

The Hidden Cost of Heterogeneity: Navigating Internal Contention in SoC SmartNICs

SIQI LIU, University of Waterloo, Canada

MINA TAHMASBI ARASHLOO, University of Waterloo, Canada

RAOUF BOUTABA, University of Waterloo, Canada

SmartNICs promise flexible, high-performance offload by composing diverse hardware accelerators on a single chip. However, real deployments often require multiple components to execute concurrently, leading to complex interactions that are not captured by existing models and benchmarks. This paper presents the first systematic study of internal resource contention in SmartNIC datapaths. Using the BlueField-3 as a representative example, we show that contention is pervasive yet largely difficult to anticipate from the developer-visible architecture, highly sensitive to configuration parameters, and can produce non-intuitive performance outcomes when features are composed. These effects lead to severe degradation in realistic services, including over 50% throughput loss in a storage workload and a 4× gap between standalone and deployed accelerator performance. We distill these findings into four recurring contention patterns, only one of which is predictable from the device’s visible architecture, and provide concrete guidelines for both developers and NIC vendors to build predictable and efficient SmartNIC services.

CCS Concepts: • **Networks** → **Network performance analysis**; **Network adapters**; *Network measurement*; • **Hardware** → *Networking hardware*.

Additional Key Words and Phrases: SmartNICs, DPU, resource contention, performance characterization, hardware offloading, heterogeneous architectures, feature composition

ACM Reference Format:

Siqi Liu, Mina Tahmasbi Arashloo, and Raouf Boutaba. 2026. The Hidden Cost of Heterogeneity: Navigating Internal Contention in SoC SmartNICs. *Proc. ACM Meas. Anal. Comput. Syst.* 10, 3, Article 81 (December 2026), 27 pages. <https://doi.org/10.1145/3857384>

1 Introduction

To support the exploding demands of diverse novel workloads, such as artificial intelligence and cloud-native applications, data center operators increasingly deploy System-on-Chip (SoC) SmartNICs to offload infrastructure services and free host resources [4, 36, 37]. To deliver the performance and flexibility that modern services require, SoC-based SmartNICs adopt a heterogeneous architecture, in which various programmable processors, as well as fixed-function accelerators for primitives such as cryptography, packet steering, compression, and regular expression matching, are integrated on a common on-chip substrate.

While individual hardware units, i.e., processors and accelerators, are highly efficient for isolated workloads, practical offloaded services typically use multiple hardware units on the Network Interface Card (NIC). This multi-unit execution forces components to share critical on-chip resources, such as memory, internal interconnects, and switching fabric, creating complex interactions whose

Authors’ Contact Information: Siqi Liu, tyler.liu@uwaterloo.ca, University of Waterloo, Waterloo, ON, Canada; Mina Tahmasbi Arashloo, University of Waterloo, Waterloo, ON, Canada, mina.arashloo@uwaterloo.ca; Raouf Boutaba, University of Waterloo, Waterloo, ON, Canada, rboutaba@uwaterloo.ca.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2476-1249/2026/12-ART81

<https://doi.org/10.1145/3857384>

performance implications are not captured by single-function benchmarks. As a result, the performance of services can diverge from expectations built on isolated component behavior, potentially undermining the flexible acceleration promise of SmartNICs. This problem will only compound as future hardware generations integrate more diverse accelerators for more capabilities.

Despite the increasing need to reason about such multi-unit executions, existing work on SmartNIC performance largely abstracts away these interactions. Early studies on modeling SmartNIC performance [9, 10, 39] model devices and shared resources as independent processing nodes connected by constant throughput links, omitting shared-resource effects. Similarly, prior analyses [7, 38] and optimization frameworks [19, 40] focus on specific SmartNIC components or performance aspects in isolation, without considering how they interact when composed within an end-to-end service. Furthermore, these approaches do not capture the cross-component interactions that arise when services concurrently utilize multiple acceleration primitives.

In this paper, we address this gap by conducting the first systematic study of internal resource contention in the SmartNIC datapath, revealing the complexity of the interactions and the resulting contentions previously overlooked in SmartNIC performance analysis. Using the BlueField-3 SmartNIC as an example, we empirically characterize the existence, conditions, and root causes of cross-component contention to demonstrate how these interactions can lead to severe and non-intuitive performance degradation under realistic workloads. We distill our findings into four recurring contention patterns and actionable guidelines for both SmartNIC service developers and SmartNIC vendors. Our study reveals the following key observations.

Contention is pervasive among SmartNIC hardware components. Contention arises in every subsystem we exercise: lookaside accelerator invocations degrade under network traffic, the datapath accelerator's memory access and network I/O suffer under cross traffic from nearly every direction, and the NIC subsystem exhibits contention under feature composition. These effects surface even between workloads that appear entirely decoupled at the programming interface.

Feature interactions produce unexpected performance effects. When multiple acceleration primitives are composed, their interactions can produce distinct and often non-intuitive performance outcomes, and the combined cost of composed features is not always the sum of their individual costs. For example, when combining packet cloning with drops, which copy is discarded changes the cost of cloning, and composing packet cloning with IPsec ESP anti-replay, an inline offload that adds no overhead on its own, triggers a severe throughput collapse.

Most contention cannot be anticipated from the visible architecture and API documentation. Each workload leaves a footprint on the internal fabric. This includes the set of internal links its traffic occupies, which depends on where its corresponding processing units and accelerators sit in the topology as well as the data and control streams the hardware generates to serve their operations. For most of the contention we observe, this footprint, as well as the arbitration strategy when footprints overlap, is not visible from high-level architectures and available documentation. Developers can impact some of these effects through (indirect) configuration parameters such as queue and thread counts and message sizes, but only after discovering them through measurement.

To understand how these contention patterns translate into end-to-end behavior, we evaluate three representative SmartNIC services. These case studies illustrate how the low-level interactions identified above directly shape application-level performance. In a distributed file system, for example, enabling AES-XTS encryption acceleration under concurrent traffic incurs an over 50% throughput loss compared to an equivalent AES-GCM accelerator version. In a distributed machine learning parameter server, offloading tiered parameter aggregation to a datapath accelerator yields a net throughput gain of only 10 Gbps, 25.2% of the over 40 Gbps the accelerator achieves standalone. Finally, a network intrusion detection scenario exhibits composition-induced degradation: cloning-based traffic sampling costs the host path 15–30%, IPsec ESP anti-replay costs nothing on its own,

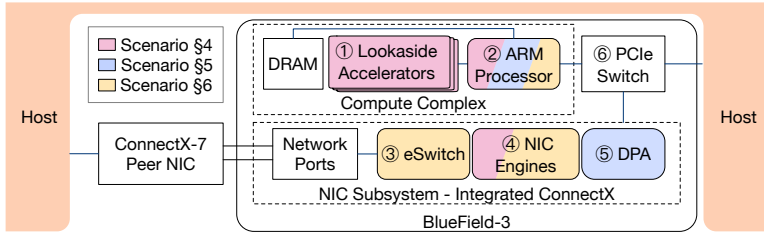


Fig. 1. A high-level overview of the BlueField-3 architecture and our testbed. Our three representative acceleration scenarios collectively cover the main architectural components.

yet enabling the two together raises the loss to about 70%.

Across these studies, we identify four recurring contention patterns, only one of which is predictable from the visible architecture alone. Building on these patterns, we provide a four-step contention-aware development guideline for SmartNIC service developers. We also discuss what information about internal resource usage NIC vendors could expose to support contention analysis, including which internal paths an operation's traffic traverses, how arbitration allocates shared resources, and how composed features interact. Such support would enable developers to reason about cross-component interactions and more effectively utilize the device's performance potential. Finally, although our findings are grounded in BlueField-3 measurements, they arise from architectural properties common to SoC-based SmartNICs, and we discuss how the patterns and guidelines apply to other SmartNICs as well.

Contributions. We make the following contributions:

- (1) A systematic characterization of internal resource contention across a representative SoC-based SmartNIC datapath, identifying the existence, root causes, and severity of cross-component interference through microbenchmarks.
- (2) Validation of these contention patterns through three representative services on SmartNICs, demonstrating that the contention observed in microbenchmarks arises in realistic services and degrades their end-to-end performance.
- (3) Four recurring contention patterns distilled from these measurements, only one of which is predictable from the visible architecture, their inferred causes and available mitigations.
- (4) A four-step contention-aware development guideline for SmartNIC service developers, and a starting point for what NIC vendors could expose to support contention analysis.

The code and reproduction instructions are available on GitHub: <https://github.com/tylerliu/snic-contention-measurement>.

2 Smart Network Interface Cards

SmartNICs augment traditional NICs with programmable accelerators for network-related processing. Sitting between the server and the network, they support a range of *services*: near-network processing for the host (e.g., the network stack and network-intensive distributed applications), and data-exchange inspection and enforcement for the data-center operator (e.g., hypervisors).

SoC-based SmartNICs. To accommodate varying requirements for performance and flexibility, SmartNIC architectures employ different combinations of fixed-function ASICs, FPGAs, specialized packet processing pipelines, and general-purpose processors. This results in a heterogeneous architecture where each hardware unit handles the tasks best suited to its performance-flexibility trade-offs. Packaged with auxiliary memory and datapaths onto a single chip, they form a System-on-Chip (SoC), commercially also called Data Processing Units (DPUs) or Infrastructure Processing Units (IPUs). Several SoC-based SmartNIC models are currently deployed in data centers, including

the NVIDIA BlueField [24], the AMD Pensando DPU [3], the Intel ASIC-based IPU [11], the Marvell LiquidIO [21], and presumably, the AWS Nitro Card [5].

Common architectural patterns. Architecturally, many SoC-based SmartNICs share a common high-level architectural pattern comprising a packet-processing network (NIC) subsystem and a general-purpose compute complex. The two are bridged by an internal fabric, such as a PCIe channel bus, and augmented by specialized hardware accelerators deployed either inline on the network datapath or as lookaside engines within the compute complex. Models differ in how these blocks are placed and interconnected: for example, the Intel ASIC IPU and AMD Pensando DPU center their NIC subsystem on a P4 packet-processing pipeline with tailored protocol engines, and the Pensando DPU also disaggregates the compute complex and connects components to the NIC pipeline and host interface through a Network-on-Chip (NoC) fabric. §10 provides a more detailed comparison of these SmartNIC architectures.

BlueField-3 as a representative architecture. This paper focuses on the NVIDIA BlueField-3, a state-of-the-art SmartNIC that is highly representative of the above architectural paradigm of SoC-based SmartNICs. As illustrated in Figure 1, the device hardware is partitioned into a NIC subsystem and an ARM compute complex, interconnected via an internal 512 Gbps PCIe 5.0 bus and switch ⑥ that also interfaces with the host system(s).

The NIC subsystem, based on the ConnectX-7 architecture, has an ASAP² embedded switch ③, dedicated hardware for inline protocol processing ④, and a Data-Path Accelerator (DPA) ⑤. DPA is a specialized 16-core, 256-thread RISC-V processor designed for multi-threaded I/O-intensive acceleration. Connected to the network-host datapath via an internal PCIe switch, the compute complex centers on a 16-core Armv8.2+ A78 ARM processor ② for both operational and management tasks, supported by 32 GB of DDR5 memory, 128 GB of SSD storage, and various integrated lookaside accelerators ①. BlueField-3 also has various fixed-function acceleration engines for networking, security, storage, management, and HPC/AI acceleration tasks, distributed in either the NIC subsystem ④ or the ARM compute complex ①. NVIDIA provides the DOCA SDK to facilitate software development, offering platform-specific APIs alongside modified implementations of open-source toolkits such as the Data Plane Development Kit (DPDK) and Open vSwitch (OVS).

