HardMesh: Enabling High-performance Service Mesh Ingress Processing with SmartNICs

Myoungsung You
University of Seoul

Jaehyun Nam
Dankook University

Minjae Seo
ETRI

Taejune Park
Chonnam National University

Seungwon Shin
KAIST

Abstract

Service meshes have become essential for enabling microservices in cloud environments; however, they also introduce substantial network overhead. In particular, the ingress gateway, which serves as the primary entry point for external traffic, has emerged as a major performance bottleneck due to CPU-intensive traffic analysis and prolonged forwarding paths through multiple network stack layers. Our analysis indicates that these inefficiencies can result in a 4-fold reduction in network throughput and increased CPU resource consumption. In response, we propose *HardMesh*, a hardware-software hybrid ingress gateway that leverages a SmartNIC for high-performance traffic analysis and efficient traffic routing. This process is augmented by a lightweight CPU-based proxy for traffic management. Evaluations show that *HardMesh* outperforms existing ingress gateways, achieving up to 4.4× higher throughput while providing the same range of traffic management services.

CCS Concepts

• **Computer systems organization → Cloud computing**.

Keywords

Cloud computing, Service mesh, Containers, SmartNIC, eBPF

ACM Reference Format:

Myoungsung You, Jaehyun Nam, Minjae Seo, Taejune Park, and Seungwon Shin. 2025. HardMesh: Enabling High-performance Service Mesh Ingress Processing with SmartNICs. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3718958.3750471

1 Introduction

Service meshes [11] have become the de facto standard for managing communications among cloud-native microservices [1]. In service meshes, microservices are deployed across multiple containers [10], each isolated within its own network environment. To manage inter-service communication, a sidecar proxy is attached to every container, intercepting and processing all ingress and egress traffic on behalf of the container. While this abstraction separates communication logic from the application, it introduces significant performance overhead due to inherent complexity [18]

SIGCOMM '25, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/25/09
https://doi.org/10.1145/3718958.3750471

Most previous studies [13, 14, 16–18] have focused on reducing overheads caused by sidecars, which introduce extra network hops within container communication. However, we identify a more significant bottleneck: *the ingress gateway*. This gateway acts as the entry point, dispatching external packets to their destination containers. Furthermore, it provides various traffic management services [9], such as circuit breaking and canary deployments. Despite its critical role, this gateway emerges as a main bottleneck in service meshes. We identify two key challenges behind this bottleneck.

Challenges. The first one is the *CPU-intensive traffic analysis task*. Since most microservices handle L7 protocols, such as REST API [12] and gRPC, dispatching packets to the destination containers requires extensive parsing of variable-length L7 fields and execution of multiple regular expression (regex) matches. For example, routing HTTP traffic requires inspection of hostnames, paths, and user-defined ones. This complexity is further amplified by the need to manage a large set of policies associated with active services. The second challenge arises from the *long packet forwarding path*. As the ingress gateway itself runs as a separate container, forwarding packets from the gateway to destination containers involves multiple interactions with the kernel network stack and the virtual switch connecting containers [13], resulting in repeated context switching and packet copies. Our evaluation shows that these two challenges can lead to a reduction in HTTP throughput of up to 4 times, even with 100 clients and a single routing rule.

Our approach. To address these challenges, we propose *HardMesh*, a novel hybrid ingress gateway combining hardware acceleration with flexible software control. *HardMesh* leverages a SmartNIC [8] to offload the CPU-intensive traffic analysis tasks from host CPUs. A lightweight CPU-side proxy complements this by forwarding packets to their destination and handling various traffic management services based on the analyzed results provided by the SmartNIC. *HardMesh* also introduces a direct socket-based data transfer mechanism, bypassing the kernel network stack and the virtual switch during packet forwarding. Evaluations show that *HardMesh* outperforms existing ingress gateways [5, 6], achieving up to 4.4× higher throughput and 2.87× higher power efficiency.

2 System Design

The functionalities of an ingress gateway comprise traffic analysis, traffic management, and traffic relaying. Although offloading all these tasks to hardware may appear beneficial for performance, our analysis identifies that full offloading is ineffective, as ingress gateways perform complex tasks that are not well-suited to hardware implementation. For example, traffic management requires flexible error handling, and relaying requires direct interaction

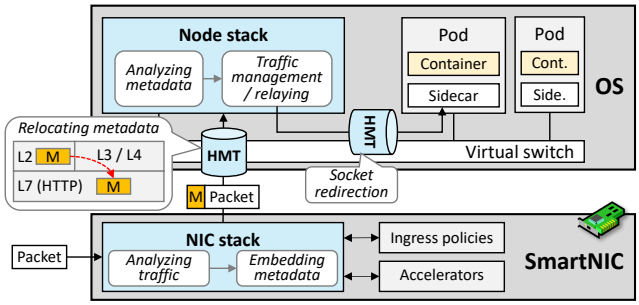

Figure 1: The overall architecture of *HardMesh* system.

