Article

Not peer-reviewed version

# Performance And Latency Efficiency Evaluation Of Kubernetes Container Network Interfaces For Built-In And Custom Tuned Profiles

Vedran Dakić *, Jasmin Redžepagić *, Matej Bašić, Luka Žgrablić

Posted Date: 14 August 2024

doi: 10.20944/preprints202408.0970.v1

Keywords: Kubernetes; CNI; Antrea; Geneve; containers; Docker; Flannel; Cilium; Calico; HPC

 doi:10.20944/preprints202408.0970.v1



*Article*

# Performance and Latency Efficiency Evaluation of Kubernetes Container Network Interfaces for Built-In and Custom Tuned Profiles

**Vedran Dakic *, Jasmin Redzepagic *, Matej Basic and Luka Zgrablic**

Department of Operating Systems, Algebra University, 10000 Zagreb, Croatia; mbasic2@algebra.hr (M.B.); lzgrabl@algebra.hr (L.Z.)
* Correspondence: vedran.dakic@algebra.hr (V.D.); jasmin.redzepagic@algebra.hr (J.R.)

**Abstract:** In the era of DevOps, developing new toolsets and frameworks that leverage DevOps principles is crucial. This paper demonstrates how Ansible's powerful automation capabilities can be harnessed to manage the complexity of Kubernetes environments. This paper evaluates efficiency across various CNI (Container Network Interface) plugins by orchestrating performance analysis tools across multiple power profiles. Our methodology, which involves managing numerous components, dependencies, and configurations, requires high expertise and precision. Our performance evaluations across network interfaces with different theoretical bandwidths gave us a comprehensive understanding of CNI performance and overall efficiency. Our research confirms that certain CNIs are better suited for specific use cases, mainly when tuning our environment for smaller or larger network packages and workload types, but also that there are configuration changes we can make to mitigate that. This paper also provides research into how to use performance tuning to optimize the performance and efficiency of our CNI infrastructure, with practical implications for improving the performance of Kubernetes environments in real-world scenarios, particularly in more demanding scenarios such as High-Performance Computing (HPC) and Artificial Intelligence (AI).

**Keywords:** Kubernetes; CNI; ANTREA; Geneve; containers; Docker; Flannel; Cilium; Calico; HPC

## 1. Introduction

Kubernetes is increasingly emerging as the primary technology in HPC and AI applications due to its robust capabilities in automating containerized applications' deployment, scaling, and administration. Initially created by Google, it has quickly become the widely accepted standard for container orchestration because of its robust architecture, adaptability, and scalability. It has dramatically transformed how organizations deploy and manage their software by effectively handling large clusters of hosts, both on-premises and in the cloud. This has made it an essential tool for modern IT infrastructure.

Scalable and dependable computing environments are of utmost importance in high-performance computing. HPC and AI workloads frequently encompass intricate simulations, data analysis, and computational tasks that require significant computational capacity and effective resource utilization. Kubernetes meets these requirements by offering a scalable infrastructure that can adapt to changing workloads, guaranteeing efficient resource usage and maximum availability.

For instance, let's consider the situation where you need to oversee several Kubernetes clusters specifically designed for different workloads. This could include managing clusters for high-bandwidth video streaming and traditional web server applications. Video streaming applications necessitate resilient and low-latency network setups to effectively manage substantial incoming data volumes, whereas web servers prioritize swift response times and consistent availability amidst fluctuating workloads. Kubernetes enables the customization of clusters to effectively meet diverse requirements by providing an extensible architecture and support for custom network configurations

 doi:10.20944/preprints202408.0970.v1



through CNIs. The type of workload is directly related to the kind of traffic that workloads will use, as we expect that there will be a massive difference between the TCP and UDP performance in these scenarios. We will analyze both scenarios to get a picture of the whole performance.

Nevertheless, the wide range of CNIs offered by Kubernetes poses challenges in assessing and selecting the most suitable CNI for workloads and environments. An automated and orchestrated testing methodology utilizing tools like Ansible is highly valuable. Ansible, renowned for its straightforwardness and robust automation capabilities, can automate various CNI deployments and testing across multiple Kubernetes clusters. Automating these processes makes conducting thorough and replicable tests that assess performance metrics like throughput, latency, and resource consumption across various load scenarios feasible. The critical point is that efficient communication between nodes is vital for high performance, especially in massively parallel systems [1].

Implementing a standardized testing methodology provides numerous advantages. Firstly, it guarantees uniformity in testing, as each CNI is assessed under the same conditions, eliminating any variations and potential prejudices in the outcomes. The ability to repeat experiments is essential for making well-informed decisions, as it offers dependable data that can be utilized to compare the performance and appropriateness of various CNIs for specific applications. Furthermore, automation dramatically diminishes the time and exertion needed to carry out these tests, facilitating quicker repetition and more frequent testing cycles. This flexibility is especially advantageous in dynamic environments where new computer network interfaces (CNIs) and updates are regularly released.

Moreover, implementing a standardized and automated testing framework allows for the democratization of the evaluation process, enabling a more comprehensive range of individuals and organizations, including smaller entities and individual developers with limited resources, to participate in the testing process without requiring extensive manual testing. Facilitating a lucid and user-friendly protocol for CNI assessment enhances transparency and fosters the implementation of optimal network configuration and optimization techniques.

In 2022, Shengyan Zhu, Caiqiu Zhou, and Yongjian Wang introduced a novel hybrid wireless-optical on-chip network as a solution to address the increasing data interaction difficulties in IoT HPC systems. This concept optimizes multicast message transmission by incorporating wireless single-hop transmission to minimize communication distance and an on-chip surface waveguide for reliable wireless transmission. An adjustable routing technique ensures efficient coordination between the optical and wireless layers, alleviating the transmission burdens of each layer individually. Compared to traditional optical NoC with mesh topology, their simulations significantly improve average delay, energy consumption, and network performance [2].

In 2022, Y. Shang, Xingwen Guo, B. Guo, Haixi Wang, Jie Xiao, and Shanguo Huang proposed the use of Deep Reinforcement Learning (DRL) to create a flexible optical network for HPC systems. This network can adapt to changes in traffic efficiently. The network can switch between several topologies to optimize performance seamlessly. The network prototype was evaluated utilizing Advantage Actor-Critic (A2C) and Deep Deterministic Policy Gradient (DDPG) algorithms, leading to a performance enhancement of up to 1.7 times. This approach offers a promising alternative for efficiently controlling traffic in HPC systems. However, further investigation is necessary to enhance the deep reinforcement learning (DRL) models and easily integrate them into existing HPC infrastructures [3].

In their 2023 paper, Harsha Veerla, Vinay Chindukuri, Yaseen Junaid Mohammed, Vinay Palakonda, and Radhika Rani Chintala analyze the rapid advancement of HPC systems and their use in several domains, including machine learning and climate modeling. They emphasize the need for advanced hardware and software technologies to enhance performance. The study highlights the importance of HPC in shaping the advancement of future networking. It addresses complex computing issues through parallel processing and specialized hardware. It is recommended that future research focuses on developing optimization algorithms to improve performance management in HPC networks [4].

The 2022 study by Min Yee Teh, Zhenguo Wu, M. Glick, S. Rumley, M. Ghobadi, and Keren Bergman examines the performance constraints in reconfigurable network topologies built explicitly

 doi:10.20944/preprints202408.0970.v1



for HPC systems. The user is comparing two types of networks: ToR-reconfigurable networks (TRNs) and pod-reconfigurable networks (PRNs). The authors found that PRNs demonstrate better overall trade-offs and considerably outperform TRNs in high fan-out communication patterns. This study highlights the importance of tailoring network architecture based on traffic patterns to improve performance. It implies that further investigation is necessary to optimize these structures for broader uses [5].

In their 2022 paper, Po Yuan Su, D. Ho, Jacy Pu, and Yu Po Wang investigate the implementation of the Fan-out Embedded Bridge (FO-EB) package to meet the demands of HPC applications, particularly the necessity for fast data rates and high-speed transmission, by integrating chiplets. The FO-EB package offers enhanced flexibility and higher electrical performance than traditional 2.5D packages. The paper highlights the challenges in efficiently controlling heat and proposes techniques to enhance the capability of FO-EB packages in dealing with thermal problems. Additional research is necessary to improve the integration process to enhance performance and reduce expenses [6].

In 2021, Pouya Haghi, Anqi Guo, Tong Geng, A. Skjellum, and Martin C. Herbordt conducted a study to investigate the impact of workload imbalance on the performance of in-network processing in HPC systems. After examining five proxy applications, it was concluded that 45% of the total time used for execution on the Stampede2 compute cluster is wasted due to an uneven distribution of work. This inefficiency highlights the need for enhanced workload distribution strategies to optimize network performance. It is recommended that future research focuses on creating effective methods for distributing workloads across network resources [7].

In 2023, Steffen Christgau, Dylan Everingham, Florian Mikolajczak, Niklas Schelten, Bettina Schnor, Max Schroetter, Benno Stabernack, and Fritjof Steinert introduced a communication stack explicitly designed to integrate Network-attached Accelerators (NAA) that utilize FPGA technology in HPC data centers. The planned stack uses high-speed network connections to enhance energy efficiency and adaptability. The empirical findings achieved using a 100 Gbps RoCEv2 exhibit performance were close to the theoretical upper limits. This approach offers a promising alternative for improving the efficiency of HPC applications. However, further research is necessary to explore potential enhancements and incorporations [8].

In their 2022 publication, S. Varrette, Hyacinthe Cartiaux, Teddy Valette, and Abatcha Olloh present a detailed report on the University of Luxembourg's initiatives to enhance and reinforce its HPC networks. The study significantly improved the capacity and performance by merging Infiniband and Ethernet-based networks. The technique consistently maintained a high bisection bandwidth and exhibited minor performance limitations. The solutions could serve as a model for other HPC centers aiming to enhance their network infrastructures. Further research should prioritize implementing additional scalability and optimization strategies [9].