3 Systematically Characterizing Internal Contention

With BlueField-3's diverse components described in §2, exhaustively testing every combination for resource contention is impractical. Therefore, we ground our study in the common deployment pattern followed by SmartNIC-accelerated applications.

SmartNIC offloads span several components. Developers identify and offload parts of the applications that are amenable to acceleration, typically infrastructure functions such as storage replication, (meta)data aggregation, or traffic monitoring for security purposes, onto the SmartNIC as a *service*. The service's control logic typically runs on the SoC ARM cores, while the datapath operations run on the NIC's specialized accelerators and processors. We use the term *service traffic* to refer to the offloaded service traffic moving among these components, the host, and the network. Unless the application is offloaded in its entirety, the rest of it remains on the host, and may generate traffic, referred to as *application traffic*, that is entirely processed on the host and uses the NIC simply for communication over the network. A SmartNIC-accelerated application, therefore, may leverage *several components* across the NIC and have *several concurrent streams* of service and application traffic active at the same time, all sharing the NIC's internal fabric. To make matters worse, these streams may also have to contend with those of other applications at the same endpoint.

The gap between the expected and observed contention. To develop SmartNIC services for their applications, most developers do not have access to the detailed architecture of the NIC. They typically have to rely on the high-level architecture, i.e., high-level descriptions of acceleration

Acceleration scenario	Description	Representative service	Covers
ARM cores + lookaside accelerators (§4)	Traffic goes to ARM cores (off-path), which invoke lookaside accelerators for data transformations such as encryption and compression.	Secure distributed file system	①④
DPA on-path processing (§5)	Traffic goes to DPA, whose manycore processor computes on packets directly and on-path.	Hierarchical parameter server	⑤
NIC-subsystem packet pipelines (§6)	Traffic goes through packet processing pipelines, which apply steering, replication, and protocol offloads.	Network traffic monitor	③④

Table 1. SmartNIC acceleration scenarios. Each scenario represents a major class of SmartNIC services and exercises one or more accelerators or processors in the SmartNIC.

units and the paths connecting them, provided by the vendor (e.g., Figure 1). The programming APIs do not expose much more than that; they do not document what traffic streams a call triggers or how the hardware shares each path. Reasoning about contention, however, requires exactly that information: which paths a workload will use, and how the architecture mediates between traffic streams on the same path. As such, there is a non-trivial gap between a developer’s expectation and the post-deployment reality of what competing traffic streams will exist in the final system and how much performance degradation to expect for each one.

The goal of our measurement study is to expose this gap. Of the four recurring contention patterns we observe (Table 4), three do not match one’s expectation just from the high-level visible architecture and documentation: workloads that share no path in the visible architecture still degrade each other (§4 and §5), arbitration rules at contention points are multi-faceted (e.g., sensitive to the number of queues and threads a workload opens) and undocumented (§4, §5); a working set growing past an undocumented capacity boundary collapses not only its own workload but all competing traffic (§5); and throughput collapses that appear only when composed features interact through processing that per-feature interfaces do not expose (§6).

Three representative scenarios for contending traffic streams (Table 1). Guided by the above development and deployment norms, we structure our study around *three distinct NIC acceleration scenarios*, each pairing one major kind of acceleration unit with a representative service, so that each unit is exercised under the service and application traffic that accompanies it in a real deployment. Together, the three scenarios cover all major processing subsystems shown in Figure 1. Table 1 describes each scenario and its representative application.

Characterization workflow. Within each scenario, we characterize contention in two phases: component- and path-level microbenchmarks (steps 1–3 below), followed by end-to-end experiments using a representative application (step 4).

- (1) **Standalone baselines.** We measure the standalone throughput of the acceleration paths in the offloaded service. Every performance loss percentage in the paper is computed against these standalone baselines, measured on the same configuration.
- (2) **Microbenchmarks with contention-inducing cross traffic.** Traffic streams from the accelerated service can co-exist with the application’s remaining service traffic, the rest of the application traffic from the host, and traffic from other applications (accelerated or not) at the same endpoint. Whichever application they belong to, these streams are established between the three primary endpoints: the host, the external network, and the ARM cores. We therefore emulate them with six unidirectional flows between each pair of these endpoints: Host-to-Network, Host-to-ARM, Network-to-Host, Network-to-ARM, ARM-to-Host, and ARM-to-Network, each saturating a different internal data path (generation details in Table 2(left)). We measure the degradation for both the service traffic on the target acceleration path and the cross traffic, each

Flow vector	Generation method	Host System	NICs
Host ↔ Net	DPDK, 1500 B MTU, 2 queues (1/port)	CPU: AMD EPYC 9254, 24-Core, 2.9GHz	BlueField-3 B3220
ARM ↔ Net	DPDK, 1500 B MTU, 2 queues (1/port)	Memory: 12×32GB DDR5-4800	ARM OS: Ubuntu 22.04 LTS
Host ↔ ARM	DOCA Communication Channel	PCIe 5.0×16 (512Gbps)	DOCA: 3.1.0
		OS: Ubuntu 24.04 LTS	Firmware: 32.45.1020
		DPDK: MLNX_DPDK 22.11	Peer NIC: ConnectX-7
			Cable: 2×QSFP56 200 Gbps
			Firmware: 28.46.1006

Table 2. Contention-inducing flow setups (left) and Testbed hardware and software configuration (right) used for contention measurements.

against its own standalone baseline.

- (3) **Parameter sweeps.** When these microbenchmarks expose contention, we extend them with sweeps of the parameters that are controlled by the host application and the ARM-side service control logic which can plausibly govern the contention: queue count, message size, buffer size, and thread count. Because these parameters are under developers' control, the sweeps also identify the available mitigations.
- (4) **End-to-end application experiments.** Finally, we evaluate the scenario's representative application end-to-end, assessing whether the contention observed in the microbenchmarks persists in a realistic setting.

Testbed. Our testbed is illustrated in Figure 1, with full hardware and software details in Table 2(right). To saturate the 400 Gbps aggregate networking capacity of the dual-port BlueField-3, we connect it directly to a ConnectX-7 peer NIC using two 200 Gbps Ethernet cables. The SmartNIC and the peer NIC are connected to the host with matching PCIe to saturate their communication capacities. Workloads for host and ARM cores use NVIDIA's DOCA SDK, along with accompanying open-source libraries and tools, to access the SmartNIC's capabilities. Network-facing flows (Host↔Net, ARM↔Net) are generated with DPDK at 1500B MTU and two queues each (one per port) by default. Host↔ARM traffic uses the DOCA Communication Channel.

Thermal conditions. NIC overheating can cause thermal throttling, reducing BlueField-3 performance and confounding contention measurements. We monitored BlueField-3's ASIC temperature during the experiments using its built-in thermal sensors. The reported temperature remained below 71°C, well below the thermal throttling threshold of 91°C¹. Individual experiments were short, with a maximum observed temperature increase of approximately 6°C. These observations suggest that thermal throttling was unlikely to have affected performance during our experiments.

4 Contention Involving Lookaside Accelerators

BlueField-3 integrates lookaside accelerators for data transformations common in network services, such as encryption, compression, and hashing. The host or the ARM cores invoke them in an RPC-like manner: the processor submits requests to the accelerator, retrieves the results when ready, and may use queueing and request batching to pipeline these operations. An offloaded service that invokes an accelerator therefore adds request and response streams to the NIC's internal fabric, where they coexist with the service and application traffic (§3). Following the workflow from §3, we characterize the contention between these request and response traffic streams and other traffic. We find that functionally similar accelerators with similar APIs respond very differently to the same cross traffic depending on where they sit in the NIC's topology, and that the severity of the contention can depend on parameters such as the number of transmit queues the cross traffic uses.

¹Obtained from BlueField-3 using the command in the troubleshooting guide [25].

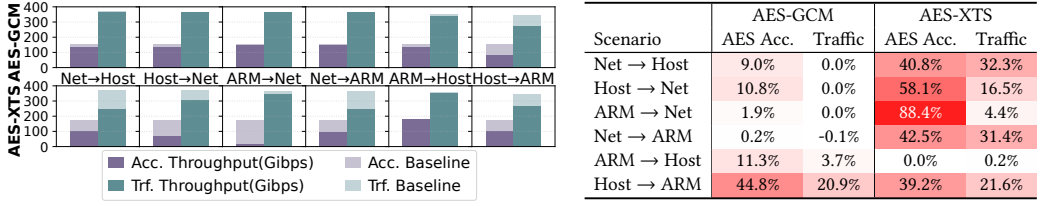


Fig. 2. AES accelerators (Acc.) along traffic flows (Trf.) throughput (left) and contention-induced loss (right).

We confirm these findings using a secure distributed file system as an end-to-end case study.

Selecting Target Accelerators. We focus on the AES-XTS and AES-GCM accelerators on BlueField-3 for our primary case study. Our preliminary screening of the lookaside accelerators (Appendix A) shows that, despite similar functionality and a common software API, these two accelerators reach their standalone throughput limits simultaneously when run concurrently, without measurable interference. This suggests they are backed by distinct hardware implementations, likely exercise different parts of the NIC datapath, and contend for different resources. This pair therefore lets us compare how two functionally similar accelerators, placed differently within the NIC, interact with the same cross traffic.

4.1 Topological Roots of Contention

When ARM cores invoke a cryptographic accelerator, the request and response streams may traverse the internal fabric alongside other traffic. To explore the potential resulting contention, we measure the throughput of AES-GCM and AES-XTS, first standalone and then alongside each of the six cross-traffic flows from §3, and report each side’s throughput loss relative to its standalone baseline.

Figure 2 shows the results for 32 KB decryption operations, with absolute throughput in (left) and loss percentages relative to the standalone baselines in (right). Both accelerators lose throughput under cross traffic, but they differ sharply in which flows hurt them: AES-XTS loses $\geq 40\%$ of its throughput alongside every flow with the network as one endpoint, dropping from over 175 Gbps to below 20 Gbps in the worst case, whereas AES-GCM loses at most 11% under the same flows. We hypothesize that this disparity stems from differences in accelerator placement within the SmartNIC internal topology. The AES-GCM accelerator resides within the ARM complex (① in Figure 1), whereas AES-XTS is integrated into the NIC subsystem (④), consistent with its availability in non-BlueField models [26]. Consequently, using AES-GCM, the ARM cores access the accelerator via the uncongested path ⑥, while for AES-XTS, the path is indirect ③ and shares internal links with network traffic, making it susceptible to interference. Neither the placement nor the shared links are visible in the high-level architecture developers see: at the API level, an AES-XTS call and, say, the application’s Host-to-Network traffic appear entirely unrelated.

The way contention changes with traffic direction supports this explanation. Alongside Host-to-Net traffic, AES-XTS loses 58% throughput while the cross traffic loses $<17\%$, suggesting that the arbitration point where the two streams converge, likely the internal PCIe switch, favors the Host-to-Net traffic. Reversing the traffic direction (Net-to-Host) makes the degradation more balanced, 40% for AES-XTS and 32% for the traffic. This indicates that reversing the traffic moves the contention to a different arbitration point, likely within the NIC subsystem. The two arbitration points seem to mediate between traffic streams differently: one favors the cross traffic, while the other spreads the loss more evenly. As such, the extent to which a workload degrades depends on where its streams converge with others, information not readily available to most developers.

