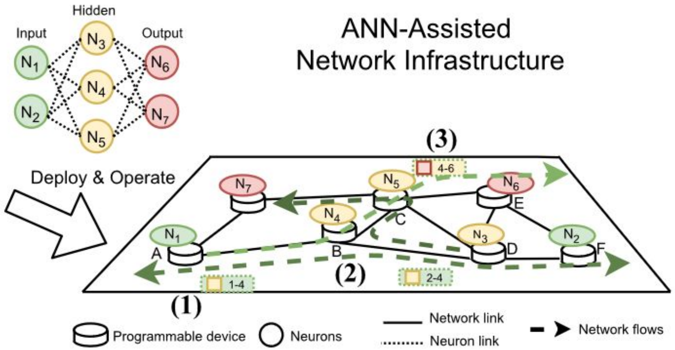
These are some of the things we can research about SmartNIC, with time more things will emerge, and these case studies will increase

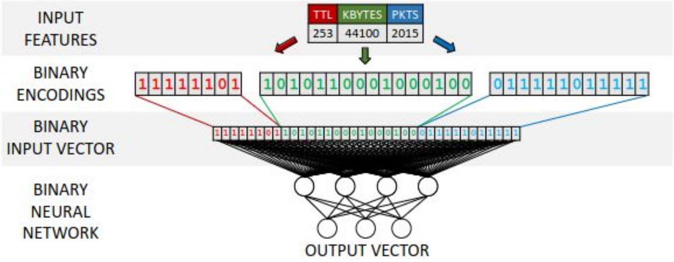
https://www.cs.cornell.edu/~mt822/docs/smartNICs_lecture.pdf

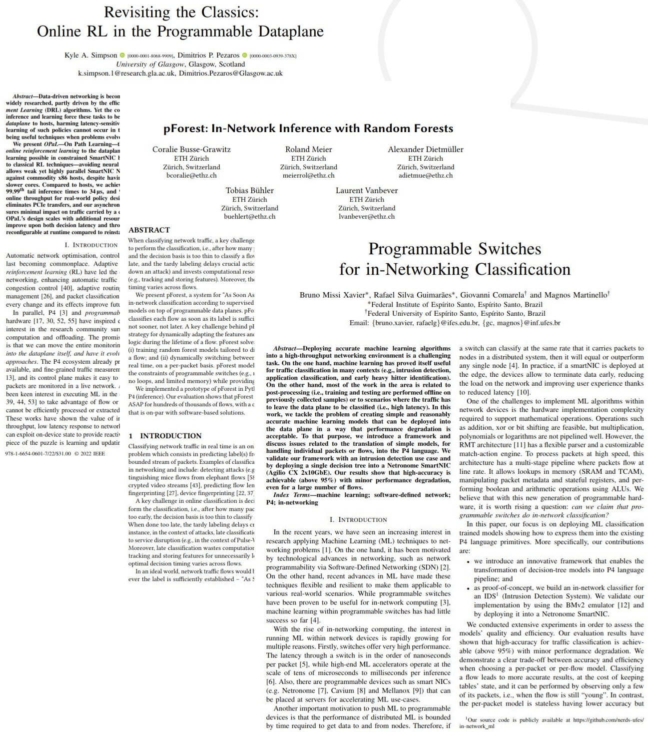
https://pages.cs.wisc.edu/~yxy/cs839-s20/slides/L17.pdf

Research and Development Trends

Applying in-network AI







Revisiting the Classics: Online RL in the Programmable Dataplane

Kyle A. Simpson, Dimitrios P. Pezaros
University of Glasgow, Glasgow, Scotland
k.simpson.1@research.gla.ac.uk, Dimitrios.Pezaros@Glasgow.ac.uk

pForest: In-Network Inference with Random Forests

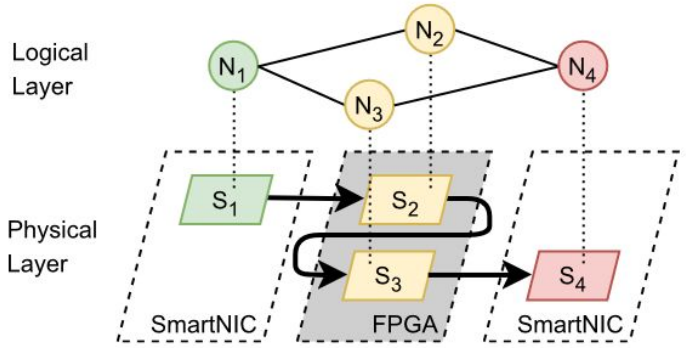
Coralie Busse-Grawitz, ETH Zürich, Zürich, Switzerland, bcoralie@ethz.ch
Roland Meier, ETH Zürich, Zürich, Switzerland, meierrol@ethz.ch
Alexander Dietmüller, ETH Zürich, Zürich, Switzerland, adietmue@ethz.ch
Tobias Bühler, ETH Zürich, Zürich, Switzerland, buehlert@ethz.ch
Laurent Vanbever, ETH Zürich, Zürich, Switzerland, lvanbever@ethz.ch