The 2021 study was authored by Yijia Zhang, Burak Aksar, O. Aaziz, B. Schwaller, J. Brandt, V. Leung, Manuel Egele, and A. Coskun introduces a Network-Data-Driven (NeDD) job allocation paradigm that seeks to enhance the performance of HPC by considering real-time network performance data. The NeDD framework monitors network traffic and assigns tasks to nodes based on the level of network sensitivity. The studies showcased an average reduction in execution time of 11% and a maximum reduction of 34% for parallel applications. Additional research should improve the NeDD framework to suit complex network scenarios and integrate it into broader HPC environments [10].

In their 2021 study, researchers T. Islam and Chase Phelps have collaborated to create a course that focuses on instructing students in constructing scalable HPC systems that prioritize the reduction of parallel input/output (I/O). The curriculum emphasizes addressing the challenges presented by extensive parallelism and different architectures. The course provides students with the skills and expertise to effectively navigate complex current HPC systems through hands-on training and real-world situations. Additional research could analyze the effectiveness of these instructional techniques in producing skilled HPC specialists [11].

The 2021 paper authored by Fulong Yan, Changchun Yuan, Chao Li, and X. Deng presents FOSquare, an innovative interconnect topology designed for HPC systems. FOSquare leverages rapid

 doi:10.20944/preprints202408.0970.v1



Optical Switches (FOS) and distributed rapid flow control to enhance performance. The architecture maximizes bandwidth, latency, and power efficiency performance. The simulation findings indicate that FOSquare outperforms traditional Leaf-Spine topologies by reducing latency by over 10% in actual traffic conditions. It is recommended to provide higher importance to investigating the implementation of FOSquare in real-world HPC environments and improving its efficiency [12].

The 2021 editorial by Jifeng He, Chenggang Wu, Huawei Li, Yang Guo, and Tao Li introduces a special issue focused on energy-efficient designs and dependability in HPC systems. The selected papers cover many subjects, including FPGA-based neural network accelerators and frameworks for resource allocation. The issue highlights the importance of developing HPC systems that are both energy-efficient and reliable to meet the demands of growing applications. Additional research should be undertaken to examine innovative approaches for enhancing the effectiveness and durability of high-performance computing networks [13].

Saptarshi Bhowmik, Nikhil Jain, Xin Yuan, and A. Carried out a research investigation in 2021. Using extensive system simulations, Bhatele thoroughly analyzed how hardware design parameters affect the efficiency of GPU-based HPC platforms. The study examines the number of GPUs assigned to each node, the capacity of network links, and the policies that drive NIC scheduling. The findings suggest that these parameters significantly influence the efficiency of real-world HPC tasks. Additional research is required to fine-tune these parameters to improve the performance and efficiency of future HPC systems [14].

The individuals mentioned include Mohak Chadha, Nils Krueger, Jophin John, Anshul Jindal, M. Gerndt, and S. Published a research article in 2023. Benedict explores the potential of WebAssembly (Wasm) to distribute HPC applications that utilize the Message Passing Interface (MPI). The MPIWasm embedder enables the efficient execution of WebAssembly (Wasm) code and supports high-performance networking interconnects. The experimental findings indicate that MPIWasm performs similarly and significantly reduces binary sizes. This approach offers a promising option for overcoming the challenges of designing container-based HPC applications. Future investigations should concentrate on improving WebAssembly (Wasm) for HPC environments [15].

In their 2021 study, Yunlan Wang, Hui Liu, and Tianhai Zhao present Hybrid-TF, a hybrid interconnection network topology that combines the advantages of both direct and indirect networks. The topology is meticulously designed to fit precisely with the communication patterns of HPC applications, lowering the time it takes for communication to occur and improving the overall performance. The NS-3 network simulator's simulation findings indicate that Hybrid-TF outperforms traditional topologies in various traffic patterns. Additional research is required to further apply Hybrid-TF in real-world HPC systems and improve performance [16].

In their paper released in 2022, E. Suvorova proposes using a unified and flexible router core to establish various networks within HPC systems. The strategy aims to eliminate fluctuations in network segments and guarantee a dependable Quality of Service (QoS). The suggested design's capacity for dynamic reconfiguration allows it to adjust to various attributes of data streams, leading to a decrease in data transmission and network congestion. Future investigations should prioritize implementing this methodology in practical HPC systems and evaluating its effectiveness [17].

Researchers P. did a study in 2022. Maniotis, L. Schares, D. Kuchta, and B. Karaçali analyzed the benefits of integrating co-packaged optics in high-radix switches built explicitly for HPC and data center networks. The increased escape bandwidth enables the deployment of complex network structures that demonstrate improved network closeness and reduced switch requirements. Simulations show significant performance improvements and reduced execution times for large-scale applications. This strategy offers a promising solution for improving the capability of networks in the future. Nevertheless, further investigation is necessary to enhance and integrate the technology into existing infrastructures [18].

In their paper released in 2022, H. Lin, Xi-Zhang Hsu, Long-Yuan Wang, Yuan-Hung Hsu, and Y. Wang examine the thermal challenges associated with the integration of sizable chiplets using Fan-out Embedded Bridge (FOEB) packages for HPC applications. Two proposed alternatives include the

 doi:10.20944/preprints202408.0970.v1



adoption of thick Through Mold Via (TMV) structures and utilizing Metal TIM treatments. These procedures are intended to enhance the thermal capacity. The experimental results show that the FOEB structure, with the indicated enhancements, effectively passes reliability tests and meets the thermal performance parameters. Additional research should prioritize improving the efficacy of heat management methods to get greater efficiency [19].

The paper's authors are Nicholas Contini, B. Ramesh, Kaushik Kandadi Suresh, Tu Tran, Benjamin Michalowicz, M. Abduljabbar, H. Subramoni, and D. Authored a scholarly article in 2023. Panda introduces a communication framework that utilizes MPI to enhance the adaptability of HPC systems. This solution employs PCIe interconnects to enable message passing without necessitating specialized system setups. Advanced designs reduce the need for explicit data movement, leading to a substantial reduction of up to 50% in latency for point-to-point transfers. This technique proposes a promising method for integrating FPGAs into HPC systems, although further research is needed to improve the communication structure [20].

Jinoh Kim, M. Cafaro, J. Chou, and A. Organized a workshop in 2022. Sim primarily concentrates on developing scalable telemetry and analysis approaches for HPC and distributed systems. The program centers on the challenges of monitoring and evaluating diverse input streams from end systems, switches, and developing network elements. The goal is to create novel approaches integrating HPC and data sciences to improve performance, availability, reliability, and security. Additional research should prioritize the development of innovative methods for effective data transmission and analysis in HPC environments [21].

Narūnas Kapočius conducts a performance analysis of nine commonly used CNI plugins, including Flannel, Calico, and Weave, in virtualized and physical data center environments. The analysis is outlined in their paper titled "Overview of Kubernetes CNI Plugins Performance." The study evaluates the performance of TCP and HTTP protocols in VMware ESXi and a physical data center, assessing the performance of each plugin. Kapočius highlights the significant discrepancies in performance that result from the deployment environment, aiding in the choice of suitable plugins for specific infrastructures. Comprehensive research illustrates that specific plugins perform exceptionally well in virtualized systems, while others are better suited for physical setups. The author emphasizes the importance of understanding the particular demands of the deployment environment to make an informed selection. The study underlines that scalability and performance constraints are the primary challenges, suggesting the necessity for further research to improve the optimization of CNI plugins in different network situations. Further investigations could explore the integration of these plugins with new network technologies to enhance overall performance and dependability. Moreover, the research suggests investigating how network regulations and configurations impact the effectiveness of various CNI plugins in diverse deployment circumstances [22].

"Performance Studies of Kubernetes Network Solutions" by Narūnas Kapočius assesses the efficiency of four CNCF-recommended CNI plugins (Flannel, Calico, Weave, and Cilium) in a real-world data center environment. The study measures TCP's delay and average transmission capacity for various Maximum Transmission Unit (MTU) sizes, combined network interfaces, and segmentation offloading settings. The results reveal significant discrepancies in performance among the plugins, with specific ones exhibiting higher flexibility in different network settings. A comprehensive analysis indicates that specific plugins may perform better depending on circumstances, such as different MTU sizes or network offloading strategies. Kapočius emphasizes the need for more investigation into network interface aggregation and MTU size optimization to enhance plugin performance. Further research should investigate the impact of evolving network standards and hardware developments on the functionality of CNI plugins. Moreover, the study suggests exploring the potential benefits of integrating these plugins with software-defined networking (SDN) technologies to improve network performance and scalability [23].

In their study titled "The Performance Analysis of Container Networking Interface Plugins in Kubernetes," Siska Novianti and Achmad Basuki evaluate various Container Networking Interface (CNI) plugins, such as Flannel, Calico, Cilium, Antrea, and Kube-router. Based on their

 doi:10.20944/preprints202408.0970.v1



benchmarking tests, Kube-Router demonstrates the highest throughput, exceeding 90% of the anticipated link bandwidth for TCP transfers. The analysis highlights the efficacy of various plugins for CPU and memory usage. The authors thoroughly examine the plugins' performance across multiple network contexts, emphasizing the significance of careful selection based on specific usage scenarios. Novianti and Basuki suggest that future research should focus on improving inter-host communication and reducing the expenses associated with network virtualization. Furthermore, future studies could explore incorporating innovative network technologies and protocols to enhance the efficiency and expandability of plugins. The paper also proposes to investigate the long-lasting impacts of different CNI plugins on the overall stability and performance of massive Kubernetes deployments [24].

Shixiong Qi, Sameer G. Kulkarni, and K. Ramakrishnan evaluate various open-source CNI plugins in the paper "Assessing Container Network Interface Plugins: Functionality, Performance, and Scalability." The assessment analyzes the additional expenses and constraints associated with these plugins. The study emphasizes the significance of incorporating overlay tunnel offload support to attain optimal performance for plugins that use overlay tunnels. The study also examines the capacity to manage growing workloads and the efficiency of HTTP queries. This implies that further research should be undertaken to enhance the efficiency of data processing and minimize the startup time of a pod. The authors stress the need to understand the trade-offs linked to CNI plugins to make informed judgments for specific network environments. Future research should prioritize creating novel techniques to save expenses and enhance the capacity to manage bigger workloads in different deployment scenarios. Furthermore, the study suggests integrating advanced network monitoring technologies to improve comprehension and mitigate the performance constraints associated with different CNI plugins [25].