The degradation in the presence of Host-to-ARM traffic, in contrast, is similar for both accelerators (44.8% for AES-GCM and 39.2% for AES-XTS). Both accelerators are invoked from the ARM cores,

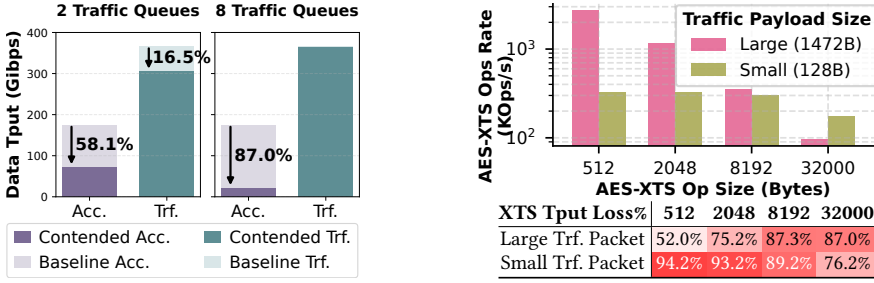


Fig. 3. AES-XTS throughput (tput) and corresponding loss % with varying (left) Host-to-Network traffic queues, and (right) message sizes.

so this flow plausibly contends with the invocation path the two accelerators share rather than with either one's placement-specific datapath. We also vary the competing traffic's rate and observe AES-XTS's throughput restored smoothly towards its standalone level (Appendix C).

Main takeaways. Contention arises between seemingly unrelated functional units. For example, an accelerator invocation and network traffic that appear independent at the API level can degrade each other's throughput by 40% or more. The inferred root cause is oversubscription of shared "links" in the NIC's internal fabric. An accelerator's placement determines which links its requests traverse, and each contention point, where streams converge, arbitrates the oversubscribed link in its own way. With visibility into the detailed topology of the NIC, this behavior can be largely predictable, but this is precisely what current programming interfaces and documentation do not expose. Until they do, developers can navigate these kinds of contentions by empirically narrowing down the placement of accelerators within the high-level NIC architecture and selecting, among functionally similar accelerators, the one whose path avoids the dominant cross traffic.

4.2 Dynamics of Internal Resource Arbitration

Having established how accelerator invocations contend with cross traffic depending on their location and where their traffic streams converge with others in the fabric, following step 3 of our workflow (§3), we sweep the workload parameters that plausibly govern the severity of that contention: the number of transmission queues and the message size. Both are routinely set by the host application and the service control logic in the ARM cores, so they determine how much control developers can have over the contention.

Impact of queue count. When AES-XTS contends with Host-to-Network traffic, increasing the number of host send queues from two to eight (four per port) drastically shifts the bandwidth split (Figure 3(left)): AES-XTS loses an additional two-thirds of its throughput, while the cross-traffic fully recovers from its prior degradation. Increasing the queue count does not change the standalone throughput of Host-to-Network traffic. Under contention, however, the additional queues appear to give this traffic a larger share of internal bandwidth, starving the accelerator traffic. We hypothesize that this shift occurs because internal arbitration mechanisms take the number of active queues into account. This can be a hazard for developers, who commonly scale queue count on the host to parallelize transmission, unaware that it also shifts the NIC's internal arbitration behavior.

Sensitivity to message sizes. We vary the message size (packet size for the competing traffic and operation size for AES-XTS decryption) and measure the resulting performance. As shown in Figure 3(right), small-message Host-to-Network traffic combined with small AES-XTS operations drives the system into a message-rate-limited regime where AES-XTS message rate plateaus at approximately 300 Kops/s, unlike when competing with larger-message traffic. Accelerator operations with smaller message sizes are likewise disproportionately affected by small-message

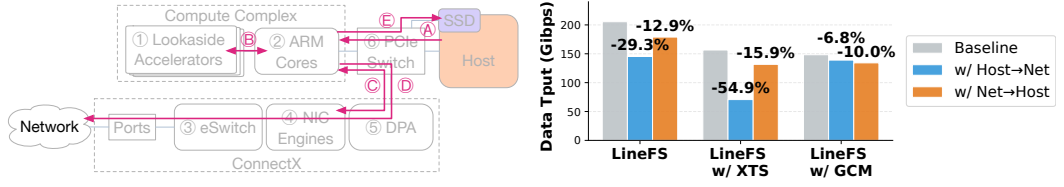


Fig. 4. Secure distributed file system offloaded service (extended LineFS). (left) The offloaded service's traffic streams; (right) throughput of LineFS variants under competing traffic.

cross-traffic, as the two compete for message rate rather than datapath bandwidth.

Generalizability to other accelerators. To check that these observations are not specific to AES decryption, we repeat the experiments with other operations and accelerators. AES encryption shows contention behavior identical to decryption, so the contention is not operation-specific. Other accelerators, such as the compression engine (Appendix A), have contention profiles that resemble AES-GCM rather than AES-XTS. While contention severity thus varies across accelerators, the general observation holds that the driver for contention is consistently the datapath topology and internal resource sharing rather than accelerator functionality.

Main takeaways. The severity of contention can depend on how a workload's transmission is configured. For example, transmitting the same amount of cross traffic over more queues on the host can cause an accelerator's performance to degrade significantly, and depending on the message size, the contention may shift from a bandwidth-bound regime to a message-rate-bound one. This behavior stems from the NIC's internal arbitration policies. Because the arbitration rules are undocumented, a change to these parameters on the host, e.g., increasing the number of send queues with the goal of parallelizing transmission, can unintentionally increase contention for offloaded NIC services. In the absence of well-defined arbitration rules or knobs to directly modify them, developers can and should use these parameters to tune the performance of their NIC offloads.

4.3 Case study: Secure Distributed File System

We evaluate whether the contention observed in the microbenchmarks persists in a realistic setting (step 4 of our workflow), using LineFS [15], a distributed file system optimized for SmartNICs. The SmartNIC's primary responsibility in the LineFS root node is replicating LineFS logs, or filesystem journals, to both local storage and remote nodes. We extend LineFS to support filesystem-level encryption, following existing systems such as fscrypt [34], using the SmartNIC's built-in AES accelerators. We also incorporate data chunk size and DMA optimizations from [38]. The final offloaded service is a logical pipeline whose stages, described below, generate the service traffic shown in Figure 4(left). They use different hardware resources in parallel, triggering competing encryption- and replication-related traffic streams:

- (A) Poll a chunk of journal from the host application(s) using Direct Memory Access (DMA).
- (B)/C) Verify and process the journal (optional data chunk encryption using crypto accelerator).
- (D) Send the journal along a replication chain via Remote Direct Memory Access (RDMA).
- (E) Persist the journal to the host storage using DMA.

Standalone performance. We compare three LineFS variants: a baseline with no encryption, LineFS with AES-XTS, and LineFS with AES-GCM (authentication tags omitted) for comparison. Under large-file workloads that saturate the datapath, the baseline achieves over 200 Gibps (Figure 4(right)), consistent with existing bottleneck analyses [38] and the improved capabilities of newer SmartNIC models. When LineFS runs in isolation, we do not observe the significant throughput degradation seen in the encryption microbenchmarks for encrypted LineFS variants, because

Operation	Used Memory	Net → Host		Host → Net		ARM → Net		Net → ARM		ARM → Host		Host → ARM	
		DPA	Trf.	DPA	Trf.	DPA	Trf.	DPA	Trf.	DPA	Trf.	DPA	Trf.
Read	DPA	54.2%	10.7%	50.0%	1.4%	62.7%	16.5%	55.3%	12.7%	34.5%	31.5%	45.2%	6.0%
	ARM	49.7%	0.0%	60.2%	0.0%	89.9%	5.0%	39.0%	7.6%	3.7%	0.0%	42.8%	-1.3%
Write	DPA	62.0%	0.0%	0.0%	0.0%	0.0%	0.1%	73.5%	7.3%	0.0%	0.0%	0.1%	6.5%
	ARM	2.3%	36.1%	0.8%	0.0%	63.2%	1.0%	22.6%	62.8%	2.5%	0.1%	55.1%	0.9%
Send	DPA	0.0%	61.7%	0.3%	57.9%	0.8%	58.5%	1.9%	59.2%	0.0%	68.8%	1.0%	68.8%
	ARM	1.6%	0.0%	18.4%	73.4%	44.9%	56.3%	2.6%	33.4%	3.0%	6.2%	52.0%	34.3%
Receive	DPA	N/A		0.0%	0.0%	11.3%	0.3%	N/A		-0.1%	14.6%	0.0%	19.2%
	ARM	N/A		0.1%	0.0%	24.0%	18.9%	N/A		2.5%	2.4%	61.2%	30.9%

Table 3. Lost-throughput (%) for competing 254-thread DPA operations (DPA) and the six traffic flows (Trf.). Used Memory (DPA/ARM) is the memory accessed for Read/Write or packet-buffer location for Send/Receive.

the accelerator becomes the pipeline bottleneck (B/C) and throttles downstream stages (D and E), limiting pressure on shared internal resources.

Performance sensitivity to host application traffic. Although the encrypted variants perform similarly standalone, their performance diverges once they share the NIC with other traffic from the same host (§3). When we introduce concurrent Host-to-Network traffic alongside LineFS replication (Figure 4(right)), the AES-XTS variant loses over 50% of its throughput, while the AES-GCM variant remains largely unaffected. This is the placement effect we observed in our microbenchmarks: AES-XTS shares internal resources with the cross traffic, and AES-GCM does not. Notably, AES-XTS is the primitive intended for this task because it is designed for data-at-rest encryption [24], is suited to partial journal updates, and aligns with systems like fscrypt. However, it suffers more severe performance degradation than AES-GCM, which typically targets data in transit, such as TLS [32]. Thus, standalone measurements do not capture how a NIC-accelerated application will behave after deployment. When the host exchanges other network traffic, the topological effects observed earlier resurface, and the service can suffer unexpected contention.

5 Contention involving the DPA

BlueField-3 introduces the Data Path Accelerator (DPA, ⑤ in Figure 1), a specialized manycore processor in the NIC subsystem optimized for on-path acceleration of packet and I/O processing [29], in contrast to the off-path ARM cores and lookaside accelerators (§4). Prior work has characterized DPA's standalone performance [7]. Following the workflow from §3, we instead characterize the contention between traffic streams that DPA operations generate and other traffic sharing the NIC.

We find that DPA memory and network I/O operations lose throughput along traffic they seemingly share no path with in the high-level architecture, and that given its many-core design, it receives a disproportionate share at contention points. Moreover, once its memory's working set outgrows its cache, throughput drops for all traffic, not just the DPA. Using a hierarchical parameter server for distributed training as the end-to-end case study, we show that these effects can cap the aggregate throughput gain from the DPA at a quarter of its standalone capacity.

5.1 DPA Operations Competing with Cross Traffic

DPA relies on resources beyond its own processor for two of its fundamental operations: memory access and network I/O. Because these operations traverse the same internal fabric as other NIC functions, they may use shared hardware resources, leading to contention. To characterize this contention, we follow steps 1–2 of our workflow (§3): we establish standalone baselines for both operations, then measure their performance under interference from the six cross-traffic flows.

DPA memory operation contention. DPA processors can access DPA-dedicated memory, as well as ARM memory by mapping the ARM memory region into the DPA's address space. Therefore, offloaded services can place data in either DPA or ARM memory. To form baselines, we measure the

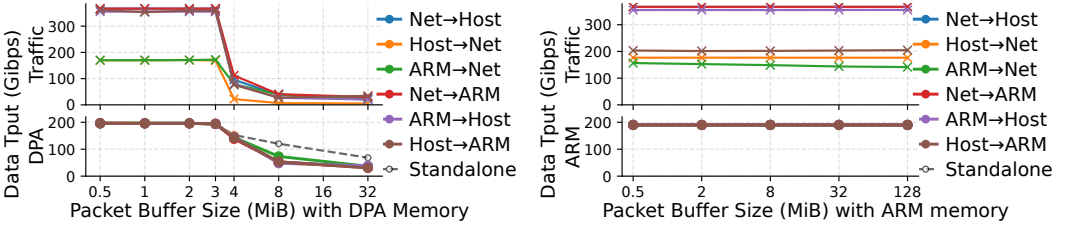


Fig. 5. DPA send throughput with competing traffic while varying buffer size for both packet buffer locations.

standalone read and write throughput of DPA threads accessing both memory regions via repeated memory operations in an ARM-to-DPA RPC call with 254 DPA threads². The results (Appendix B) confirm prior observations [7] that using DPA memory exhibits higher write throughput compared to read, whereas using ARM memory exhibits similar read and write throughput.