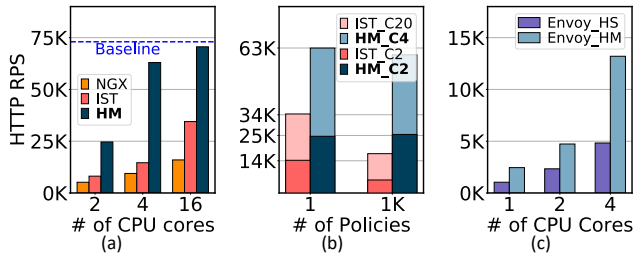

Figure 2: HTTP performance comparison result.

with containers. Hardware alone cannot efficiently support these tasks due to their complexity. Thus, we adopt a hardware-software hybrid design that partitions gateway functionalities between the SmartNIC and CPU based on their respective strengths.

Figure 1 shows the architecture of *HardMesh*, which integrates a hardware-based NIC stack on the SmartNIC with a lightweight, software-based node stack, connected via the *HardMesh* transport (HMT). In this design, incoming packets are initially processed by the NIC stack. It parses headers and payloads, comparing them to policies stored in NIC memory. Hardware accelerators enable efficient parallel processing of regex matching tasks. The analysis results are embedded into the packet's L2 header as a 24-bit metadata before it is forwarded to the host. HMT operates before the kernel network stack, providing essential support for the node stack. It identifies packets from the NIC stack and relocates their metadata from the L2 header to a designated field in the payload (HTTP method field), allowing the node stack running in the user space to access it. The node stack identifies the destination container and relevant policies for each packet by utilizing the embedded metadata, eliminating CPU-intensive traffic analysis. It then applies required traffic management services and relays the packet to the destination. Note that packets without metadata, which cannot be processed by the NIC stack, are handled by the node stack as in software ingress gateways. Finally, HMT enables direct redirection of packets from the node stack's Tx socket to the destination container's Rx socket, bypassing the kernel stack and virtual switch.

3 Evaluation and conclusion

We implement a prototype of *HardMesh* on a BlueField-2 SmartNIC [2] and evaluate it on a Kubernetes-based service mesh. We compare *HardMesh* (HM) against two widely-used ingress gateways: Istio [5] (IST) and Nginx [6] (NGX). We deploy eight web server pods, each using one CPU core, as a single service endpoint. HTTP traffic is generated from an external client machine.

Table 1: Cost effectiveness and power efficiency comparison.

Hardware setting	*Normalized*			*Cost-effect.*	*Power-efficiency*
	Price	*Power*	*Throughput*		
CPU-only	$1,000 [4, 7]	100W	1K RPS	1.0	1.0
Hybrid	$921	73.8W	2.1K RPS	**2.3**	**2.87**

E2E performance. As shown in Figure 2 (a), *HardMesh* achieves up to 4.4× higher throughput than the other ingress gateways when using the same number of CPUs. Notably, *HardMesh* with 4 cores sustains 85% of the baseline throughput (no ingress gateway), whereas Istio and Nginx achieve only 41% throughput even when using 16 cores. This advantage is further amplified when a high number of policies (URL patterns) are enforced. As depicted in Figure 2 (b), *HardMesh* with less than a 5% drop when enforcing 1K policies, regardless of the number of cores, whereas Istio ingress with 20 cores (IST_C20) suffers a twofold drop. These results highlight the efficiency of our hybrid design.

We further compare *HardMesh* to an optimized software-only solution utilizing Hyperscan [15] for traffic analysis. Since no existing ingress gateway supports Hyperscan, we employ the Envoy proxy's beta-level Hyperscan matching feature [3]. Specifically, we compare two Envoy proxies under 1K policies: one configured with Hyperscan matching (Envoy-HS) and the other utilizing metadata matching enabled by the NIC stack (Envoy-HM). As shown in Figure 2 (c), Envoy-HM delivers twice the throughput of Envoy-HS when using the same number of CPU cores. While Hyperscan can enhance software matching efficiency, its optimization is less effective for the short L7 fields (e.g., hostnames and paths) typically used in service mesh routing [15]. In contrast, our hybrid design offloads these CPU-intensive tasks to the dedicated hardware device.