"Seamless Hardware-Accelerated Kubernetes Networking" is a scholarly article written by Mircea M. Iordache-Sica, Tula Kraiser, and O. Komolafe, introduces the HAcK architecture, which integrates hardware networking appliances to enhance the efficiency of Kubernetes in data centers. The article presents the HAcK LoadBalancer, a system that utilizes top-notch network appliances to accomplish the integrated administration of applications and infrastructure. The performance study in real-world data center scenarios shows significant improvements while recognizing challenges in smoothly integrating hardware acceleration. Additional research is required to broaden the range of network functions that can be supported by hardware acceleration. The authors suggest investigating the benefits of combining software-based solutions with hardware acceleration to achieve optimal efficiency. Furthermore, the study proposes to research the impact of different hardware configurations on the performance of CNI plugins. This will help identify the optimal setups for application requirements [26].

T.P. Nagendra and R. Hemavathy's research paper, titled "Unlocking Kubernetes Networking Efficiency: Exploring Data Processing Units for Offloading and Enhancing Container Network Interfaces," examines the use of DPUs to delegate networking tasks and improve the efficiency of Kubernetes. The report analyzes prominent CNI plugins and demonstrates the benefits of DPUs through case studies. The authors suggest undertaking additional research on integrating DPU and its impact on different Kubernetes cluster requirements. Furthermore, they highlight the capacity of DPUs to enhance network performance and dependability in various deployment settings. Future research should prioritize developing efficient ways to employ DPUs and explore their interaction with emerging network technologies. Moreover, the paper proposes evaluating the cost-benefit analysis of deploying DPUs in large-scale Kubernetes environments to determine their feasibility and efficacy [27].

The authors of the study titled "Understanding Container Network Interface Plugins: Design Considerations and Performance," Shixiong Qi, Sameer G. Kulkarni, and K. Ramakrishnan, did a comprehensive assessment of various CNI plugins, analyzing the effects and constraints on performance resulting from network models, host protocol stacks, and iptables rules. The study suggests that while choosing CNI plugins, it is essential to consider the dominant communication type, whether it is within a single host or between numerous hosts. Furthermore, the paper proposes

 doi:10.20944/preprints202408.0970.v1



performing additional research to reduce costs and improve the execution of network policies. The authors provide helpful perspectives on the trade-offs linked to CNI plugins and their impact on the overall network performance. Future studies could explore innovative methods to enhance network policies and maximize plugin performance in various environments. In addition, the study suggests investigating the impact of different network security configurations on the effectiveness of CNI plugins to provide optimal security without compromising performance [28].

The study, titled "A Comprehensive Performance Evaluation of Different Kubernetes CNI Plugins for Edge-based and Containerized Publish/Subscribe Applications" by Zhuangwei Kang, Kyoungho An, A. Gokhale, and Paul N Pazandak, investigates the impact of Kubernetes network virtualization on DDS applications. The study assesses various CNI plugins on hybrid edge/cloud K8s clusters, specifically examining throughput, latency, and resource utilization. The findings provide valuable insights into choosing a suitable network plugin for latency-sensitive applications. Additionally, they propose the necessity for further investigation into enhancing the costs connected with virtualization. The authors emphasize the significance of investigating edge computing environments' unique requirements further and developing tailored solutions to meet these needs. Furthermore, the paper proposes exploring sophisticated network optimization strategies to enhance the efficiency and reliability of DDS applications in Kubernetes systems [29].

Ritik Kumar and M. Trivedi conducted a comprehensive analysis of Kubernetes networking and CNI plugins in the article "Networking Analysis and Performance Comparison of Kubernetes CNI Plugins." The study performs benchmark tests to assess various plugins, focusing on differences in performance concerning cost, maintenance, and fault tolerance. The authors propose further investigation into multi-host communication and ingress management to enhance network performance holistically. In addition, they suggest investigating innovative approaches to improve the scalability and reliability of Kubernetes networking solutions. Moreover, the research highlights the importance of continuous monitoring and adjusting CNI plugins to maintain the best possible network performance in complex and extensive setups [30].

A paper by Ayesha Ayub, Mubeena Ishaq, and Muhammad Munir presents a new CNI plugin for Multus that improves network performance by removing the need for additional plugin configurations. The study demonstrates significant enhancements in the efficiency of DPDK (Data Plane Development Kit) applications in cloud-native environments. It highlights the necessity for further investigation into smoothly integrating sophisticated networking functionalities into CNI plugins. The authors also highlight the possible benefits of combining DPDK with other high-performance networking technologies for optimal results. Additional research is required to assess the capacity of the proposed solutions to be readily expanded and adjusted in various scenarios to validate their practicality in real-life contexts [31].

An article written by Andrea Garbugli, Lorenzo Rosa, A. Bujari, and L. Foschini has just unveiled a cutting-edge technology known as KuberneTSN. This Kubernetes network plugin enables expedited communication. The study showcases KuberneTSN's ability to meet predictable deadlines and outperform alternative overlay networks. Future research should focus on prioritizing the enhancement of support for diverse application requirements and the optimization of the scalability of time-sensitive networks. The authors suggest investigating the integration of KuberneTSN with supplementary real-time communication protocols to improve performance further. Furthermore, the study proposes exploring the possible benefits of incorporating KuberneTSN with sophisticated scheduling and resource management approaches to enhance the overall system performance [32].

The article by Youngki Park, Hyunsik Yang, and Younghan Kim investigates and compares the efficiency of various CNI plugins in both OpenStack and Kubernetes environments. The study highlights crucial performance metrics and suggests implementing containerized cloud applications. Further study is recommended to optimize computer network infrastructure (CNI) configuration and improve network performance in various contexts. The authors stress the significance of regularly monitoring and fine-tuning CNI parameters to guarantee optimal performance. Moreover, the study suggests researching the impact of multiple hardware configurations and virtualization technologies

 doi:10.20944/preprints202408.0970.v1



on the performance of CNI plugins. This research aims to identify the most efficient setups for application requirements [33].

A research paper by Francesco Lombardo, S. Salsano, A. Abdelsalam, Daniel Bernier, and C. Filsfils examines the integration of SRv6 with the Calico CNI plugin. The study demonstrates the benefits of SRv6 in large-scale and distributed scenarios involving many data centers. It highlights the exceptional performance and characteristics of SRv6. Future research should focus on enhancing control plane architectures and expanding support for SRv6 in Kubernetes environments. The authors suggest exploring new uses for SRv6 in different networking situations to utilize its strengths fully. Additionally, the study proposes to investigate the impacts of alternative SRv6 configurations on the efficiency and scalability of Kubernetes installations to optimize results in diverse settings [34].

The study is authored by Larsson, William Tarneberg, C. Klein, E. Elmroth, and M. Kihl's study titled "Impact of etcd deployment on Kubernetes, Istio, and application performance" explores the vital importance of the etcd database in Kubernetes deployments. The study examines the impact of etcd on the performance of Kubernetes and Istio, emphasizing the need for a storage system that operates at a high level of performance. Further work is required to analyze the process of optimizing etcd setups and resolving unanticipated behaviors in cloud environments. The authors propose investigating innovative methods to enhance the reliability and effectiveness of etcd in large-scale deployments. Moreover, the study suggests integrating state-of-the-art storage technologies with etcd to improve performance and reliability in dynamic and demanding environments [35].

The paper by Yining Ou presents a new approach to improving CNI metrics for better network performance. The study significantly improves network latency, bandwidth, and packet loss by optimizing CNI settings. The authors suggest further studies to enhance the calibration of CNI measurements to enhance service levels in different cloud settings. Future research may focus on developing automated ways to optimize CNIs and simplify the management of container networks. Additionally, the study proposes to investigate the impact of varied workloads and network circumstances on the performance of improved CNI metrics to ensure their efficiency in diverse deployment scenarios [36].

The study by Nupur Jain, Vinoth Mohan, Anjali Singhai, Debashis Chatterjee, and Dan Daly. The article showcases the application of P4 in developing a high-performance load balancer for Kubernetes. The study examines the challenges associated with size, security, and network performance, highlighting the benefits of using P4 data planes. Future investigations should focus on expanding P4-based solutions to include more network functions and enhancing their interface with Kubernetes. The authors suggest exploring the possible benefits of combining P4 with other advanced networking technologies to achieve maximum performance and scalability in various deployment scenarios. Moreover, the study proposes to investigate the impact of different P4 configurations on the effectiveness of Kubernetes load balancers to identify the most optimal setups for particular application requirements [37].

The paper by D. Milroy et al. investigates integrating HPC operations into Kubernetes. The study demonstrates advancements in scaling MPI-based systems and improving scheduling capabilities using the Fluence plugin. The authors propose further investigation to enhance the efficiency of Kubernetes in high-throughput computing (HPC) settings and optimize the scheduler's performance. Furthermore, the research highlights the importance of understanding the specific requirements of HPC workloads to develop tailored solutions for their efficient coordination in Kubernetes contexts. Additional research is required to analyze the incorporation of sophisticated resource management methods to enhance the efficiency and expandability of high-performance computing processes in cloud-native systems [38].

The authors, Ki-Hyeon Kim, Dongkyun Kim, and Yong-hwan Kim, present a new approach called Container Network Interface (CNI) in their article "Open-Cloud Computing Platform Design based on Virtually Dedicated Network and Container Interface." This approach, which combines Software-Defined Networking (SDN) technology with Kubernetes, is a significant advancement in cloud computing. The study demonstrates substantial improvements in network performance and flexibility. The authors stress the importance of further inquiries to prioritize the expansion of this

 doi:10.20944/preprints202408.0970.v1



methodology to include more cloud environments and enhance network efficiency. They also suggest investigating the benefits of integrating advanced software-defined networking (SDN) capabilities with the new container network interface (CNI) to obtain the best possible performance and scalability. Furthermore, the study proposes to examine the impact of different network configurations and virtualization technologies on the performance of the new CNI to identify the most optimal setups for particular application requirements [39].