We then evaluate how concurrent traffic impacts DPA memory access by repeating the measurements while injecting competing traffic from the six contention-inducing flows described in §3. The Read and Write rows of Table 3 summarize the throughput drops. We observe that contention disproportionately penalizes DPA memory operations compared to the competing traffic. In extreme cases, DPA memory throughput collapses to less than one-ninth of its standalone capacity. Moreover, DPA memory access degrades under concurrent traffic originating from nearly all directions, including flows along paths that do not overlap with the expected path for the DPA data movement: for example, read operations targeting either memory pool contend heavily with all traffic, except for reading from ARM memory under competing ARM-to-Host traffic. Conversely, pacing the competing traffic’s rate down restores DPA read throughput toward its standalone level (Appendix C). The most consistent explanation is that servicing the DPA memory operation puts additional streams, for data or control, onto the NIC’s internal fabric beyond what the operation’s semantics alone would suggest, so the operation’s true “footprint” spans many internal paths and contends with traffic across them.

DPA network I/O contention. For DPA network I/O, packet buffers can be provisioned in either DPA or ARM memory. We first measure the packet sending and receiving throughput with all 254 DPA threads under both choices without competing traffic. Receive throughput is measured with a 256 KiB memory buffer per thread, consistent with NVIDIA sample code. Send throughput uses a 1.5 MiB total memory buffer, based on the results in the §5.2. Our results (Appendix B) show that using DPA memory for packet buffers yields marginally better per-thread throughput before aggregate throughput saturation, while using ARM memory yields higher aggregate throughput.

We then repeat the experiment with the same competing traffic. The Send and Receive rows of Table 3 show the results. Similar to memory access, DPA I/O operations can cause contention-induced throughput loss when competing with any of the six traffic flows, whether packet buffers reside in DPA or ARM memory. For example, DPA send operations interfere with not only other outbound traffic (e.g., Host-to-Network or ARM-to-Network), but also with the inbound traffic (e.g., Network-to-ARM or Network-to-Host). Unlike memory access, DPA I/O throughput generally degrades less than the competing traffic. This is consistent with the observations for DPA memory operations, and suggests that DPA network I/O introduces extra data and/or control traffic streams beyond those expected from the operations’ packet movement semantics. As such, it can impact and be impacted by seemingly unrelated traffic.

For DPA receive operations, we overall observe higher throughput loss when using ARM vs.

²Of the DPA’s 256 hardware threads, two are reserved by the firmware and are unavailable to applications [28]. Some BlueField-3 configurations may have only 190 DPA threads available.

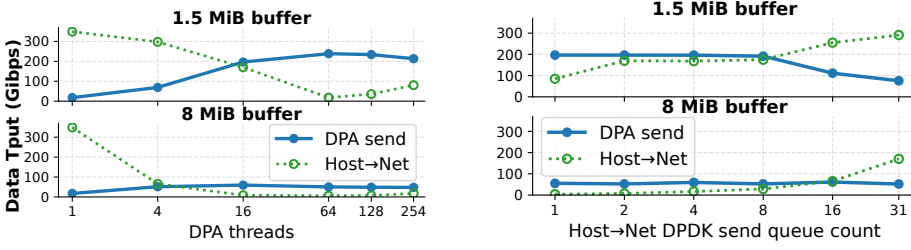


Fig. 6. DPA send throughput with competing Host-to-Net traffic while varying workload queue counts. (left) Varying DPA send threads; (right) varying send queue counts for Host-to-Net traffic competing with 16 DPA send threads.

DPA memory for packet buffers. However, this is due to the higher baseline receive throughput using ARM memory (187 Gibps) vs. DPA memory (48 Gibps). When we cap the receive throughput to 50 Gibps, we see similar contention patterns between the two buffer locations (results omitted for brevity). We exclude the Net-to-Host and Net-to-ARM results because, for these inbound flows, the receiving network port saturates before the traffic reaches the DPA, so contention at the port masks any contention effects involving the DPA.

Main takeaways. DPA memory access and network I/O both suffer from and inflict contention with cross traffic from nearly every direction, including flows that share no path with the DPA in the high-level architecture. In the worst case, DPA memory throughput falls below a ninth of its standalone capacity. The most consistent explanation is that the DPA places data and control streams onto the internal fabric on paths beyond what the semantics of its operations suggest. These hidden streams expand its footprint across the NIC and couple it to otherwise unrelated traffic. The developers, however, have some control through buffer placement on DPA vs ARM. For network I/O operations, ARM buffers reach higher DPA throughput, while DPA buffers keep the DPA's own operations largely protected from competing traffic.

5.2 A Closer Look at Network I/O Contention

To better understand the factors and parameters impacting our observed network I/O contention, we follow step 3 of our workflow and sweep the parameters a developer can control: the DPA packet buffer size, and the DPA thread and competing-traffic queue counts.

Throughput inflection with increasing buffer size. We vary the packet buffer size and repeat both standalone and contention experiments for DPA packet send operations. With DPA memory buffers, Figure 5(left) shows the DPA send and competing traffic throughputs measured in separate experiments, each either running DPA send alone or pairing it with one of the six contention-inducing flows. We observe a distinct performance inflection near buffer size 3 MiB, which is the DPA's L3 cache capacity. This inflection is absent with ARM memory buffers near either ARM or DPA L3 capacities, where throughput remains stable at 190 Gibps regardless of buffer size (Figure 5(right)). This inflection affects not only DPA operations but also all competing traffic, whose throughput collapses at the same buffer size. The global nature of this degradation, its alignment with the L3 cache size, and its appearance only with DPA memory buffers suggest that exceeding the DPA cache capacity changes how DPA I/O is serviced, triggering a regime shift that leads to widespread throughput degradation across both DPA and competing traffic.

Arbitration driven by thread and queue count. We independently scale DPA thread counts and Host-to-Network send queues, evaluating with both 1.5 MiB and 8 MiB DPA buffer sizes to represent both sides of the above inflection point. Figure 6(left) shows that increasing the number of DPA threads disproportionately reduces competing traffic throughput for both buffer sizes. With

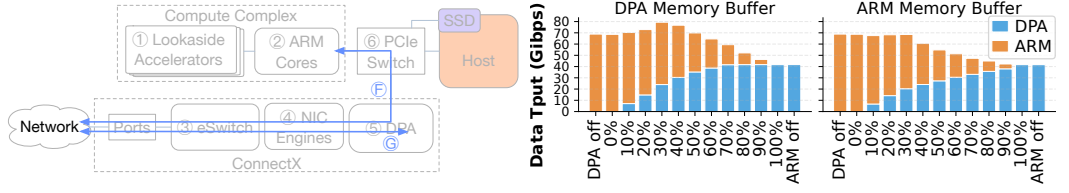


Fig. 7. Hierarchical parameter server offloaded service. (left) The offloaded service’s traffic streams; (right) DPA and ARM throughput under different DPA load fractions and packet buffer locations.

the 8 MiB buffer, competing throughput drops from 350 Gibps to <100 Gibps, while DPA throughput itself does not exceed 100 Gibps. This indicates that competing traffic is being starved even when the DPA does not fully utilize the datapath. With competing traffic running against 16 DPA send threads, increasing its send queue count can partially restore its throughput (Figure 6(right)) without reducing DPA throughput until saturation. Nevertheless, the system only reaches the 400 Gbps physical limit for the 1.5 MiB buffer.

We observe two patterns across both experiments and both buffer sizes: certain thread- and queue-count combinations leave the datapath underused, and changing one workload’s thread or queue count does not produce a proportional change in the other’s throughput. This suggests that the arbitration logic mediating between the traffic streams that share this datapath is sensitive to the number of active queues. Since the DPA’s manycore design spreads its work across many queues, its share of the bandwidth is out of proportion to its aggregate bandwidth, and competing traffic can reclaim part of its share only by spreading itself across more queues of its own.

Main takeaways. Our prior observation from §4.2 generalizes to the DPA contentions as well: the extent of contention can depend on the configuration of a workload’s transmission. For example, the extent of contention can disproportionately depend on the DPA’s memory working set or thread count, even if its aggregate throughput is below the peak, and impact all traffic not just the DPA. In the absence of well-defined knobs to directly control these effects, developers can and should use these parameters as control knobs to tune the performance of their NIC offloads, e.g., by keeping the DPA’s memory-related working set below the L3 boundary and picking queue counts in a way that achieves the arbitration outcome they desire.

5.3 Case study: Hierarchical Parameter Server

We evaluate whether the observed contention persists in a realistic setting (step 4 of our workflow), using a hierarchical parameter server for distributed training on BlueField-3. Parameter servers [18] are widely used for gradient aggregation: workers transmit sub-gradients keyed by model parameters, which are accumulated into a global value and broadcast back once aggregation completes. Following prior systems that offload aggregation to spare CPU resources (e.g., BytePS [12]), we instead offload it to the SmartNIC, using the DPA and ARM cores as two separate processing tiers.

Two-tier aggregation. We target sparsely activated models such as Mixture-of-Experts (MoE) [33], where parameters exhibit skewed access patterns under expert parallelism [8, 17]. We exploit this skew by partitioning aggregation across two tiers: the DPA handles the frequently accessed parameters that require high aggregation throughput (Ⓒ in Figure 7(left)), while the ARM cores process the remaining sub-gradients (Ⓔ). Each tier runs the same all-reduce collective: it accumulates incoming sub-gradients into a global value tensor through a localized lookup in the device’s local memory, and broadcasts the updated gradient back once a key’s aggregation completes. Packet steering rules direct incoming sub-gradient packets to the appropriate tier.

Aggregate throughput behavior. To evaluate how the previously observed contention behaviors impact this end-to-end NIC-accelerated application, we vary the fraction of aggregation

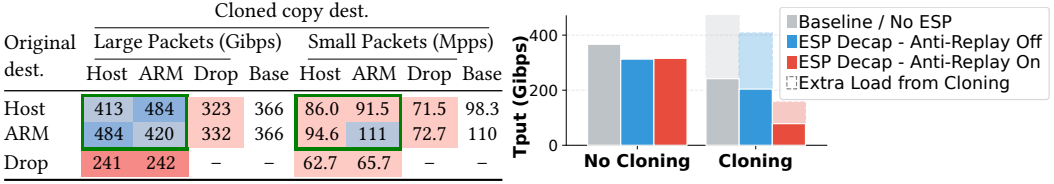


Fig. 8. Aggregate throughput with varying destination and protocol processing. (left) With varying original and cloned packet destinations (blue/red: above/below baseline); (right) the IPsec ESP decapsulation impact. handled by the DPA and the ARM cores. This controls how much traffic is routed through the DPA versus competing traffic through the NIC subsystem toward the ARM cores, exercising the memory and I/O contention points identified in the earlier experiments. We measure the aggregate service throughput when one of the processors is saturated.

In isolation, the DPA achieves >40 Gbps, while the ARM cores sustain just under 70 Gbps. However, when both operate concurrently, aggregate throughput is neither monotonic nor proportional as we increase the fraction of traffic handled by the DPA (Figure 7(right)). Aggregate throughput peaks when ~30% of the workload goes to the DPA, where the ARM cores and the DPA reach 55 Gbps and 24 Gbps respectively. Beyond this point, increasing DPA load, even though it could achieve 40 Gbps in isolation, reduces overall throughput. The net benefit from adding the DPA is therefore only 10 Gbps, 25.2% of its standalone capacity. Moreover, using ARM memory for DPA packet buffers provides negligible benefit while further degrading ARM-side processing capacity, mirroring the contention patterns observed in the memory and I/O experiments. This degradation may be compounded by contention for ARM memory, which serves both DPA packet buffers and ARM-side processing.