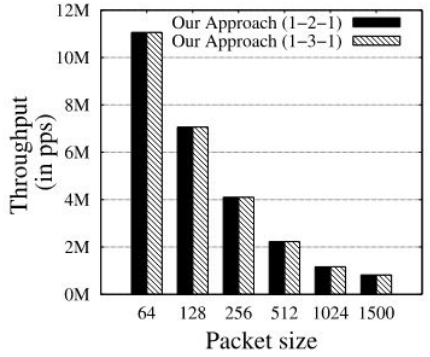
Programmable Switches for in-Networking Classification

Bruno Missi Xavier*, Rafael Silva Guimarães*, Giovanni Comarela† and Magnos Martinello†
*Federal Institute of Espírito Santo, Espírito Santo, Brazil
†Federal University of Espírito Santo, Espírito Santo, Brazil
Email: {bruno.xavier, rafaelg}@ifes.edu.br, {gc, magnos}@inf.ufes.br

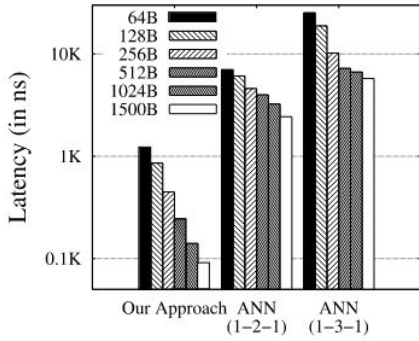
Research and Development Trends

Applying in-network AI: Toward In-Network Intelligence





(c) Physical testbed throughput.



(d) Physical testbed latency.

Toward In-Network Intelligence: Running Distributed Artificial Neural Networks in the Data Plane

https://doi.org/10.1109/LCOMM.2021.3108940

Conclusion & Closing Remarks

- Key Takeaways
- SmartNICs`s Impact in Networking
- Further Exploration and Adoption

Conclusion & Closing Remarks

Key Takeaways

As the demand for high-speed data processing continues to grow exponentially, SmartNIC technology has paved the way for significant advancements in networking and data processing for AI training and cloud computing. These specialized network interface cards have proven to be instrumental in overcoming the challenges of modern computing, enabling efficient data offloading, acceleration of critical tasks, and seamless integration with existing infrastructure.

With ongoing research and development, we can expect further innovations in SmartNIC technology, unlocking new possibilities and driving the next wave of advancements in networking and data-driven applications.

Everyone are doing SmartNICs today!
Evolution, not revolution

Conclusion & Closing Remarks

SmartNICs`s Impact in Network Technology

SmartNICs utilize additional computational resources to process network traffic upon both entry and exit from the server. Moreover, they alleviate the burden on the host CPU at the application level by offloading specific tasks.

SmartNICs have had an impact across various areas of technology, including:

- Bumping up the computing power
- Offloading
- Storage control with a SmartNIC
- Electronic trading
- Security
- Acceleration

Conclusion & Closing Remarks

Further Exploration and Adoption

The future seems bright for SmartNICs. In the last years most companies in the sector announced their first line of products, e.g. Nvidia, Intel, Broadcom, Silicom, Inventec, AMD. In the future the development will be driven by even faster network speeds. E.g. Nvidia plans to increase the throughput of their devices to 400 Gbps including a 100-times performance improvement by 2023.

https://medium.com/@rfadda741254/smart-nic-market-size-growth-forecast-2023-2030-83f8c0508a8a
https://semiengineering.com/the-rise-of-smartnics/
https://www.smallake.kr/wp-content/uploads/2021/11/NET-2021-05-1_05.pdf

Conclusion & Closing Remarks

Further Exploration and Adoption

According to market research firm Dell'Oro Group, the SmartNIC market is forecast to surpass $600M and comprise 23% of the worldwide Ethernet adapter market by 2024, and we've seen a new generation of SmartNICs from companies like Broadcom, Intel, Mellanox, and Xilinx.

In conclusion, the Smart NIC market is poised for significant growth due to the increasing demand for high-performance networking in data-intensive applications, virtualized environments, and edge computing. With their ability to offload network processing tasks and enhance overall server performance, Smart NICs are expected to play a vital role in the future of network infrastructure.

https://medium.com/@rfadda741254/smart-nic-market-size-growth-forecast-2023-2030-83f8c0508a8a
https://semiengineering.com/the-rise-of-smartnics/
https://www.smallake.kr/wp-content/uploads/2021/11/NET-2021-05-1_05.pdf

SmartNIC - the next leap in networking

Questions?

Next...

Hands-on experience with NVIDIA Bluefield-2

Access our github

https://github.com/smartness2030/netsoft-smartnic-tutorial/

ACKNOWLEDGMENTS / DISCLAIMER

ACKNOWLEDGMENTS / DISCLAIMER

This work has been performed within the framework of the FAPESP Engineering Research Center (ERC) Program under FAPESP grant agreement #2021/00199-8 (SMARTNESS).