The paper, "Characterising resource management performance in Kubernetes," written by Víctor Medel Gracia et al., investigates the performance of Kubernetes using a Petri nets-based model. The study highlights the importance of efficiently allocating and scheduling resources for adaptable cloud systems. Future research should focus on improving performance models and optimizing resource allocation in Kubernetes contexts. The authors emphasize the significance of continuously monitoring and modifying resource management systems to guarantee optimal performance. Additionally, the study proposes to investigate the impact of different types of workloads and network circumstances on the efficacy of Kubernetes resource management to ensure its success in diverse deployment scenarios [40].

The research, authored by Dinh Tam Nguyen et al., investigates the integration of SR-IOV with CNFs in Kubernetes to improve the performance of the 5G core network. The study shows a notable 30% improvement in network speed for 5G applications using SR-IOV and CPU-Pinning. The authors propose further investigating SR-IOV and other technologies that enhance performance in cloud-native 5G deployments. Furthermore, the study highlights the potential benefits of combining SR-IOV with other cutting-edge networking technologies to optimize performance and scalability in various deployment circumstances. It is essential to prioritize further exploration into the creation of the best methods for integrating SR-IOV and other performance-enhancing technologies with CNFs. This is necessary to ensure that these technologies are feasible, usable, and effective [41].

This paper is organized as follows. The next sections provide relevant information about all background technologies covered in this paper—Kubernetes, CNIs, and related technologies. Then, we'll describe the problem, our experimental setup, and study methodology, followed by results and the related discussion. In the last sections of the paper, we'll go over some future research directions and conclusions we reached while preparing this paper.

## 2. Technology Overview

This section will describe all the CNIs and technologies related to our paper: Antrea, Flannel, Cilium, Calico, Kubernetes, and various offload technologies.

### *2.1. Antrea*

Antrea is a CNI plugin developed explicitly for Kubernetes to improve cluster network performance and security. It utilizes Open vSwitch (OVS) to enforce network policies and achieve high performance by efficiently processing packets. Antrea supports sophisticated functionalities such as Network Policy, Traffic Engineering, and observability. These characteristics are crucial for effectively managing intricate network needs in cloud-native systems. To assess their performance and resource utilization, Novianti and Basuki [24] evaluated Antrea and other CNIs like Flannel, Calico, and Cilium. The study emphasized that Kube-Router had the best throughput. However, Antrea's architecture, which seamlessly integrates with Kubernetes networking models, makes it a reliable option for environments that need scalable and secure network solutions. In addition, Qi et al. [25] observed that Antrea's utilization of Open vSwitch (OVS) offers adaptability and effectiveness in overseeing network policies and traffic streams, which is essential for contemporary microservices architectures. The plugin's emphasis on utilizing hardware offloading capabilities boosts its performance, making it well-suited for applications that require fast data throughput and minimal delay. Budigiri et al. [42] highlighted the significance of implementing security measures with minimal impact on performance in Kubernetes systems. Antrea's features are well-suited to meet these criteria. Jain et al. [43] also examined the advantages of adaptable data plane implementations, which play a crucial role in Antrea's architecture.

 doi:10.20944/preprints202408.0970.v1



### 2.2. Flannel

Flannel is a straightforward and configurable Kubernetes CNI plugin developed to establish a flat network structure within Kubernetes clusters. The primary method employed is VXLAN encapsulation to oversee overlay networks, guaranteeing smooth communication between pods on various nodes. Novianti and Basuki [24] conducted a performance evaluation study on Flannel, which showed significant throughput, albeit it was not the highest compared to other CNIs such as Kube-Router. Flannel's design is characterized by its simplicity, which has advantages and disadvantages. On the positive side, it allows for effortless deployment and requires minimal configuration. However, it may need help to handle additional functionalities, such as network policies and traffic segmentation, which other CNIs offer. However, Flannel remains a favored option for small to medium-sized clusters that stress simplicity and ease of setup and operation. Qi, Kulkarni, and Ramakrishnan [25] emphasized that Flannel's performance can be enhanced with practical tuning, especially in network-intensive workloads. Zeng et al. [44] discovered that while not superior, Flannel's performance offers a harmonious blend of simplicity and functionality, rendering it appropriate for numerous typical Kubernetes deployments. Larsson et al. [35] emphasized the significance of considering fundamental infrastructure components such as etcd when assessing the efficiency of Kubernetes networking solutions, including Flannel.

### 2.3. Cilium

Cilium is a robust CNI plugin for Kubernetes that uses eBPF (extended Berkeley Packet Filter) to deliver efficient and secure networking. This system is precisely engineered to manage intricate network policies and offer comprehensive insight into network traffic, essential for security and performance monitoring. Novianti and Basuki [24] conducted a performance analysis that incorporated Cilium. Although Cilium did not achieve the highest throughput, its range of features and strong security capabilities make it the best option for situations with strict security needs. In their study, Qi, Kulkarni, and Ramakrishnan [25] highlighted the smooth integration of Cilium with Kubernetes, along with its advanced features like network policy enforcement and observability. Cilium utilizes eBPF to effectively manage network traffic without additional costs, rendering it well-suited for high-performance applications. Moreover, Cilium offers advanced security capabilities like seamless encryption and precise network controls, which greatly benefit organizations seeking to protect their microservices systems. Budigiri et al. [24] corroborated similar results, suggesting that eBPF-based solutions such as Cilium can offer robust security measures without sacrificing performance.

### 2.4. Calico

Calico is a viral CNI plugin recognized for its vigorous network policy enforcement and ability to scale effectively. It offers Kubernetes's Layer 3 networking and network policies, providing encapsulated and unencapsulated networking choices. In their performance benchmark, Novianti and Basuki [24] incorporated Calico and emphasized its well-balanced performance in terms of throughput and resource utilization. In their study, Qi, Kulkarni, and Ramakrishnan [25] examined the versatility of Calico in accommodating different networking contexts and its efficacy in enforcing security regulations. Calico's architectural design enables it to expand horizontally, effectively managing a substantial volume of network policies without a notable performance decline. The versatility of this choice is due to its integration with Kubernetes network policies and its ability to function in both cloud and on-premises environments. In addition, Calico's incorporation of BGP (Border Gateway Protocol) allows for the utilization of advanced routing features, which can be essential in specific deployment situations. Zeng et al. [45] discovered that Calico performs better than other assessed CNIs, making it an excellent choice for high-performance network situations. Nagendra and Hemavathy [44] highlighted the potential of CNIs such as Calico to improve performance in challenging situations by utilizing hardware offloading techniques.

 doi:10.20944/preprints202408.0970.v1



*2.5. Tuned and Performance Profiles*

Tuned is a tool that can enhance the performance of Linux systems by applying different tuning profiles according to the current system load and activity. It is a dynamic adaptive system tuning tool. It is especially beneficial in settings where the emphasis is on maximizing performance and optimizing resources.

Tuned functions by implementing a series of predetermined tuning profiles that modify system configurations to enhance performance for workloads or hardware setups. These profiles can be used for many purposes, such as optimizing overall performance, conserving power, reducing latency, or catering to specific applications like databases or virtual machines. Tuned constantly monitors the system's activity and can switch profiles to retain the best possible performance.

The tuned daemon is a crucial element of tuning as it operates in the background and implements suitable performance tuning parameters. The parameters can encompass modifications to kernel settings, CPU governor settings, disk I/O scheduler setups, and network stack optimizations. As an illustration, the latency-performance profile configures the CPU governor to performance mode to minimize latency. Still, the power save profile utilizes the power save governor to reduce power usage. We used this in our built-in and custom profile performance measurements. There are different ways of doing performance tuning via tuned, such as merging settings from pre-existing profiles or creating new options. This modification is especially beneficial in contexts with distinct performance requirements only partially handled by the preset profiles. This makes it a perfect method to deliver additional performance across an arbitrary number of systems that can be a part of, for example, an HPC cluster.

*2.6. Kubernetes*

Kubernetes, a resilient and adaptable platform, has not just revolutionized but transformed the administration and coordination of container applications. Derived from the Greek term' κυβερνήτης,' which means 'governor' or 'pilot,' Kubernetes is designed to simplify the process of deploying, scaling, and managing applications. It achieves this by abstracting the complexities of infrastructure and enabling effective resource management. Its exceptional stability, scalability, and fault tolerance have earned it widespread recognition, making it the primary choice for managing containerized workloads in cloud-native settings. Brewer (2015) states that Kubernetes enables the development of applications as collections of interconnected yet autonomous services, making it easier to evolve and scale cloud-native apps [46]. In addition, the data center deployments of Kubernetes have gained significant popularity because of its robust capabilities, including self-healing and scaling. This is exemplified by Haja et al. (2019), who developed a specialized Kubernetes scheduler to handle delay limitations and ensure reliable performance at the edge [47].

Kubernetes has demonstrated its indispensability as a tool for optimizing infrastructure. Haragi et al. (2021) performed a performance investigation that compared the deployment of AWS EC2 with local Kubernetes deployments using MiniKube. Their analysis showcased the efficacy of Kubernetes in optimizing cloud resources, minimizing over-provisioning, and improving performance [48]. In a study conducted by Telenyk et al. (2021), a comparison was made between Kubernetes and its lightweight alternatives, MicroKubernetes and K3S. The researchers found that while the original Kubernetes performs better in most aspects, the lightweight variants demonstrate superior resource efficiency in limited contexts [49].