These results reaffirm, in an end-to-end NIC-accelerated application, the impact of the “hidden contentions” observed in the microbenchmarks. While the DPA offers additional on-path processing capacity, the contention between its operations and other traffic within the NIC prevents this capacity from translating into proportional throughput gains. As such, using the DPA effectively requires careful workload tuning in a way that takes into account both the DPA’s standalone capacity and its contention patterns so that its gains are not erased by contention within the NIC.

6 Contention in the NIC Subsystem

In addition to lookaside accelerators and DPA, the BlueField-3 NIC subsystem (③④ in Figure 1) has multiple inline packet processing pipelines (e.g., steering, encapsulation, and IPsec offloads) that operate on the incoming and outgoing packet streams. While these pipelines are designed to sustain line-rate processing, the performance implications of their interactions are not well understood. In this section, we use packet cloning as a controlled mechanism to stress the NIC subsystem’s internal data paths. Cloning amplifies the internal traffic volume beyond the external link rate, allowing us to expose spare internal processing headroom as well as unexpected interference between different packet processing pipelines. Here the contention originates within the service itself, specifically the internal streams it creates by cloning packets and by combining that cloning with other NIC features. This broadens our study beyond the six cross-traffic flows of §3 that drive the earlier scenarios. Using a traffic monitoring service as a case study, we show that combining cloning with other NIC features leads to significant and non-intuitive performance variations.

6.1 Characterizing Internal Headroom via Cloning

SmartNICs are typically evaluated at line rate (e.g., 400Gbps for BlueField-3), which masks how much internal data path capacity they actually have. However, many NIC features move more data inside the NIC than crosses the external link: monitoring replicates packets, and decompression emits

more bytes than it reads. Using such features at line rate in a NIC-accelerated application therefore relies on internal bandwidth that exceeds the link rate. We use packet cloning to directly probe this internal headroom. Each incoming packet is duplicated into two logical flows, allowing us to increase the internal traffic volume beyond the external input rate. We evaluate four configurations by directing the original and cloned copies either to the same endpoint (Host or ARM) or to different endpoints (one to Host and one to ARM).

The green-outlined boxes in Figure 8(left) report aggregate received throughput for large packets (1472B) in Gbps and aggregate received packet rate for small packets (128B) in Mpps. With large packets, cloning increases aggregate throughput in all four configurations. When both copies are directed to distinct endpoints, throughput increases by 32%, close to the ratio between PCIe bandwidth (512Gbps) and network link rate (400Gbps). When both copies are directed to the same endpoint, throughput increases by a more modest 13%. This indicates that the NIC can sustain internal data movement beyond the externally visible link rate, yet each endpoint, or the internal path serving it, has a throughput limit. In contrast, under a packet-rate-limited regime with small packets, aggregate packet rate (in Mpps) does not increase in most configurations, with the largest drop occurring when both copies are directed to the Host. These observations are consistent with prior work [38], which reports that the NIC has some resources dedicated per endpoint and others shared across endpoints: cloning to distinct endpoints draws on the dedicated capacity, whereas sending both copies to one endpoint contends for the shared capacity.

Main takeaways. The NIC internal headroom depends on the stressed resource and the traffic endpoints. In the throughput-limited regime (large packets), the NIC provides internal bandwidth higher than the line rate, creating headroom that can be exploited if needed by accelerated applications and services. However, each endpoint, or the internal path serving it, has a throughput limit below this internal bandwidth. Developers can exploit this headroom for cloning-based features when large-packet throughput is the bottleneck and clones can be spread across endpoints, but should not expect it for small-packet workloads. In that regime, which is packet-rate limited, the NIC does not offer significant headroom and cloning can consume packet processing capacity.

6.2 Downstream Processing Impacts Cloning Cost

To understand how downstream operations affect cloning cost, we evaluate “clone-and-drop” [27] used in monitoring, and “clone-and-tunnel” to combine cloning with stateful protocol processing.

Clone-and-Drop. The “clone-and-drop” pattern [27] discards some copies of packets after replication. This pattern is essential for dynamic monitoring: packets are always cloned, and relevant or more important packets are selected for analysis by dropping the replicas of those not requiring inspection. This approach preserves packet ordering between the analyzed and non-analyzed packets within a flow. Figure 8(left) shows that the cost of cloning is asymmetric depending on which copy is dropped. Throughput here is the received throughput of the delivered copy at its endpoint, measured against the no-cloning baseline of 366 Gbps. When the cloned copy is discarded, throughput for large packets falls by only 12% (to 323 Gbps). Dropping the original copy instead causes a much larger 34% reduction (to 241 Gbps). The 241 Gbps from dropping the original is almost exactly half of the 484 Gbps the NIC delivers when both copies go to distinct endpoints (§6.1). This is consistent with the NIC performing the full replication and splitting that two-copy capacity between the streams, then delivering only one. Half the capacity is spent on a copy that is ultimately discarded, leaving throughput below even the no-cloning baseline. Dropping the cloned copy, by contrast, keeps throughput within 12% of baseline, consistent with the discarded clone not incurring the full internal cost that a delivered copy would. The packet-rate-limited regime exhibits a similar trend, though less pronounced.

Clone-and-tunnel. Combining cloning with stateful protocol processing can lead to significant

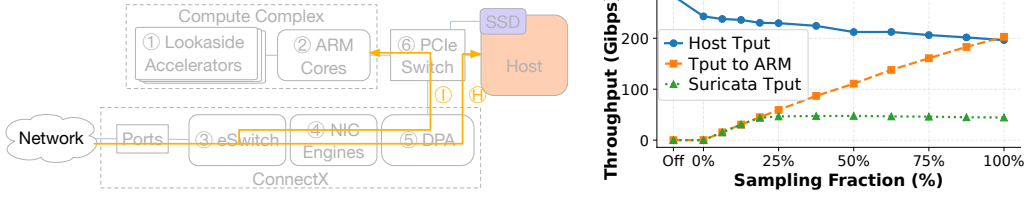


Fig. 9. IDS offloaded service. (left) The offloaded service’s traffic streams; (right) IDS throughput with varying sampling rate without ESP anti-replay.

and non-obvious performance degradation. To evaluate this interaction, we combine cloning with IPsec Encapsulating Security Payload (ESP) decapsulation. Unlike simple tunneling mechanisms that only modify packet headers, IPsec ESP requires payload decryption and anti-replay validation, introducing additional per-packet processing and state management within the NIC.

As shown in Figure 8(right), enabling IPsec decapsulation with anti-replay on the primary path to the host alongside cloning results in a significant throughput collapse. Turning on anti-replay reduces throughput by a further 61.5% relative to the same cloned, IPsec-decapsulated configuration with anti-replay disabled. Anti-replay validation updates a per-flow replay window in packet arrival order. The most consistent explanation with the observed collapse is that the cloned copies must therefore serialize on these ordered updates rather than being processed in parallel. Additionally, running IPsec ESP decapsulation concurrently with the AES-GCM and AES-XTS lookaside accelerators of §4 leaves both at their standalone throughput. Because IPsec ESP is normally secured with AES-GCM, one might expect it to share that accelerator. The absence of any interference instead indicates that ESP’s inline decryption does not contend for the lookaside accelerators, suggesting that the cryptographic accelerator in the IPsec ESP engine is likely separate from the lookaside accelerators.

Main takeaways. The cost of cloning is not intrinsic to replication itself, but to how the copies are processed. Dropping the original vs. the cloned copy introduces asymmetric overheads, and combining cloning with stateful processing (e.g., IPsec anti-replay) can trigger a severe throughput collapse, likely because the cloned copies should follow the in-order stateful anti-replay window packet processing as well rather than proceeding independently. This provides yet another piece of evidence of the gap mentioned in §3 where composed features interact through “state” that neither interface explicitly exposes. Thus, cloning is not an isolated primitive and not transparent to other NIC features. As such, developers composing it with stateful offloads such as IPsec should measure the combination directly rather than budgeting the two costs independently.

6.3 Case Study: Network Intrusion Detection

To evaluate how the contentions observed in our microbenchmarks manifest in a realistic setting, we build and evaluate a Network Intrusion Detection System (IDS). Our implementation augments Suricata [35], a widely deployed open-source IDS platform, to run on the ARM cores with packet steering managed via the DOCA flow library. Incoming traffic is sampled using “clone-and-drop” at the NIC subsystem with a configurable sampling rate. Replicated packets are forwarded to the ARM cores for Suricata inspection ① (v8.0.2 with a standard signature rule set), while primary traffic is delivered to the host ②, as shown in Figure 9(left). For the primary host path, we offload IPsec ESP decapsulation to the NIC subsystem to simulate a representative production deployment.

This design reflects a common offload pattern that uses cloning to minimize the impact of compute-intensive IDS processing on host application performance. It also directly combines the two previously studied patterns: asymmetric overheads from clone-and-drop and the interaction

Pattern	Explanation / Inferred Root Cause	Available mitigation
P1: Contention on visibly shared paths (§4, §6.1)	Demand exceeds the capacity (e.g., bandwidth or rate) of the visibly shared resource.	Follow established capacity-planning practice, e.g., spread load across endpoints or resources where the opportunity exists.
P2: Contention without a visibly shared path (§4, §5)	Similar to P1, but operation footprints (unit placement, generated data/control streams), and therefore, oversubscribed links, are not fully visible.	Empirically shape the footprint (§8) to find contended paths/end points; then mitigate accordingly.
P3: Multi-faceted arbitration at contention points (§4.2, §5.2)	Undocumented arbitration rules for resource split at contention points.	Empirically find and apply knobs through parameter sweeps (e.g., queue and thread count and message size) to reach desired split.
P4: Contention between composed features (§6)	Features interact through processing that per-feature interfaces do not expose	Measure combinations directly and redesign compositions that prove too costly.

Table 4. The four recurring contention patterns, their inferred root causes, and available mitigations. Only one pattern (P1) is predictable from the high-level architecture, documentation, and APIs. Others are not.

between cloning and stateful IPsec processing.

Contention rooted in feature interactions. Figure 9(right) shows the throughput observed on the Host (blue), the throughput of sampled packets sent to the ARM cores (orange), and the throughput of ARM cores running Suricata (green) for MTU-size packets. On the ARM cores, Suricata throughput with the full rule set reaches 47 Gbps using 15 worker threads and one control thread. This reaffirms the decision to only apply it on cloned packets to shield the host from its packet processing overhead. However, for traffic directed to the host and without IPsec anti-replay, the IDS monitoring introduces a 14% throughput loss at zero sampling rate (clone-and-drop for all packets), increasing to 30% as the sampling rate increases. Enabling IPsec anti-replay significantly amplifies the cost: throughput loss reaches 69.0% even without sampling and increases slightly to 70.2% with higher sampling rates.

These results show that the contentions observed in isolation manifest in an end-to-end setting. While cloning enables flexible traffic sampling and isolates IDS processing on the ARM cores, combining it with stateful protocol processing on the host path leads to substantial throughput degradation. This limits the viability of such designs for high-performance deployments despite the apparent isolation provided by cloning.

Evaluating an alternative stage placement. To evaluate whether software restructuring can mitigate this interaction, we move the decapsulation stage, including anti-replay processing, from the NIC subsystem’s inline engine to the ARM cores. Decryption remains in the NIC subsystem. In our evaluation, this revised placement delivers 180 Gbps to the host using four of the 16 ARM cores when Suricata receives no samples. This throughput is more than twice that of the original design with anti-replay on the host path and only 35% below the baseline without anti-replay. However, this placement sends additional traffic to the ARM cores and consumes CPU capacity otherwise available to Suricata. When Suricata runs on the remaining cores with 11 worker threads, contention between Suricata and the decapsulation stage within the ARM processor limits host throughput to 167 Gbps and Suricata inspection throughput to 21 Gbps.

