# POSTER: Astraea: Enforcing DPU Performance Isolation in Public Clouds

Qiyang Peng[1], Menghao Zhang[1], Feiyang Wang[1], Guanyu Li[2], Chunming Hu[1]
[1]Beihang University [2]Unaffiliated

## ABSTRACT

The absence of effective performance isolation mechanisms hinders deploying Data Processing Units (DPUs) in public clouds for multi-application co-location scenarios. To address this challenge, we introduce Astraea, an efficient DPU performance isolation framework targeting DPU-specific computational resources. Astraea overcomes significant obstacles posed by proprietary, high-level SDK interfaces and coarse-grained First-Come-First-Served (FCFS) scheduling via resource occupation profiling, task splitting and workload-guided scheduling. Our open-source prototype on NVIDIA BlueField-3 DPUs reduces SLA violation rates for latency-sensitive applications from 47.66% to just 14.84% versus natives, while introducing less than 4% performance overhead.

## CCS CONCEPTS

• **Hardware** → **Networking hardware**; • **Networks** → *Network performance modeling*;

## KEYWORDS

Data Processing Unit; Performance Isolation;

**ACM Reference Format:**
Qiyang Peng, Menghao Zhang, Feiyang Wang, Guanyu Li, Chunming Hu. 2025. POSTER: Astraea: Enforcing DPU Performance Isolation in Public Clouds. In *ACM SIGCOMM 2025 Posters and Demos (SIGCOMM Posters and Demos '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3744969.3748397

## 1 INTRODUCTION

Hyperscalers have been deploying DPUs throughout their datacenters as strategic infrastructure components. DPUs, equipped with heterogeneous hardware modules such as ASIC NICs, ASIC accelerators, data-path accelerators (DPAs), and ARM CPUs, enable efficient offloading of datacenter infrastructure tasks and even portions of application workloads [8]. Despite their advantages, it is difficult to deploy DPUs in public clouds for multi-application co-location scenarios because of the absence of effective and efficient performance isolation mechanisms, which cannot ensure fairness and maintain SLAs for metrics (e.g., bandwidth, and latency). Without performance isolation, co-located applications on a single DPU can experience significant performance degradation. For instance, Yala [18] demonstrates that one co-located application's performance can drop by 60% compared to solo deployment.

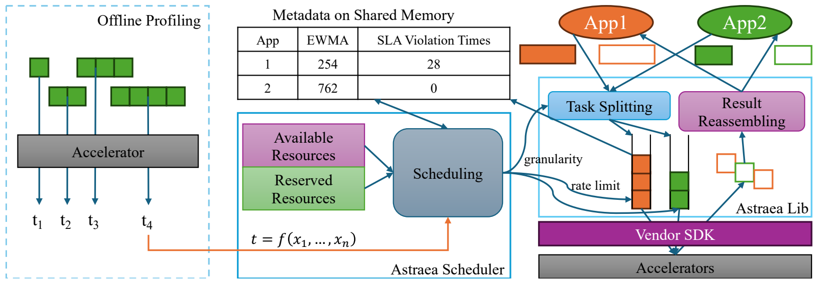
Prior works on performance isolation for RNICs [5, 10, 17, 19] and SmartNICs [7, 11, 13, 14] fall short for DPU scenarios. Justitia [19], Tassel [17], and Harmonic [10] successfully enforce fairness of bandwidth, throughput, and latency among multiple tenants, but do not address the unique computing resources on DPUs, particularly ASIC accelerators and DPAs. FairNIC [7] proposes a task-level performance isolation solution for ASIC accelerators on SmartNIC, which fails to prevent large tasks from monopolizing accelerator and violates latency SLAs for smaller, delay-sensitive tasks. Other approaches like Panic [9] and S-NIC [20] propose novel SmartNIC designs with built-in isolation strategies, but often depend on advanced hardware features (e.g., multiple hardware thread accelerators) that commercial DPUs either lack entirely or do not expose through accessible interfaces, making these strategies impractical.