Brewer (2018) highlights the scalability and flexibility of Kubernetes, emphasizing the transition from virtual machines to container-based services that Kubernetes and Istio control. This move facilitates the development of applications with increased simplicity, allowing for the flexible adjustment and control of services through APIs and processes [50]. In addition, Kim et al. (2021) investigated the resource management capabilities of Kubernetes, emphasizing the significance of performance isolation in guaranteeing service quality. Their research indicates that the decrease in container performance is frequently caused by competition for CPU resources rather than limitations in network bandwidth. This highlights the importance of implementing better strategies for managing resources. [51]

 doi:10.20944/preprints202408.0970.v1



The utilization of Kubernetes to automate software production environments is also noteworthy. Poniszewska-Marańda and Czechowska (2021) conducted a study to evaluate the utilization of Kubernetes clusters through Kops and Excel on AWS. The study demonstrated that the platform effectively fulfills the demands of a production environment efficiently [52]. Perera et al. (2021) studied database scaling in Kubernetes. They proposed an approach for automatically scaling PostgreSQL databases to overcome synchronization and scalability issues, assuring a high availability level [53].

## 3. Problem Statement

In the past five years, it's become trendy to ship all workloads as containers using Docker, Podman, and Kubernetes. This paper does not cover this problem, but it's directly related to the topic of this paper, albeit a portion of it—how to make those applications perform well from the networking standpoint. This is why selecting the correct CNI for any given workload is crucial. There are different types of applications from the network performance standpoint:

- bandwidth-sensitive;
- latency-sensitive;
- bandwidth and latency-sensitive.

Without proper evaluation, there's no way to know which CNI to use for which type of application.

The fundamental issue with CNIs is that they aren't all that efficient and don't scale well, especially on non-offloaded interfaces (FPGA, DPDK, etc.). Even if we went with the idea of working with offload techniques, making offloaded interfaces work is sometimes challenging. Furthermore, the performance metrics of CNIs are vastly different – usually, one CNI has 2x-3x the performance of a different one. These facts alone make it worthwhile to develop an automated methodology to be able to do computerized evaluations of CNIs, especially for larger HPC environments. This is why we created a new method for automating and orchestrating performance evaluation of CNIs [54].

Our previous projects pointed us in this direction, which then turned into research on how much overhead there is when implementing container-based infrastructure managed by Kubernetes for HPC use cases. This paper meets these challenges head-on and tries to help understand the absolute performance characteristics of CNIs and performance characteristics compared to physical networking when measured via a repeatable methodology.

Due to CNI overheads, the design of HPC data centers will need to evolve if Kubernetes is to be used for managing HPC workloads. Specifically, these overheads must be factored into the design to achieve and maintain a specific performance envelope with QoS. While this is a broader topic than what we cover in this paper, it is a fundamental issue that cannot be ignored. This paper can be a precursor to a more in-depth discussion. It is essential to acknowledge this challenge and the need for further research.

## 4. Experimental Setup and Study Methodology

Our setup consisted of multiple x86 servers, with network interfaces ranging from 1 Gbit/s to 10Gbit/s in different configurations. We also used a combination of copper and optical connections, various tuned optimization profiles, and simulated package sizes to check for any influence of a given tuned profile on real-life performance. We measured bandwidth and latency from a performance standpoint, essential for different workload types. HPC workloads are known to be sensitive to either one or both simultaneously. We also measured the performance of a physical system so that we could see an actual scope overhead influence on these performance metrics. This is where we realized we would have many challenges if we used manual testing and researched other avenues to make testing more efficient and reproducible. Manual measuring could be more efficient and requires a substantial time commitment. Establishing the testing infrastructure, adjusting the CNIs, executing the tests, and documenting the outcomes manually can be highly time-consuming and labor-intensive, particularly in extensive or intricate systems. Not only does this impede the review process, but it also heightens the probability of errors and inconsistencies.

 doi:10.20944/preprints202408.0970.v1



Our automated methodology [54] guarantees that every test is conducted under identical conditions on each occasion, resulting in a high degree of consistency and reproducibility that is challenging to attain manually. This results in increased reliability and comparability of outcomes, which is crucial for making well-informed judgments regarding the most suitable CNI for given workloads. Moreover, automation significantly decreases the time and exertion needed to carry out these tests. Tools like Ansible can optimize the deployment, configuration, and testing procedures, facilitating quicker iterations and more frequent testing cycles. This enhanced efficiency enables a more flexible evaluation procedure, making including new CNIs and updates easier.

Furthermore, manual testing can be demanding on resources, frequently necessitating specialized expertise and substantial human resources to carry out efficiently. On the other hand, an automated and coordinated testing framework allows for a more inclusive evaluation process, making it easier for a broader range of people to participate, including smaller enterprises and individual engineers. Offering a concise and user-friendly method for assessing CNI enhances transparency and fosters optimal network configuration and optimization techniques.

In our pursuit of improving computing platforms, we often neglect to step back from improvements and see if possible solutions would allow us to optimize our current environments. Part of this paper tests some basic optimization options and examines how they affect typical computing platforms' network performance.

Before we dig into the tests and their results, let us establish some basic terminology. For this paper, we will introduce three basic terms: test configuration, test optimization, and test run. A test optimization refers to a set of optimization changes made to both the server and client side of testing. A test configuration refers to testing the networking setup. A test run, therefore, combines test optimization, configuration, and settings, such as MTU, packet size, test duration, etc. Firstly, let's define our test optimizations.

The first test optimization isn't an optimization; we just have to have a baseline starting point. The first optimization level is called "default_settings," and as the name would suggest, we don't make any optimization changes during this test. This will allow us to establish a baseline performance to be used as a reference for other optimizations.

The second optimization is called "kernel_optimizations," at this optimization level, we introduce various kernel-level optimizations to the system. The Linux kernel can be a significant bottleneck for network performance if not configured properly, starting with send and receive buffers. The default value for the maximum receive and send buffer sizes for all protocols, "net.core.rmem_max and net.core.wmem_max," is 212992 (2MB), which can significantly impact performance, especially in high latency or high bandwidth environments. We increased these values to 16777216 (16MB). The second kernel-level optimization we can take advantage of is TCP window scaling. By default, TCP window scaling is turned off on Linux, and by enabling this setting, we can increase the size of the scaling factor of the TCP receive window. Further, we will disable TCP Selective Acknowledgement (TCP SACK). Disabling TCP SACK can improve performance by slightly reducing CPU overhead. Still, it should be used carefully as it can lead to a significant loss in network performance in lossy networks. The last kernel-level optimization we will introduce is increasing the SYN Queue Size to 4096 packets from the default 128. This setting might not affect our testing environment but can significantly improve performance in environments where a single server handles many connections.

The third optimization is called "nic_optimizations," at this optimization level, we introduce several network interface-level optimizations. The first important setting is enabling Generic Receive offload. GRO helps coalesce multiple incoming packets into a larger one before passing them to the networking stack, improving performance by reducing the number of packets the CPU needs to process. The second important network interface level optimization we can introduce is enabling TCP Segmentation offload. By default, the CPU handles the segmentation of network packets. By enabling this option, we can offload that work to the network interface card and reduce the work needed by the CPU. This setting could significantly improve our performance as part of our tests, which will also test network performance while forcing segmentation. The last network interface level

 doi:10.20944/preprints202408.0970.v1



optimization change we will make is to increase se the receive and transmit ring buffers from the default value of 256 to 4096 packets. We don't expect this setting to significantly impact the test results, as it will probably only affect performance during bursts of network traffic.

These are the only "custom" optimization levels used during tests. The rest are built into the tuned service available on Linux distributions. Tuned profiles used as the rest of the optimization levels are accelerator performance, hpc compute, latency performance, network latency, and network throughput.

Let us now look at different test configurations used during our testing. We tested five network setups: 1-gigabit, 10-gigabit, 10-gigabit-lag, fiber-to-fiber, and local. 1-gigabit, as the name would suggest, is a test conducted over a 1 Gbit link between the server and the client. The same applies to 10-gigabit. 10-gigabit-lag is a test conducted over the 10 Gbit links aggregated into a link aggregation group. Fiber-to-fiber is a test conducted over a fiber optic connection between the server and client. Finally, local is a local test where the server and the client are on the same physical machine.

We included a few test settings alongside these settings and configuration options in each test run. These combine two different MTU values, 1500 and 9000, and five different packet sizes, 64, 512, 1472, 9000, and 15000. Finally, all tests were conducted using TCP and UDP protocols separately.

This paper focuses on the results of using our methodology. Specifically, bandwidth and latency were analyzed across multiple Kubernetes CNIs, tuned performance profiles, MTU sizes, and different Ethernet standards/interfaces. Let's now examine the results to reach some conclusions.

## 5. Results

The results analysis is split into three parts: bandwidth, latency, and a comparison of both with the performance of an environment that doesn't use Kubernetes or CNIs. This will give us a good set of baselines to understand the results from multiple angles and the level of performance drop compared to an environment that doesn't use Kubernetes or CNIs. The results presented in the paper will be related to 10 Gigabit network connections. However, upon publication of this paper, we will make available a complete set of charts with absolute values for bandwidth and latency across multiple Ethernet standards and network interface combinations. We presented only efficiency results in this paper as they're the most important for analysis.

The color scheme is going to be the same for all the performance charts in the following sections, both for TCP and UDP performance efficiency charts:


**Figure 1.** The color scheme used for testing results in this paper.

Let's now check the bandwidth and latency efficiency testing results for these four CNIs. We'll compare TCP bandwidth and latency efficiency, followed by UDP.