7 Recurring Contention Patterns

Most of the contention we observed through careful measurement in §4-§6 could not have been anticipated from the high-level architecture and the API documentation: workloads with no visible resource in common degrade each other, and even when workloads do share a visible path, the degradation does not match the proportions one would expect. Across our three studies, four patterns recur. Only the first (P1) matches what the visible architecture predicts; the other three (P2–P4) stem from mechanisms that are not documented in the architecture diagrams or the

programming interfaces. The patterns and possible mitigations are summarized in Table 4.

P1: Contention on visibly shared paths. Traffic streams that share a path or endpoint in the high-level architecture split its capacity (e.g., bandwidth or rate) when that path is the bottleneck. In §4, Host-to-ARM cross traffic degrades throughput of invocations of AES-GCM (44.8%) and AES-XTS (39.2%) to a similar extent, consistent with contention on the ARM-side invocation path that both accelerators visibly share with this flow. In §6.1, two streams that target the same endpoint, the original and cloned copies of the same traffic, are together capped by that endpoint's delivery path at 413-420 Gbps. Spreading the same two streams across two endpoints lifts that cap. But, since the two streams still share the NIC's aggregate internal capacity, throughput rises only to 484 Gbps. In both cases, the streams split a resource the high-level architecture shows them sharing. P1 is the *only pattern developers can plan for from the visible architecture alone*.

P2: Contention without a visibly shared path. Workloads that appear unrelated at the API level degrade each other. In §4, AES-XTS invocations degrade severely under cross traffic involving the network, in the microbenchmarks and in the LineFS case study alike, while AES-GCM, a functionally similar accelerator with the same API, remains largely unaffected under the same traffic. In §5, DPA memory access and network I/O degrade under cross traffic from nearly every direction, including flows that share no path with the DPA in the visible architecture, falling below one-eighth of standalone throughput in the worst case. As another example, once the DPA's packet-buffer working set outgrows its 3 MiB L3 cache, throughput drops sharply not only for the DPA but for all six cross-traffic flows, traffic that never touches the DPA.

The most consistent explanation for such contention is oversubscription of internal-fabric links that the visible architecture does not show as shared. Each workload has a *footprint* on the fabric: the set of internal links its traffic occupies, shaped by where the units it uses sit in the topology and by the underlying traffic streams the hardware generates to serve its operations. Parts of this footprint, e.g., accelerator placement (§4), the extra streams DPA operations generate (§5), and how they shift once a working set outgrows the cache (§5.2), are visible only through measurement. In the absence of more information from vendors, the mitigation is to shape the footprint empirically, as we have done in our measurement study and recommend developers to do in §8.

P3: Multi-faceted arbitration at contention points. How a contention point splits its capacity among competing workloads depends not just on each workload's aggregate rate but on how that traffic is presented, e.g., its queue and thread counts and message sizes. In §4.2, spreading the same cross traffic over more send queues (two to eight) costs AES-XTS an additional two-thirds of its throughput even though the cross traffic's achieved throughput is unchanged. In §5.2, adding DPA threads drops competing traffic from 350 Gbps to below 100 Gbps even though the DPA's own throughput stays below 100 Gbps, and the competing traffic reclaims bandwidth only by spreading its traffic over more queues. These factors and their impact on arbitration are not well-documented so the split between competing traffic can shift under seemingly unrelated changes. That said, our findings also offer queue and thread counts and message sizes as knobs to developers to deliberately steer the split toward the outcome they want, and re-tune them when the workload changes.

P4: Contention between composed features. Features composed within a single service can degrade each other, and their combined cost is not the sum of their individual costs. In §6, even which copy of a clone-and-drop pair is discarded matters: dropping the original costs 34% of throughput versus 12% for dropping the clone. Composing cloning with stateful processing can have more severe implications. The cloning-based sampling in the IDS case study costs the host path 15–30% by itself, and IPsec ESP decapsulation with anti-replay does not have extra overhead as it is an inline offload designed for line rate. Their composition, however, costs a further 61.5% of throughput in the microbenchmarks and raises the IDS host-path loss to about 70%. Per-feature costs therefore can be misleading in isolation. A feature can impose its processing on traffic the

developer never associated with it, as anti-replay does on the cloned copies, so the cost of a feature combination must be measured directly rather than inferred from per-feature numbers.

8 Guidelines for Contention-Aware SmartNIC Service Development

Out of the above four recurring contention patterns, only P1 is predictable from the visible architecture. As such, decisions that developers make on functional grounds or for one application or service in isolation, including but not limited to which accelerator to call, where to place a buffer, how many queues to open, are also contention decisions, whether the developer intends them to be or not. Based on our experience in §4–§6, we recommend the following four steps for contention-aware development of SmartNIC services.

Step 1: Enumerate placement candidates for the service’s stages, and the traffic streams they add. Developers typically decompose a service into data-flow stages, i.e., blocks of data operations, that can be offloaded to different SmartNIC components. LineFS’s functionality, for example, is decomposed into journal polling, optional encryption, persistence to host storage, and replication to remote nodes (§4). For contention-aware development, the developer should enumerate, for each stage, the candidate placements, i.e., the units that it can be offloaded to, together with the traffic each candidate placement would add to the NIC’s internal fabric.

The latter (traffic part) is inevitably best-effort. As our measurements show, the traffic added by using a NIC processor or accelerator may go beyond its primary data movement, so developers should enumerate what they can anticipate based on the operation’s inputs, outputs, and observable memory accesses, and revise it as they perform measurements in Step 2. Our DPA network I/O experiments (§5) illustrate this: the operation requires a packet buffer in either ARM or DPA memory. In either case, accessing that buffer introduces supporting memory traffic beyond the DPA’s own processing. Developers should therefore account for these accesses and the memory they target when enumerating the operation’s traffic streams.

Step 2: Measure candidate placements under *all* cross traffic. Our study shows that relying on just the visible architecture or standalone benchmarks is not sufficient (P2 - P4). For the acceleration units (lookaside accelerators, DPA, NIC-subsystem pipelines) and NIC features our studies cover, Table 4 provides the expected contention patterns and mitigations directly. For others, developers can use a workflow similar to ours that helps systematically characterize contention (§3): establish standalone baselines, add the six cross-traffic flows augmented with the service’s own auxiliary streams, and sweep the developer-controlled parameters. It’s worth noting that these measurements should cover each acceleration unit the service uses, even if they share the same API. For example, as discussed in §4, AES-GCM and AES-XTS share an API, but each operation uses a different acceleration unit and has its own distinct and very different contention profile.

The results indicate which pattern is at play: degradation under flows with no visibly shared path points to hidden internal streams (P2); shares that shift with a workload’s queue or thread configuration, even when its aggregate rate is unchanged, point to arbitration effects (P3); a sharp drop when a buffer or working set crosses a size threshold points to a capacity-triggered footprint change (P2); and degradation that appears only when features are enabled together points to feature interaction (P4).

Step 3: Rank placements by service-specific splits. Contention can rarely be avoided outright. More realistically, one must understand when and to what extent it occurs and plan for it. Planning for contention means deciding how its cost should be split across the service’s flows and metrics. As our measurement study shows, this is difficult to directly estimate in the absence of more detailed NIC architecture. Instead, we suggest developers rank the measured placements by how closely each realizes that split. In LineFS (§4), the developer must decide what share of bandwidth the offloaded replication service keeps under load from the application it serves: too little and replication falls

behind, too much and the application the offload is meant to serve is starved.

In the IDS (§6), developers may balance the throughput to the host against the ARM inspection throughput. Our IDS experiment (§6) demonstrates a concrete software re-architecting mitigation: restructuring the service pipeline to move IPsec ESP decapsulation and anti-replay processing to the ARM cores more than doubles throughput delivered to the host. This placement avoids the clone-and-tunnel performance degradation in the NIC subsystem. However, this placement introduces contention within the ARM processor, reducing the capacity available for IDS inspection. More generally, selecting an alternative execution unit for a data-flow stage may mitigate contention by bypassing contended resources, while introducing contention elsewhere. Developers should therefore weigh the benefits of each placement against its impact on other aspects of their service.

Step 4: Tune the configuration of the chosen placement. The parameter sweeps of Step 2 already reveal how queue and thread counts, message sizes, and buffer sizes shift contention (P2, P3), which is taken into account in the ranking in Step 3. Once a placement is chosen, developers should tune these parameters for their specific offloaded NIC services and the cross traffic that exists in their deployment. While developers routinely set these parameters for reasons unrelated to contention, our study suggests setting them with contention in mind: keep the DPA working set below the L3 limit to avoid degrading both DPA and competing traffic (§5.2, P2), choose queue counts to obtain the desired bandwidth split between accelerator and competing traffic (§4.2, P3), and re-examine these choices whenever workloads change.

9 How NIC Vendors Can Help

Architectural transparency. Today, developers must reason about contention from the high-level architecture and programming APIs, neither of which documents which paths operations of different NIC components use or how the NIC mediates between multiple streams sharing the same path. While vendors may limit such visibility for practical or proprietary reasons, our studies have shown the cost of this opacity in practice, making it difficult for developers to reason about contention and fully utilize the device. This points to a broader research question: how much information about internal resource usage is enough for predictable performance, and how can vendors expose it without revealing proprietary architectural detail?

The patterns revealed by our measurement studies (§7 and Table 4) offer a starting point for what such visibility should cover: which internal paths an API call's traffic traverses and how that changes if certain capacity thresholds are exceeded (P1, P2), the granularity at which arbitration allocates shared resources (P3) and how features behave when composed with stateful processing (P4), together with hardware counters for the utilization of internal paths to help developers attribute an observed degradation to a specific contended resource, rather than inferring the resource from throughput changes alone. This does not necessarily require disclosing the detailed internal architecture. Vendors know their architecture well and are best positioned to abstract it into a small set of contention-relevant paths, capacities, and counters that developers can reason about without seeing the design behind them. What such an abstraction should expose, and how to keep it both useful for contention analysis and safe to publish, is an interesting avenue for future work, one that measurement studies like this one can ground.

Interoperability of software frameworks. Building the services and workloads for this study also surfaced “functional” gaps in the SmartNIC development lifecycle where further vendor support would likewise help. For example, BlueField-3 supports several programming and configuration frameworks beyond the native DOCA SDK with overlapping functionality. Support for packet steering alone is available through DOCA, OVS, DPDK `rte_flow`, and the `libmlx5` driver, each with unique APIs and features. While vendors extensively document each API in isolation, composing them can result in undocumented behavior that developers must discover experimentally, just

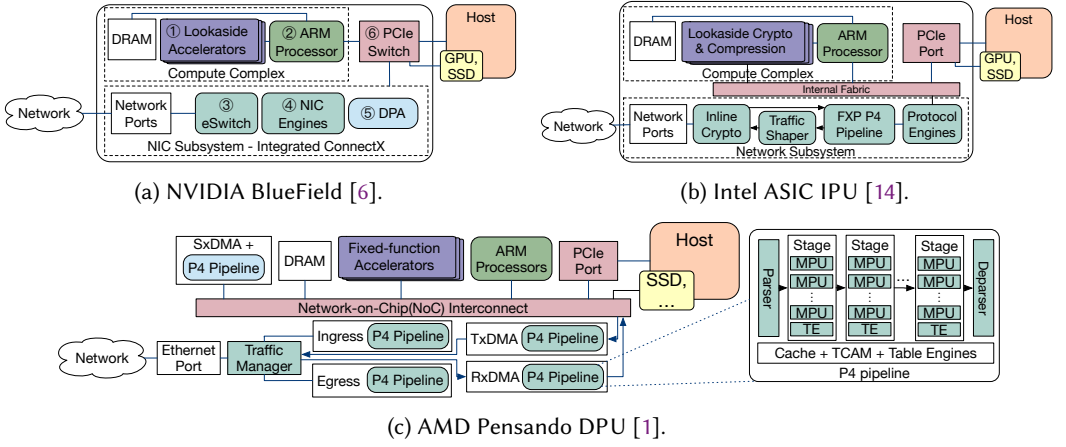


Fig. 10. High-level architectures of NVIDIA BlueField, Intel ASIC IPU, and AMD Pensando DPU, reflecting our understanding based on public information. Internal details are simplified and may be incomplete.

as composed hardware features interact in ways per-feature documentation does not cover (P4). Documentation that covers how these frameworks behave when used together would remove much of this trial and error.