Designing an effective DPU performance isolation framework faces several significant challenges. First, many DPU vendors provide only high-level, proprietary interfaces (e.g., NVIDIA DOCA [6]), creating a "black box" relationship between input parameters and resource utilization. This lack of transparency prevents developers from understanding how these APIs leverage the underlying accelerators and makes direct hardware resource allocation control impossible. Second, hardware tasks are typically scheduled using a FCFS approach at the coarse granularity of the SDK API calls. Each task must wait for all previously submitted tasks to complete, regardless of its size or priority requirements. This causes large tasks to monopolize resources and violates latency SLAs for smaller, time-sensitive operations, resulting in unfair performance.

Through theoretical analysis of the underlying mathematical properties of these DPU accelerators, we discover that large tasks can be divided into smaller subtasks that, when executed sequentially and their results properly combined, produce identical outcomes to executing the original large task. This insight enables us to implement a transparent shim layer between applications and SDK APIs that breaks down large tasks and performs fine-grained scheduling without modifying application code. Based on this insight, we propose Astraea, an efficient DPU performance isolation framework, focusing primarily on DPU-specific computational resources. Astraea employs three key techniques. First, Astraea quantifies resource occupation by profiling the relationship between input parameters and hardware execution time prior to deployment, establishing a foundation for fair resource allocation despite the "black box" nature of accelerators. Second, Astraea implements accelerator-specific task splitting and result reassembling strategies that are completely transparent to users, enabling fine-grained scheduling without application modifications. Third,

This work is supported in part by the National Natural Science Foundation of China (No. 62402025) and the Fundamental Research Funds for the Central Universities. Menghao Zhang is the corresponding author.

*SIGCOMM Posters and Demos '25, September 8–11, 2025, Coimbra, Portugal*
© 2025 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 979-8-4007-2026-0/25/09.
https://doi.org/10.1145/3744969.3748397



| App | EWMA | SLA Violation Times |
|---|---|---|
| 1 | 254 | 28 |
| 2 | 762 | 0 |

**Figure 1: ASTRAEA architecture and workflow.**

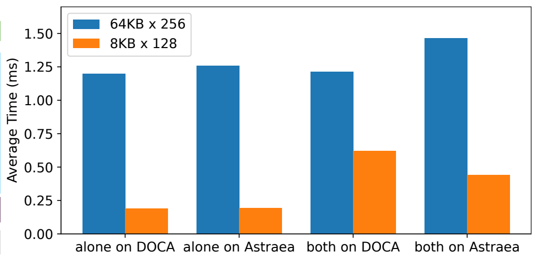


**Figure 2: Workloads' completion time.**

ASTRAEA features a novel workload-guided scheduling algorithm that dynamically allocates resources among applications based on real-time workload patterns. We implement an open-source prototype of ASTRAEA on NVIDIA BlueField-3 DPU [4], and our preliminary experiments demonstrate significant improvements in SLA compliance compared to native implementations, particularly for latency-sensitive applications.

## 2 ASTRAEA DESIGN

Figure 1 presents the architecture of the ASTRAEA framework, with the following three key modules.

**Resource Occupation Profiling.** Prior to deployment, ASTRAEA builds a precise model of each accelerator's resource utilization by measuring task execution times across various parameter configurations. These measurements are used to fit mathematical functions per task type (e.g., erasure coding, AES-GCM encryption), expressing the relationship between input parameters and execution time. The resulting functions serve as the foundation for effective resource allocation decisions by the scheduler during the deployment phase, enabling accurate prediction of task execution times even without visibility into the accelerator's internal architecture.