Cost analysis. Finally, we evaluate the power efficiency and cost-effectiveness of *HardMesh* (4 cores) in comparison with Istio ingress (20 cores) using server-grade CPUs [4]. For a fair comparison, the price and power consumption of a normal NIC [7], which provides the same bandwidth as the SmartNIC employed in this study, are included in the assessment of Istio ingress. As shown in Table 1, our hybrid design achieves 2.3× higher cost-effectiveness and 2.8× greater power efficiency when handling 1K policies by distributing workloads between the SmartNIC and CPU cores. These metrics are significant in cloud environments, where they have a substantial impact on total cost of ownership (TCO).

Conclusion. We identify the ingress gateway as the main performance bottleneck in the service mesh and propose a hybrid design that partitions gateway tasks between hardware and software components. Evaluations show that this hybrid design outperforms software-only gateways in terms of performance and cost efficiency.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00212738).

References

[1] 2022. CNCF Annual Survey 2022. https://www.cncf.io/reports/cncf-annual-survey-2022/.

[2] 2022. NVIDIA BLUEFIELD-2 DPU. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[3] 2023. Service Mesh - Envoy Regular Expression Matching Acceleration with Hyperscan User Guide. https://www.intel.com/content/www/us/en/content-details/761773/service-mesh-envoy-regular-expression-matching-acceleration-with-hyperscan-user-guide.html.

[4] 2024. Intel® Xeon® Silver 4114 Processor. https://www.intel.com/content/www/us/en/products/sku/123550/intel-xeon-silver-4114-processor-13-75m-cache-2-20-ghz/specifications.html.

[5] 2024. Istio Ingress Gateways. https://istio.io/latest/docs/tasks/traffic-management/ingress/ingress-control/.

[6] 2024. NGINX Ingress Controller. https://docs.nginx.com/nginx-ingress-controller/.

[7] 2024. NVIDIA Mellanox® MCX623106AN-CDAT 100GbE ConnectX-6 Dx Card Crypto Disabled. https://www.tes-itsolutions.com/product-page/refurbished-nvidia-mcx623106an-cdat-100gbe-connectx-6-dx-card-crypto-disabled.

[8] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure accelerated networking:{SmartNICs} in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 51–66.

[9] Istio. 2025. Istio Ingress: Circuit Breaking. https://istio.io/latest/docs/tasks/traffic-management/circuit-breaking/.

[10] Hui Kang, Michael Le, and Shu Tao. 2016. Container and microservice driven design for cloud infrastructure devops. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 202–211.

[11] Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han. 2019. Service mesh: Challenges, state of the art, and future research opportunities. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 122–1225.

[12] Guozhi Liu, Bi Huang, Zhihong Liang, Minmin Qin, Hua Zhou, and Zhang Li. 2020. Microservices: architecture, container, and challenges. In *2020 IEEE 20th international conference on software quality, reliability and security companion (QRS-C)*. IEEE, 629–635.

[13] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-chin Wang, and KK Ramakrishnan. 2022. SPRIGHT: extracting the server from serverless computing! high-performance eBPF-based event-driven, shared-memory processing. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 780–794.

[14] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-Chin Wang, and KK Ramakrishnan. 2024. : High-Performance eBPF-Based Event-Driven, Shared-Memory Processing for Serverless Computing. *IEEE/ACM Transactions on Networking* (2024).

[15] Xiang Wang, Yang Hong, Harry Chang, KyoungSoo Park, Geoff Langdale, Jiayu Hu, and Heqing Zhu. 2019. Hyperscan: A fast multi-pattern regex matcher for modern {CPUs}. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 631–648.

[16] Myoungsung You, Jaehyun Nam, Minjae Seo, and Seungwon Shin. 2023. HELIOS: Hardware-assisted High-performance Security Extension for Cloud Networking. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*. 486–501.

[17] Myoungsung You, Minjae Seo, Jaehan Kim, Seungwon Shin, and Jaehyun Nam. 2024. Hyperion: Hardware-based High-performance and Secure System for Container Networks. *IEEE Transactions on Cloud Computing* (2024).

[18] Xiangfeng Zhu, Guozhen She, Bowen Xue, Yu Zhang, Yongsu Zhang, Xuan Kelvin Zou, XiongChun Duan, Peng He, Arvind Krishnamurthy, Matthew Lentz, et al. 2023. Dissecting overheads of service mesh sidecars. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*. 142–157.