10 Applicability to Other SmartNICs

While this study focuses on NVIDIA BlueField-3, architectural commonalities suggest that our approach and the recurring contention patterns we identified may also apply to other SoC-based SmartNICs.

Architectural commonalities of SmartNIC platforms. SoC-based SmartNIC platforms share a common high-level structure, as introduced in §2, despite differences in how they implement and interconnect their subsystems. Figure 10 compares the components and interconnections of BlueField, Intel IPU, and AMD Pensando DPU, based on our understanding of publicly available information. In the BlueField family (including BlueField-3, the earlier BlueField-2 [38], and upcoming BlueField-4 [30]; Figure 10a), the ConnectX NIC subsystem and ARM compute complex communicate over a dedicated internal PCIe switch. In contrast, Intel IPUs like Mount Evans [11] (Figure 10b) “tightly couple” their P4-capable NIC subsystem and protocol accelerators with the ARM complex via an undisclosed internal fabric, while AMD Pensando DPUs [3] (Figure 10c) connect their NIC subsystem based on Match Processor Unit (MPU) pipelines, ARM cores, memory, and other fixed-function accelerators over a unified Network-on-Chip (NoC) fabric. Despite these differences, each platform follows the same organization: a packet-processing NIC subsystem and a general-purpose compute complex are connected through an internal fabric and augmented by inline and lookaside accelerators. To highlight these commonalities, we color-code the components by functional group, using the labels in the BlueField-3 diagram (Figure 10a) as a reference: lookaside accelerators (①), general-purpose processors (②), packet processors (③, ④), novel programmable accelerators (⑤), and internal interconnect modules (⑥). Similarities in datapath organization allow us to place these functionally comparable components in similar relative positions across the three diagrams.

Transferable root causes vs. platform-specific triggers. Although each platform has its specific architectural implementations, their common heterogeneous architecture suggests that the root causes behind our four recurring contention patterns (§7 and Table 4) could transfer

across SmartNIC platforms, even though the specific cases that trigger them may be different. For example, the performance degradation of AES accelerators under cross traffic in §4 is inferred to be caused by the oversubscription of internal fabric links, a mechanism that could transfer to other platforms because they all rely on shared internal fabrics that workloads can contend for. In contrast, the specific accelerators showcased, AES accelerators, are particular to BlueField-3, where the placement of AES-XTS in the NIC subsystem can lead to more contention than AES-GCM under certain workloads. On other platforms, similar root causes could govern contention and may manifest across different components and through different platform-specific features, as we discuss below.

Contention over visible or invisible shared paths (P1, P2). Like BlueField-3, the diverse processing units of these platforms connect over an internal fabric, which introduces potential contention points along datapaths that are shared visibly or invisibly. As such, which processing units an offloaded service uses, and what operation footprints they create, could affect the contention it encounters and imposes on others. If such contention occurs, whether it is expected (P1) or unexpected (P2) would depend on whether the placement and footprint are visible to developers. This effect was demonstrated on BlueField-3 by the lookaside accelerator behavior observed in §4. On Intel IPU, support for custom storage protocols suggests a potential source of similar contention. For example, combining NVMe initiator protocol offloading in its network subsystem with AES-XTS and CRC lookaside accelerators [14] in its compute complex, as visualized in Figure 10b, requires data to traverse between the network and compute subsystems. AMD Pensando DPU similarly connects its host, ARM, accelerator, and network domains via a NoC fabric, as visualized in Figure 10c, which may create internal contention as data traverses multiple components for processing, although limited public NoC details prevent further analysis.

Arbitration effects (P3) and contention related to composed pipeline features (P4). Where traffic from heterogeneous components converges, hardware arbitration determines how bandwidth is shared, based on factors that are not clearly exposed to developers but may include offered load, message size, and queue configuration. One such factor is thread and queue concurrency, which can influence arbitration when flows originate from multi-core processing units, as observed on BlueField-3 with host DPDK send queues (§4.2) and DPA thread counts (§5.2). Orthogonally, composing complex features within multi-stage packet pipelines can introduce unexpected performance degradation (P4), as observed with BlueField-3's ASAP² pipeline in §6. Intel IPU's ARM cores provide a setting in which thread and queue concurrency could affect arbitration (P3). Its networking subsystem combines P4 pipelines with specialized protocol engines (Figure 10b), suggesting that chaining multiple offloaded protocol operations could also introduce composition-induced degradation (P4). AMD Pensando DPU's MPU NIC subsystem consists of pipelines of parallel MPUs (Figure 10c), suggesting potential mechanisms for both P3 and P4: arbitration among flows within pipeline stages could depend on the number of MPUs used, while MPU pipeline stages could interact adversely when complex features are composed [2].

Portability of development guidelines and workflow. The discussion above suggests that the root causes of our contention patterns can apply across platforms, so single-function benchmarks are unlikely to predict multi-feature service performance on them either. Moreover, beyond common contention mechanisms, these platforms also share the “visibility” gap mentioned in §3, where, as on BlueField-3, high-level architectures and APIs leave internal paths and accelerator placements undocumented for developers. Combining both factors, we expect our characterization workflow (§3), contention-aware development guidelines (§8), and recommendations for vendors (§9) to apply to these SmartNICs as well. Together, they provide a vendor-agnostic blueprint for developers uncovering and navigating internal contention in complex services, as well as a concrete starting point for researchers characterizing future hardware.

11 Related Work

As performance is a defining factor for SmartNIC platforms, several prior works have characterized SmartNIC capabilities. Furthermore, existing solutions for modeling and optimizing SmartNIC services are fundamentally predicated on the understanding of the device's performance envelope.

Performance characterization. While existing works characterized various aspects of SmartNIC performance, evaluations typically focus on a single hardware element, even with multiple resource usages. BenchBF3 [7] provides a detailed assessment of DPA. Other studies [13, 20, 41] confine evaluation to isolated subsystem measurements and service benchmarks built on conventional CPU workloads, effectively treating the SmartNIC as a packaged server and NIC. Michalowicz et al. [22, 23] analyze BlueField-3's capabilities for HPC communication offloading. Although Wei et al. [38] provide a guideline for concurrent communications, they only focus on communication efficiencies between the host, network, and ARM cores. Currently, no existing work comprehensively characterizes the performance envelope when resources are simultaneously stressed.

Performance modeling. Beyond characterization, several research efforts aim to model SmartNIC performance during workload execution. LogNIC [10] models the SmartNIC as a directed graph of fixed-throughput components and links. Yala [39] employs a data-driven profiling approach that models the architecture as a processor-centric system, where memory and accelerators are independent peripherals. Krude et al. [16] use symbolic execution to analyze processor performance, though their models prioritize code analysis over other integrated hardware elements. Consequently, these solutions cannot accurately capture interactions within services that utilize multiple system resources concurrently.

Performance optimization. Related efforts also focus on optimizing service performance on SmartNICs. Alkali [19] and Pipeleon [40] provide specialized optimization frameworks for P4 packet processing. Clara [31] builds a machine learning-based toolkit to facilitate the transition of CPU-based code to SmartNIC architectures. These solutions optimize within homogeneous architectural designs, hence do not address the cross-component contention that arises across a heterogeneous SmartNIC architecture.

12 Conclusion

SoC-based SmartNICs have evolved into heterogeneous architectures, integrating various accelerators with processors. This paper presents a systematic analysis of resource contention within the BlueField-3 SmartNIC, demonstrating that contention can induce significant and non-intuitive performance degradation. We distill our observations into four recurring contention patterns, only one of which is predictable from the device's visible architecture, and build on them to offer a four-step development guideline for SmartNIC service developers. The open question we leave for vendors is how to expose enough contention-relevant architectural details for developers to reason about contention without trial and error, yet without disclosing proprietary details.

Acknowledgments

The authors thank the shepherd and SIGMETRICS reviewers for their valuable feedback. This work was supported in part by a Canada Research Chair grant CRC-2023-00035 and CRC-2025-00014, and NSERC Discovery Grant RGPIN-2019-06587.

The authors acknowledge the use of generative artificial intelligence in producing code that supported the experiments and generated visualizations in the paper.

References

- [1] Advanced Micro Devices, Inc. 2024. Architecture Matters: Performance of a Multi-stage SDN Pipeline on ARM vs AMD Pensando Programmable Silicon. <https://www.amd.com/content/dam/amd/en/documents/pensando-technical->

- [docs/white-papers/pensando-comparison-of-dpu-hardware-strategies.pdf](#)
- [2] Advanced Micro Devices, Inc. 2024. Flow Offload Reference Pipeline User Guide (UG1670). <https://docs.amd.com/r/en-US/ug1670-amd-pensando-ssdk-flow-offload-user-guide/P4-Pipeline-Stages>
 - [3] Advanced Micro Devices, Inc. 2024. Product Brief: AMD Pensando Giglio Data Processing Unit. <https://www.amd.com/content/dam/amd/en/documents/pensando-technical-docs/product-briefs/pensando-giglio-product-brief.pdf>
 - [4] Amazon Web Services, Inc. . 2022. The components of the Nitro System. <https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/the-components-of-the-nitro-system.html>
 - [5] Amazon Web Services, Inc. 2024. The Security Design of the AWS Nitro System. <https://docs.aws.amazon.com/pdfs/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.pdf>
 - [6] Idan Burstein. 2021. NVIDIA Data Center Processing Unit (DPU) Architecture. In *2021 IEEE Hot Chips 33 Symposium (HCS)*. 1–20. doi:10.1109/HCS52781.2021.9567066
 - [7] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Yin Zhang, Ming Liu, and Zeke Wang. 2024. Demystifying Datapath Accelerator Enhanced Off-path SmartNIC. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*. IEEE, New York, NY, 1–12. doi:10.1109/ICNP61940.2024.10858560
 - [8] Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models. arXiv:2401.06066 [cs.CL] <https://arxiv.org/abs/2401.06066>
 - [9] Stewart Grant, Anil Yelam, Maxwell Bland, and Alex C. Snoeren. 2020. SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (Virtual Event, USA) (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 681–693. doi:10.1145/3387514.3405895
 - [10] Zerui Guo, Jiaxin Lin, Yuebin Bai, Daehyeok Kim, Michael Swift, Aditya Akella, and Ming Liu. 2023. LogNIC: A High-Level Performance Model for SmartNICs. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (Toronto, ON, Canada) (MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 916–929. doi:10.1145/3613424.3614291
 - [11] Intel Corporation. 2024. *Intel Infrastructure Processing Unit (Intel IPU) SoC E2100 Product Brief*. Technical Report. Intel Corporation. <https://www.intel.com/content/www/us/en/content-details/818147/intel-infrastructure-processing-unit-intel-ipu-soc-e2100-product-brief.html>
 - [12] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 463–479. <https://www.usenix.org/conference/osdi20/presentation/jiang>
 - [13] Arjun Kashyap, Yuke Li, Darren Ng, and Xiaoyi Lu. 2025. Understanding the Idiosyncrasies of Emerging BlueField DPUs. In *Proceedings of the 39th ACM International Conference on Supercomputing (ICS '25)*. Association for Computing Machinery, New York, NY, USA, 807–821. doi:10.1145/3721145.3725780
 - [14] Patrick Kennedy. 2025. Intel IPU E2200 400G DPU at Hot Chips 2025. <https://www.servethehome.com/intel-ipu-e2200-400g-dpu-at-hot-chips-2025/>
 - [15] Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. 2021. LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (Virtual Event, Germany) (SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 756–771. doi:10.1145/3477132.3483565
 - [16] Johannes Krude, Jan R  th, Daniel Schemmel, Felix Rath, Iohannes-Heorh Folbort, and Klaus Wehrle. 2021. Determination of throughput guarantees for processor-based SmartNICs. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies (Virtual Event, Germany) (CoNEXT '21)*. Association for Computing Machinery, New York, NY, USA, 267–281. doi:10.1145/3485983.3494842
 - [17] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=qrwe7XHTmYb>
 - [18] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 583–598. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu
 - [19] Jiaxin Lin, Zhiyuan Guo, Mihir Shah, Tao Ji, Yiyang Zhang, Daehyeok Kim, and Aditya Akella. 2025. Enabling Portable and High-Performance SmartNIC Programs with Alkali. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 107–126. <https://www.usenix.org/conference/>