**Task Splitting and Result Reassembling.** ASTRAEA designs specialized task segmentation and result reorganization algorithms for each accelerator type based on the underlying mathematical relationships between inputs and outputs. For erasure coding [16], the core operation involves matrix multiplication of a source data matrix $D^{b\times d}$ and a coding matrix $E^{d\times r}$, generating a redundant matrix $R^{b\times r}$. Leveraging the distributive property of matrix multiplication, we can divide $D^{b\times d}$ into $n$ submatrices $D_1^{b_1\times d}, D_2^{b_2\times d}, \ldots, D_n^{b_n\times d}$. By multiplying each submatrix with $E^{d\times r}$ independently, we obtain corresponding result submatrices $R_1^{b_1\times r}, R_2^{b_2\times r}, \ldots, R_n^{b_n\times r}$, which can be concatenated to reconstruct the complete result matrix $R^{b\times r}$. Other accelerators such as AES-GCM [12, 15] and LZ4 [3] decompression can be processed in a similar way.

However, this splitting and reassembling approach introduces significant memory manipulation overhead during execution. To mitigate this, we leverage DOCA's scatter/gather list function, constructing $n$ scatter/gather lists, each representing one of $n$ subtasks with $d$ buffers, enabling zero-copy submission of subtasks to the accelerator while preserving fine-grained scheduling. Additionally, we pipeline asynchronous accelerator operations with post-execution memory operations to reduce latency and enhance accelerator utilization.

**Workload-Guided Task Scheduling.** ASTRAEA uses a workload-guided task scheduling approach that balances performance isolation with efficient resource utilization. To prevent interference between different accelerator workloads, ASTRAEA maintains separate task queues for each application per accelerator type. At the beginning of each scheduling period, the scheduler traverses task queues across all applications for each accelerator and allocate resources to applications according to priority and fairness policies. And an internal thread in ASTRAEA periodically checks the available resources allocated to that application and submits tasks. Latency-sensitive tasks obtain execution priority to maintain their stricter SLAs. Between scheduling periods, the scheduler remains dormant, saving CPU resources.

To optimize utilization, ASTRAEA dynamically allocates resources based on application behavior. The scheduler monitors consumption via Exponentially Weighted Moving Average (EWMA) [1], building a predictive model for future demands. These guide task allocations, redirecting resources to active apps. Additionally, these workload predictions inform the task splitter's granularity decisions—maintaining larger task sizes when latency-sensitive workloads are minimal and an application is projected to consume substantial resources, thus reducing splitting overhead. As a safeguard against prediction inaccuracies, ASTRAEA reserves a small portion of resources, which can be allocated to applications experiencing SLA violations due to unexpected workload fluctuations.

## 3 EVALUATION AND FUTURE WORK

We implement an open-source ASTRAEA prototype [2] on BlueField-3 DPU using DOCA 2.9.1 in ~1,500 lines of C/C++ code. We evaluated it with a latency-sensitive app (8KB tasks, 128 submissions; SLA: completion time $\leq$ 2× hardware time) and a bandwidth-sensitive app (64KB tasks, 256 submissions; SLA: bandwidth $\geq$ 50% solo bandwidth). Figure 2 shows completion times (blue: bandwidth-sensitive; orange: latency-sensitive). Concurrently, ASTRAEA reduces latency-sensitive SLA violations from 47.66% to 14.84%, though bandwidth-sensitive tasks see increased completion time due to resource sharing—a normal outcome in multi-application co-location scenarios, unlike DOCA's native approach where small tasks suffer significantly. Alone, ASTRAEA adds less than 4% overhead vs. native, by skipping unnecessary splitting for single apps.

In future work, we plan to: 1) completely implement and evaluate ASTRAEA in production environments with real-world workloads; 2) extend ASTRAEA with telemetry capabilities that allow cloud providers to monitor fine-grained resource usage and performance metrics across co-located applications.