- nsdi25/presentation/lin-jiaxin
- [20] Jianshen Liu, Carlos Maltzahn, Craig D. Ulmer, and Matthew Leon Curry. 2021. Performance Characteristics of the BlueField-2 SmartNIC. *CoRR* abs/2105.06619 (2021). arXiv:2105.06619 <https://arxiv.org/abs/2105.06619>
 - [21] Marvell. 2020. Marvell OCTEON TX2 DPDK Overview. <https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-octeon-tx2-dpdk-whitepaper.pdf>
 - [22] Benjamin Michalowicz, Kaushik Kandadi Suresh, Hari Subramoni, Dhableswar Panda, and Steve Poole. 2023. DPU-Bench: A Micro-Benchmark Suite to Measure Offload Efficiency Of SmartNICs. In *Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good* (Portland, OR, USA) (PEARC '23). Association for Computing Machinery, New York, NY, USA, 94–101. doi:10.1145/3569951.3593595
 - [23] Benjamin Michalowicz, Kaushik Kandadi Suresh, Hari Subramoni, Dhableswar K. Panda, and Steve Poole. 2023. Battle of the BlueFields: An In-Depth Comparison of the BlueField-2 and BlueField-3 SmartNICs. In *2023 IEEE Symposium on High-Performance Interconnects (HOTI)*. 41–48. doi:10.1109/HOTI59126.2023.00020
 - [24] NVIDIA Corporation. 2021. *NVIDIA BlueField-3 DPU: Programmable Data Center Infrastructure On-a-chip*. NVIDIA Corporation. <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf>
 - [25] NVIDIA Corporation. 2025. Power and Thermal - BlueField Troubleshooting Guide. <https://networking-docs.nvidia.com/bfswtroubleshooting/power-and-thermal>
 - [26] NVIDIA Corporation & Affiliates. 2024. *NVIDIA MLX5 Crypto Driver*. NVIDIA Corporation & Affiliates. <https://doc.dpdk.org/guides-24.03/cryptodevs/mlx5.html>
 - [27] NVIDIA Corporation & Affiliates. 2025. *DOCA Flow - DOCA Documentation*. NVIDIA Corporation & Affiliates. <https://docs.nvidia.com/doca/archive/3-1-0/doca+flow/index.html>
 - [28] NVIDIA Corporation & Affiliates. 2025. Single Point of Resource Distribution - DOCA SDK Documentation. <https://networking-docs.nvidia.com/doca/archive/3-1-0/single-point-of-resource-distribution>
 - [29] NVIDIA Corporation & Affiliates. 2026. DPA Subsystem - DOCA Documentation. <https://docs.nvidia.com/doca/sdk/dpa-subsystem/index.html>
 - [30] NVIDIA Corporation & Affiliates. 2026. *NVIDIA BlueField-4*. NVIDIA Corporation & Affiliates. <https://resources.nvidia.com/en-us-accelerated-networking-resource-library/bluefield-4-dpu-datasheet>
 - [31] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. 2021. Automated SmartNIC Offloading Insights for Network Functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) (SOSP '21). Association for Computing Machinery, New York, NY, USA, 772–787. doi:10.1145/3477132.3483583
 - [32] Joseph A. Salowey, Abhijit Choudhury, and David McGrew. 2008. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288. doi:10.17487/RFC5288
 - [33] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=B1ckMDqlg>
 - [34] The Linux Kernel Organization. 2020. *Filesystem-level encryption (fsencrypt)*. The Linux Kernel Organization. Accessed: 2025-12-24.
 - [35] The Open Information Security Foundation. 2026. Suricata. <https://suricata.io/>
 - [36] Max Uritsky. 2023. Announcing the general availability of Azure Boost. <https://techcommunity.microsoft.com/blog/azureinfrastructureblog/announcing-the-general-availability-of-azure-boost/3981384>
 - [37] Amin Vahdat and Nirav Mehta. 2023. Titanium: A robust foundation for workload-optimized cloud computing. <https://cloud.google.com/blog/products/compute/titanium-underpins-googles-workload-optimized-infrastructure>
 - [38] Xingda Wei, Rongxin Cheng, Yuhang Yang, Rong Chen, and Haibo Chen. 2023. Characterizing Off-path SmartNIC for Accelerating Distributed Systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 987–1004. <https://www.usenix.org/conference/osdi23/presentation/wei-smartnic>
 - [39] Shaofeng Wu, Qiang Su, Zhixiong Niu, and Hong Xu. 2025. Performance Prediction of On-NIC Network Functions with Multi-Resource Contention and Traffic Awareness. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) (ASPLOS '25). Association for Computing Machinery, New York, NY, USA, 828–842. doi:10.1145/3669940.3707232
 - [40] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Songyuan Sui, Khalid Manaa, Omer Shabtai, Yonatan Piasetzky, Matty Kadosh, Arvind Krishnamurthy, T. S. Eugene Ng, and Ang Chen. 2023. Unleashing SmartNIC Packet Processing Performance in P4. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) (ACM SIGCOMM '23). Association for Computing Machinery, New York, NY, USA, 1028–1042. doi:10.1145/3603269.3604882
 - [41] Tong Xing, Hesam Tajbakhsh, Israat Haque, Michio Honda, and Antonio Barbalace. 2022. Towards portable end-to-end network performance characterization of SmartNICs. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems* (Virtual Event, Singapore) (APSys '22). Association for Computing Machinery, New York, NY, USA, 46–52.

doi:10.1145/3546591.3547528

A Accelerator Selection and Baseline

This section details the preliminary screening of accelerator performance used to inform our case study selection. This screening evaluates the standalone and concurrent processing capabilities of various lookaside accelerators to identify their degree of architectural isolation, by repeatedly submitting requests from the ARM cores to the accelerators and measuring the throughput in isolation and concurrently. These screening results, summarized for representative cases in Figure 11, demonstrate that functional similarity does not strictly necessitate shared underlying hardware resources. For instance, the decompression accelerator acts as a shared execution engine for both LZ4 and Deflate operations, creating a throughput bottleneck during concurrent execution. Conversely, the cryptographic offloads exhibit surprising performance independence; the AES-XTS and AES-GCM units reach their standalone throughput limits simultaneously without measurable interference. This observed isolation, coupled with performance variations across different workload profiles, leads us to hypothesize that despite sharing a common software API, these engines are implemented as physically distinct hardware units with differing topographical placements.

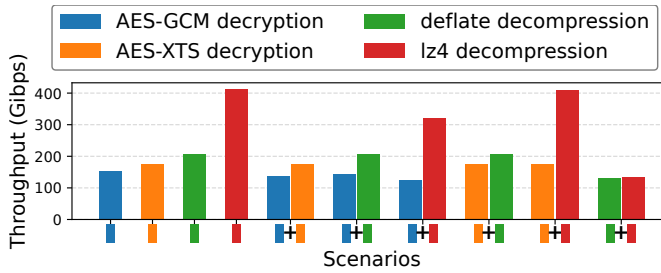


Fig. 11. Accelerator throughput under standalone and concurrent operation.

B DPA Resource Baselines

B.1 Evaluating Baseline Memory Performance

We measure the standalone read and write throughput of DPA threads accessing both memory regions via repeated memory operations in an ARM-to-DPA RPC call. With 254 DPA threads, DPA memory sustains 104 Gbps for reads and 192 Gbps for writes; ARM memory sustains 147 Gbps for reads and 111 Gbps for writes. These baselines align with prior observations [7], confirming that the two memory pools exhibit distinct performance profiles: DPA-dedicated memory provides superior write throughput, whereas ARM memory delivers more balanced read/write performance.

B.2 Evaluating Baseline DPA I/O Performance

We next evaluate the DPA's baseline I/O performance by measuring the packet sending and receiving throughput of DPA threads. Because network I/O requires packet buffers, we compare provisioning these buffers in either DPA-dedicated memory or ARM memory to draw an analogy with the memory access experiments. Single-core send throughput is 17.7 Gbps with DPA memory buffers and 17.4 Gbps with ARM memory buffers; single-core receive throughput is 21.3 Gbps with either buffer location. With 254 DPA threads, aggregate send throughput reaches 213 Gbps with DPA memory buffers and 320 Gbps with ARM memory buffers; aggregate receive throughput

reaches 48 Gbps and 180 Gbps, respectively. While these measurements confirm the observation from BenchBF3 [7] that ARM memory buffers are practically necessary to maximize aggregate throughput, DPA memory buffers actually offer marginally superior per-core performance.

C Sweeping the Rate of Competing Traffic

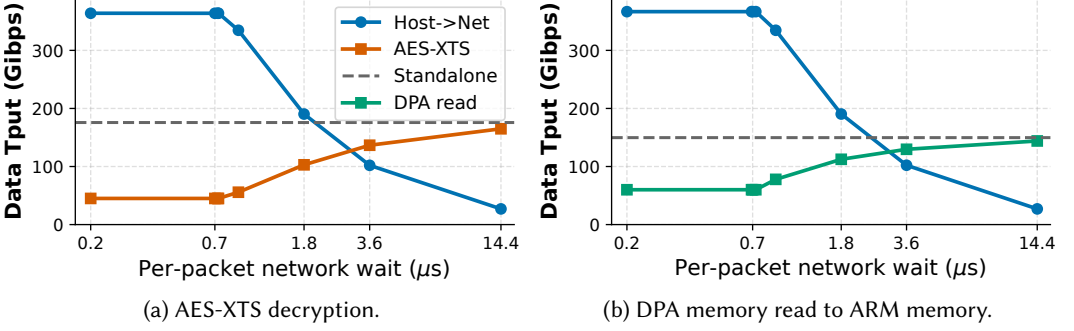


Fig. 12. Contention behavior under paced competing Host-to-Network traffic flow.

For completeness, alongside the queue-count and message-size sweeps of §4.2, we also sweep the *rate* of the competing traffic. Specifically, we pace a Host-to-Network flow by inserting a per-packet wait on the sender (the x -axis of Figure 12, higher means more throttled) and repeat two microbenchmarks under it: AES-XTS decryption (§4, Figure 12a) and DPA memory reads from ARM memory (§5, Figure 12b). In both cases, the two flows' shares shift smoothly across the sweep. As the competing flow is paced down, the offloaded workload's throughput rises monotonically and returns to within a few percent of its standalone level once the competing flow is largely paced out.

Received 10 July 2026; revised 5 October 2026; accepted 9 October 2026