## REFERENCES

[1] J. Stuart Hunter and. 1986. The Exponentially Weighted Moving Average. *Journal of Quality Technology* 18, 4 (1986), 203–210. https://doi.org/10.1080/00224065.1986.11979014 arXiv:https://doi.org/10.1080/00224065.1986.11979014

[2] Astraea. 2025. Astraea. https://github.com/Networked-System-and-Security-Group/Astraea. (2025).

[3] Yann Collet. 2011. LZ4 Library. https://github.com/lz4/lz4. (2011).

[4] NVIDIA Corporation. 2023. NVIDIA BlueField-3 Data Processing Unit. https://www.nvidia.com/en-us/networking/products/data-processing-unit/. (2023). Accessed: 2025-03-26.

[5] NVIDIA Corporation. 2025. NVIDIA ConnectX-7 RNIC. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/connectx-7-datasheet. (2025). Accessed: 2025-03-26.

[6] NVIDIA Corporation. 2025. NVIDIA DOCA Software Framework. https://developer.nvidia.com/doca. (2025). Accessed: 2025-03-26.

[7] 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 (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 681–693. https://doi.org/10.1145/3387514.3405895

[8] Jinghan Huang, Jiaqi Lou, Srikar Vanavasam, Xinhao Kong, Houxiang Ji, Ipoom Jeong, Danyang Zhuo, Eun Kyung Lee, and Nam Sung Kim. 2024. HAL: Hardware-assisted Load Balancing for Energy-efficient SNIC-Host Cooperative Computing. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 613–627. https://doi.org/10.1109/ISCA59077.2024.00051

[9] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 243–259. https://www.usenix.org/conference/osdi20/presentation/lin

[10] Jiaqi Lou, Xinhao Kong, Jinghan Huang, Wei Bai, Nam Sung Kim, and Danyang Zhuo. 2024. Harmonic: Hardware-assisted RDMA Performance Isolation for Public Clouds. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1479–1496. https://www.usenix.org/conference/nsdi24/presentation/lou

[11] Marvell. 2025. Marvell LiquidIO II SmartNIC. https://www.marvell.com/products/infrastructure-processors/liquidio-smart-nics/liquidio-ii-smart-nics.html. (2025). Accessed: 2025-03-26.

[12] David A. McGrew and John Viega. 2005. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *Progress in Cryptology - INDOCRYPT 2004*, Anne Canteaut and Kapaleeswaran Viswanathan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 343–355.

[13] Napatech. 2025. Napatech 400G SmartNIC N3070X. https://www.napatech.com/support/resources/data-sheets/n3070x-smartnic/. (2025). Accessed: 2025-03-26.

[14] Netronome. 2024. Netronome Agilio SmartNIC. https://netronome.com/agilio-smartnics/. (2024). Accessed: 2025-03-26.

[15] National Institute of Standards and Technology (NIST). 2007. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Technical Report Special Publication 800-38D. National Institute of Standards and Technology (NIST).

[16] I. S. Reed and G. Solomon. 1960. Polynomial Codes Over Certain Finite Fields. *J. Soc. Indust. Appl. Math.* 8, 2 (1960), 300–304. https://doi.org/10.1137/0108018 arXiv:https://doi.org/10.1137/0108018

[17] Zilong Wang, Xinchen Wan, Luyang Li, Yijun Sun, Peng Xie, Xin Wei, Qingsong Ning, Junxue Zhang, and Kai Chen. 2024. Fast, Scalable, and Accurate Rate Limiter for RDMA NICs. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 568–580. https://doi.org/10.1145/3651890.3672215

[18] 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 (ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 828–842. https://doi.org/10.1145/3669940.3707232

[19] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. 2022. Justitia: Software Multi-Tenancy in Hardware Kernel-Bypass Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1307–1326. https://www.usenix.org/conference/nsdi22/presentation/zhang-yiwen

[20] Yang Zhou, Mark Wilkening, James Mickens, and Minlan Yu. 2024. SmartNIC Security Isolation in the Cloud with S-NIC. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 851–869. https://doi.org/10.1145/3627703.3650071