### *5.1. TCP Bandwidth Efficiency Comparison with the Performance of Physical Network without CNIs*

The next set of results concerns the bandwidth comparison between CNIs and non-CNI scenarios. We changed the graph layout to make the performance drop between these two scenarios

 doi:10.20944/preprints202408.0970.v1



easier to read. When dealing with the default performance profile, the drop in performance as we move to larger packages grows to more than 5 Gbit/s, as we can see in Figure 2:

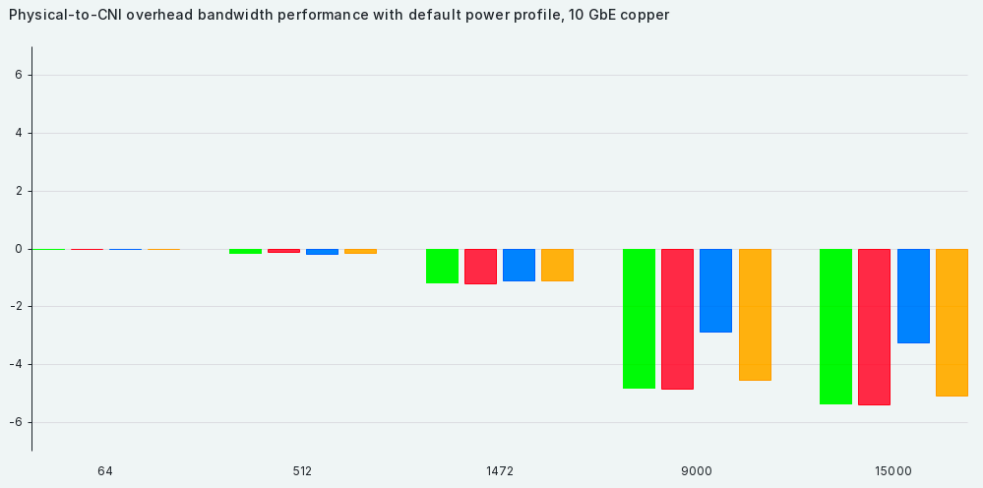


**Figure 2.** Physical to CNI comparison for TCP bandwidth in default tuned performance profile.

Switching our tuned profile to the accelerator-performance profile gets us marginally better results but still well below what we'd consider to be suitable for package sizes of 9000 or more, as can be seen in Figure 3:

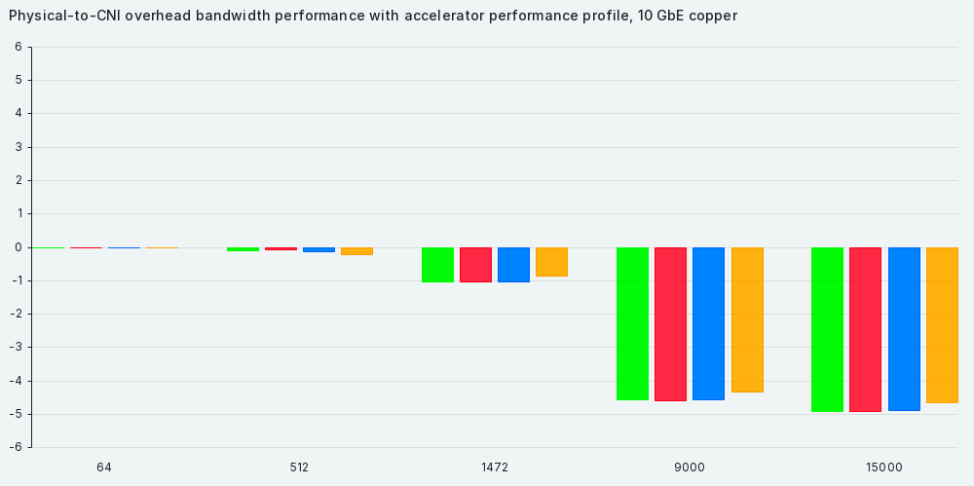


**Figure 3.** Physical to CNI comparison for TCP bandwidth accelerator-performance profile.

The following profile on our list is hpc-compute, and switching to it does almost nothing to increase TCP bandwidth when compared to a physical network scenario, as we can see in Figure 4:

 doi:10.20944/preprints202408.0970.v1



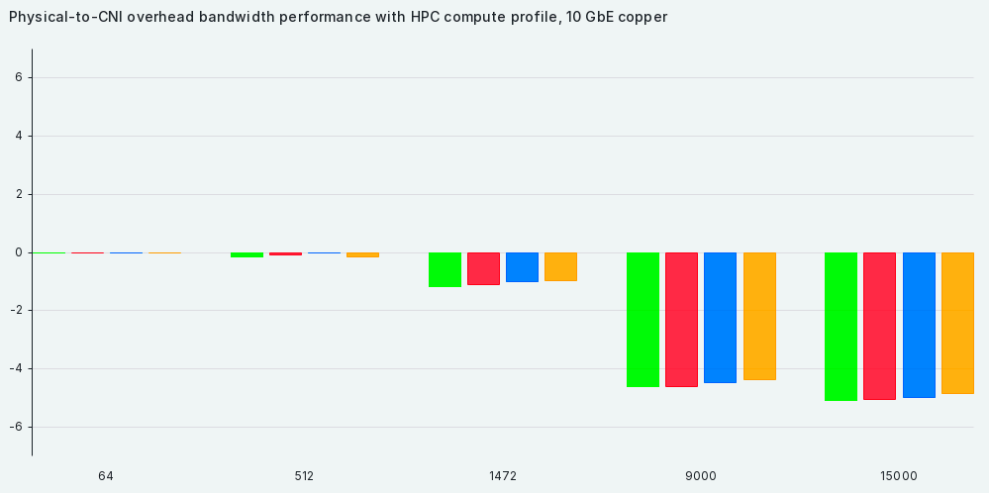


**Figure 4.** Physical to CNI comparison for TCP bandwidth in the hpc-compute performance profile.

When we switch to our kernel optimizations profile, we are not getting results that are any better, although some results get much better (for example, Calico), as we can see in Figure 5:

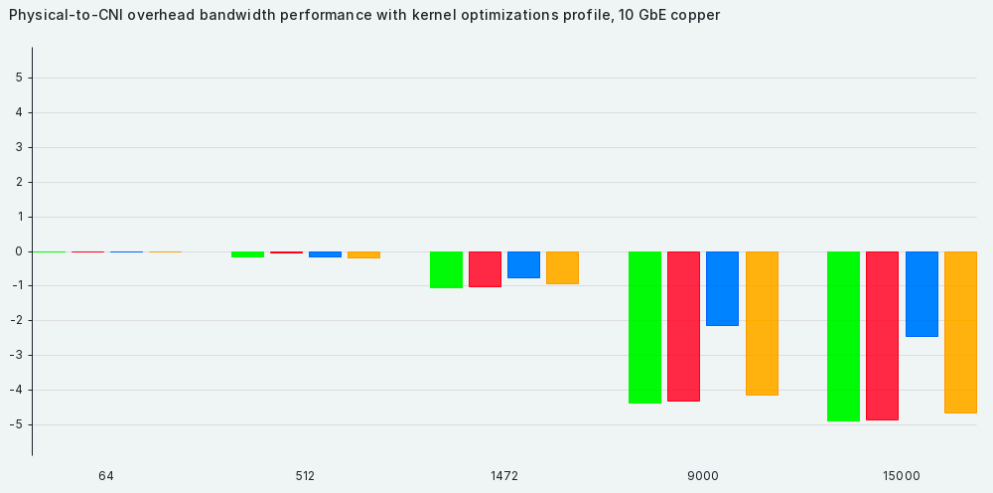


**Figure 5.** Physical to CNI comparison for TCP bandwidth in kernel optimization performance profile.

The next tuned profile on our list is the latency profile, and efficiency stays almost the same except for Calico, which drops in performance heavily, as can be seen in Figure 6:

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 August 2024 doi:10.20944/preprints202408.0970.v1



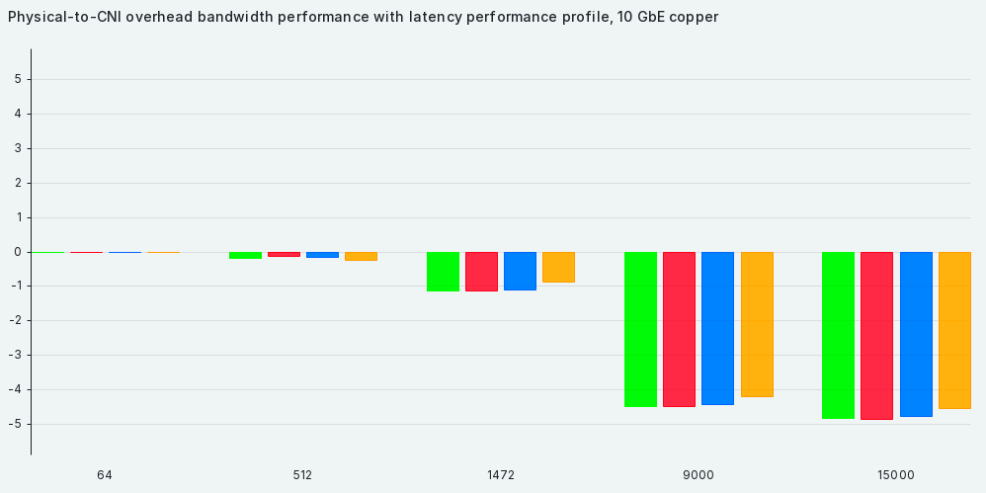


**Figure 6.** Physical to CNI comparison for TCP bandwidth in the latency performance profile.

When we switch our cluster to the network latency profile, results get marginally better for 1472 and 9000 package sizes but remain almost the same for all other package sizes, as can be seen in Figure 7:

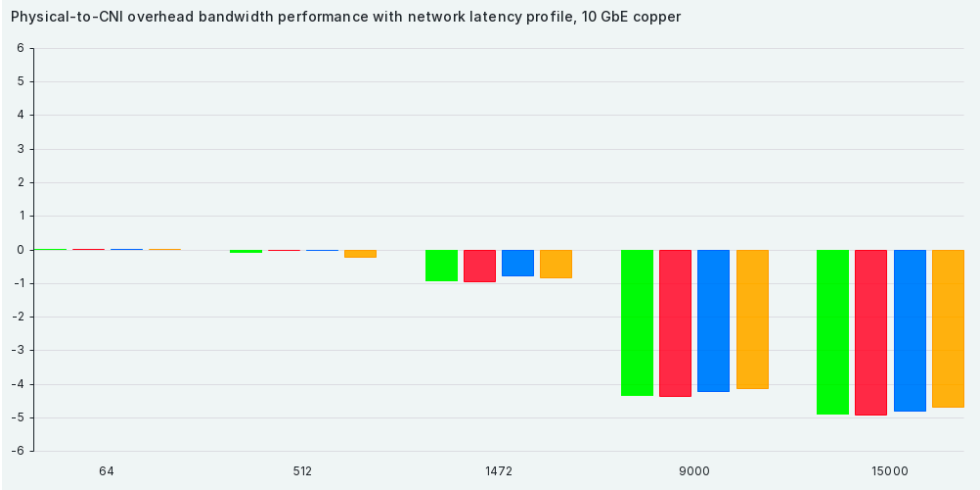


**Figure 7.** Physical to CNI comparison for TCP bandwidth in network latency performance profile.

The next profile on our list is the network throughput profile, and it offers no improvement whatsoever when compared to previous tuned profiles, as can be seen in Figure 8:

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 August 2024 doi:10.20944/preprints202408.0970.v1



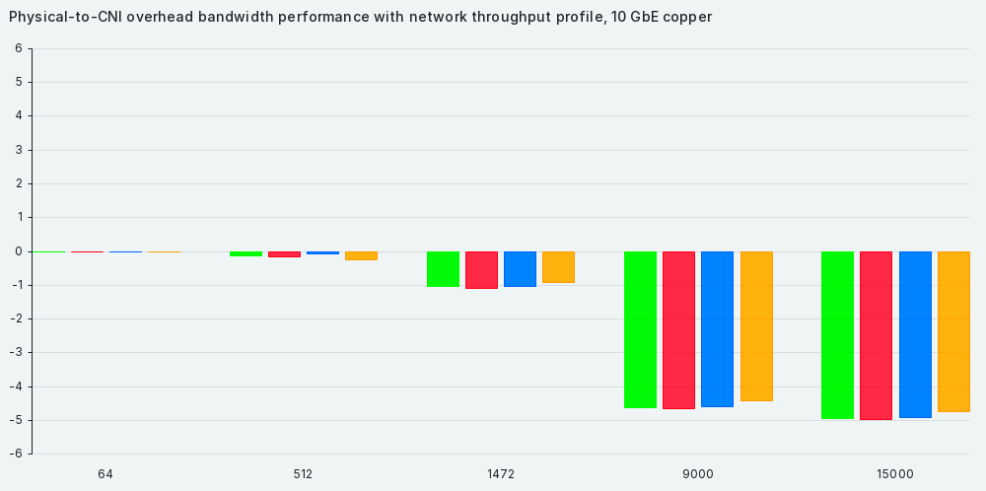


**Figure 8.** Physical to CNI comparison for TCP bandwidth in network throughput performance profile.

The last tuned profile is the NIC optimizations profile, where we see a substantial drop in performance compared to previous scenarios in 9000+ scenarios (except for Antrea), as can be seen in Figure 9:

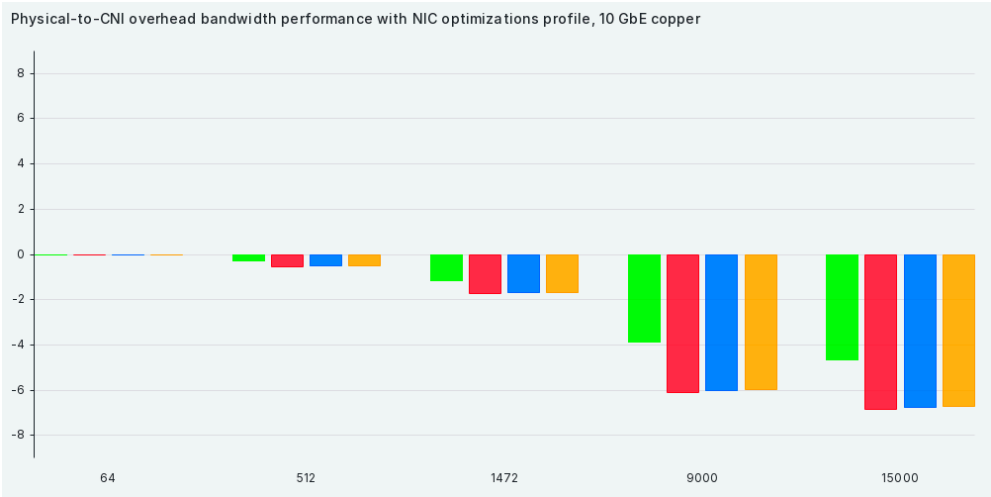


**Figure 9.** Physical to CNI comparison for TCP bandwidth in the nic-optimization performance profile.

Overall, TCP bandwidth efficiency results are okay for sub-9000 package sizes and shockingly bad for larger ones. This further underlines the point of our methodology and our paper—selecting the correct set of network settings to suit our workloads (especially HPC) and pairing that set of settings to a correct CNI is undoable without proper verification. Overall, Antrea seems to be the winner in terms of TCP bandwidth efficiency. However, there are exceptions here and there. For example, Calico seems to benefit significantly from our kernel optimizations, which significantly increase performance and efficiency in that specific tuned profile.

Let's move to the next part of our efficiency evaluation, which concerns comparing TCP latency between CNIs and regular physical network scenarios.

 doi:10.20944/preprints202408.0970.v1



*5.2. TCP Latency Efficiency Comparison with the Performance of Physical Network without CNIs*

Again, starting with the default tuned profile, we can see that CNIs have roughly 65 to 92 ms more latency across the most commonly used package sizes, as can be seen in Figure 10:

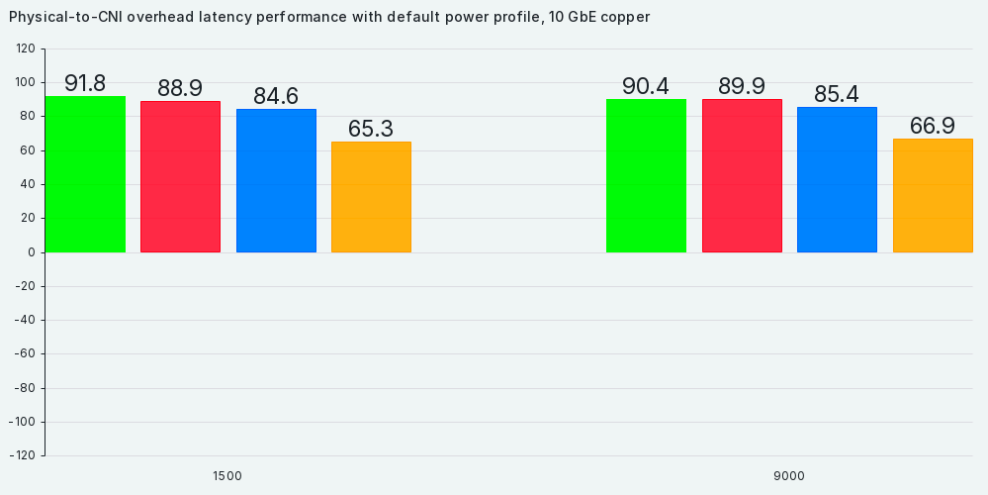


**Figure 10.** Physical to CNI comparison for TCP latency in default tuned performance profile.

Efficiency results get markedly better when we switch to the accelerator-performance tuned profile, as can be seen in Figure 11:

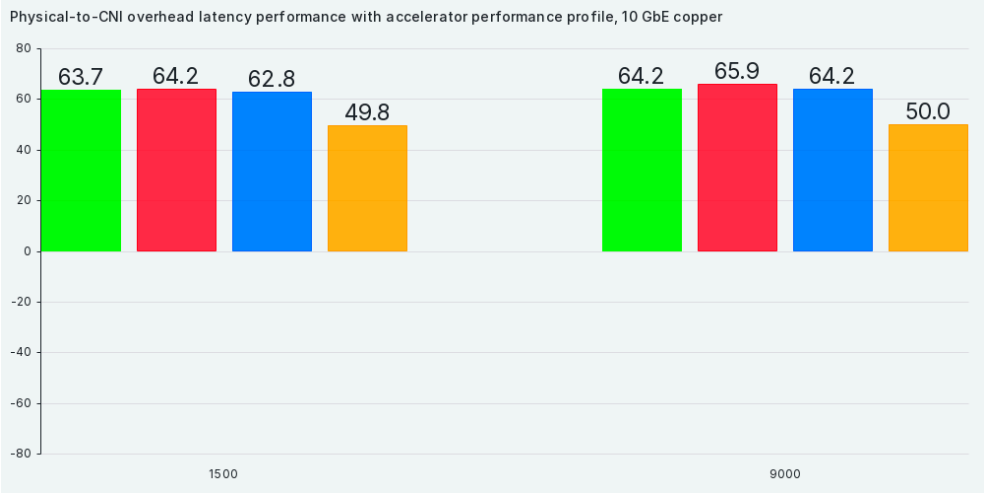


**Figure 11.** Physical to CNI comparison for TCP latency accelerator-performance profile.

Switching to the hpc-compute profile is a step in the wrong direction as latencies increase by roughly 20% across the board, as can be seen in Figure 12:

 doi:10.20944/preprints202408.0970.v1



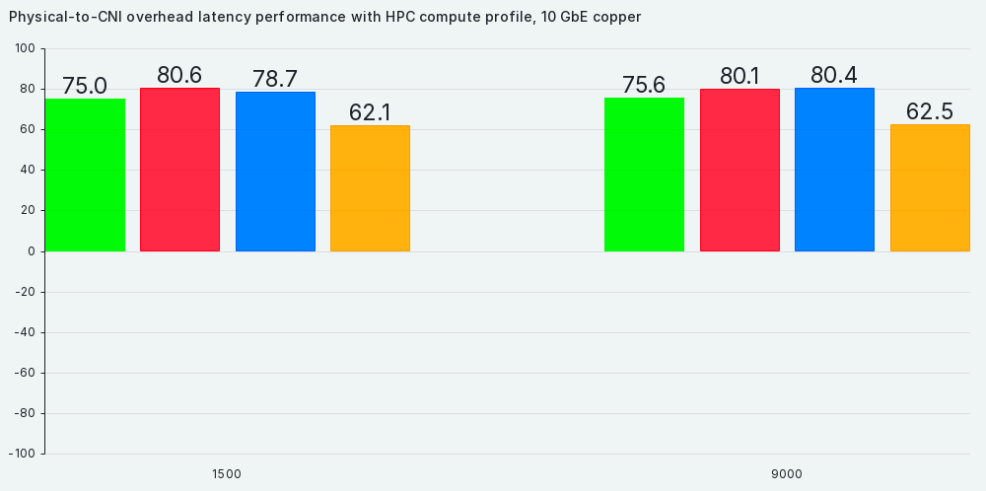


**Figure 12.** Physical to CNI comparison for TCP latency in the hpc-compute performance profile.

If we reconfigure our Kubernetes cluster to use kernel optimizations profile, latency efficiency gets mostly worse, as can be seen in Figure 13:

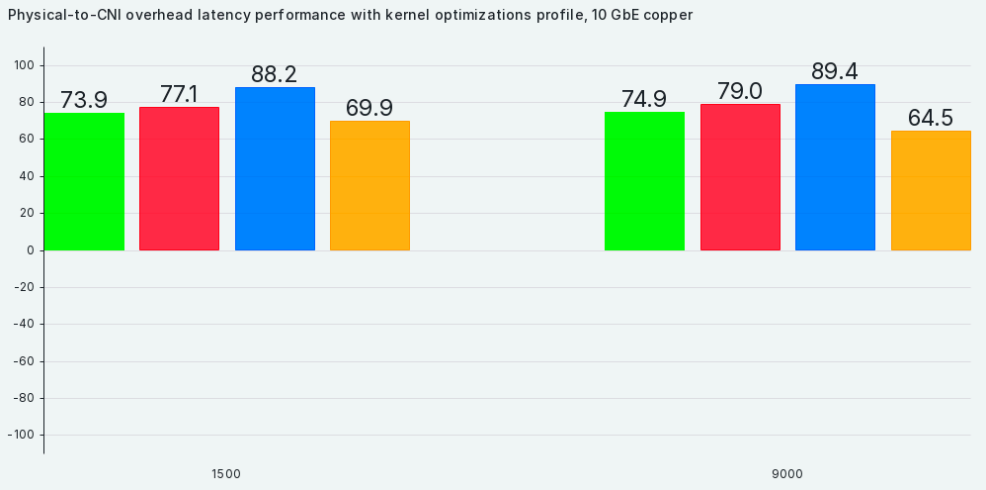


**Figure 13.** Physical to CNI comparison for TCP latency in kernel optimization performance profile.

When we switch to the latency performance profile, efficiency gets much better than all previously tuned profiles. But still, it's quite a bit worse than using the physical network, as can be seen in Figure 14:

 doi:10.20944/preprints202408.0970.v1



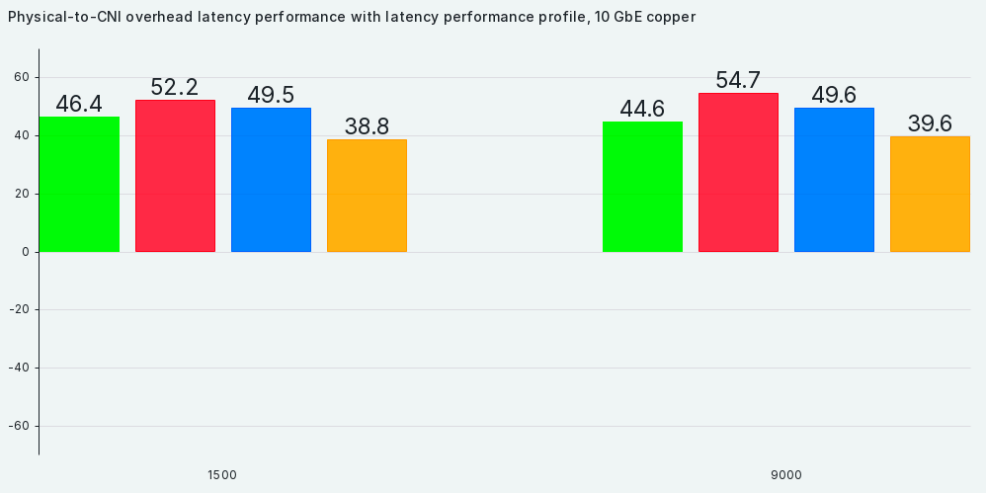


**Figure 14.** Physical to CNI comparison for TCP latency in the latency performance profile.

In the network latency profile, efficiency gets roughly 20% worse yet again when compared to the latency performance profile, as can be seen in Figure 15:

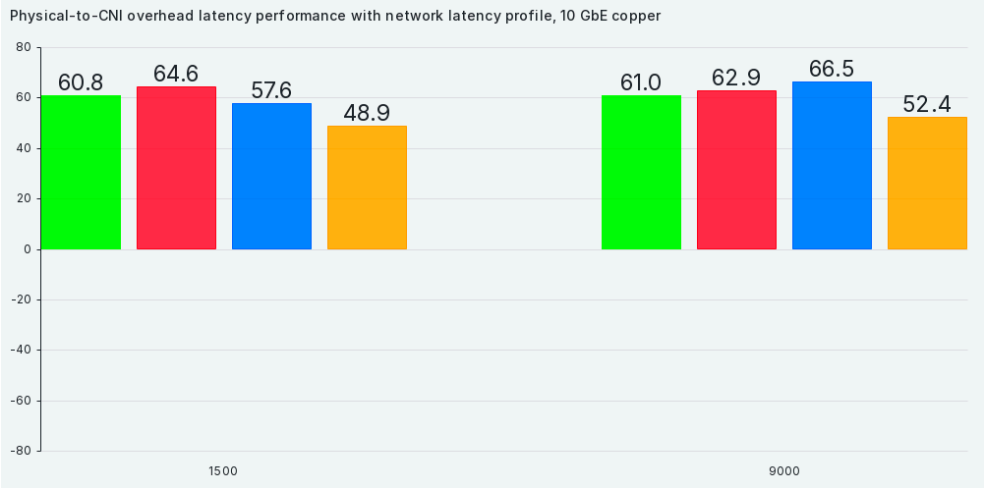


**Figure 15.** Physical to CNI comparison for TCP latency in network latency performance profile.

Switching to the network throughput profile yields marginally better results for most of the tested CNIs, as can be seen in Figure 16:

 doi:10.20944/preprints202408.0970.v1



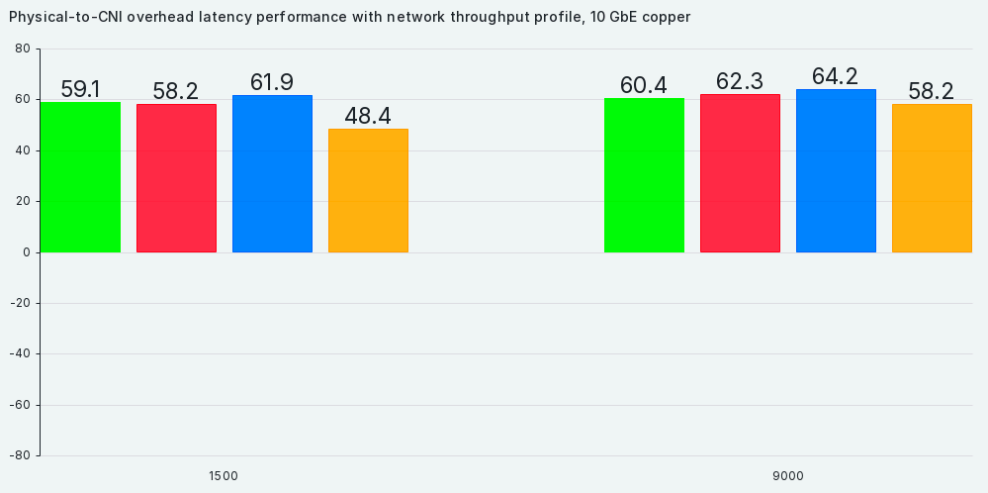


**Figure 16.** Physical to CNI comparison for TCP latency in network throughput performance profile.

Finally, when we reconfigure our Kubernetes cluster to use the NIC optimizations profile, efficiency drops heavily, as can be seen in Figure 17:

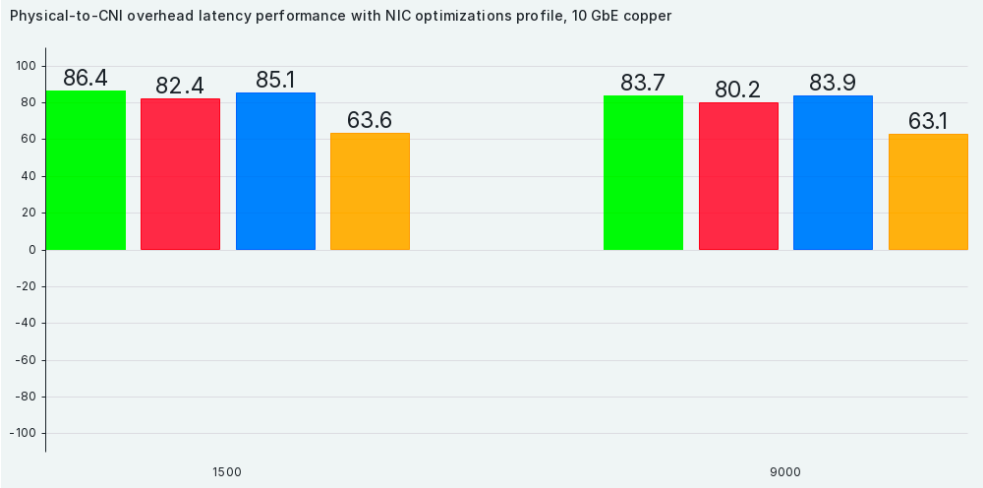


**Figure 17.** Physical to CNI comparison for TCP latency in the nic-optimization performance profile.

The latency efficiency of Kubernetes CNIs, compared to physical network latency, is even more varied than bandwidth. This is partially to be expected due to the very involved nature of the TCP protocol, which loses much efficiency because of its 3-way handshakes and other built-in mechanisms. Regarding TCP latency, Flannel is the clear winner in all the tested TCP scenarios.

With that in mind, we also extensively evaluated UDP bandwidth and latency. The only marked difference in our methodology for UDP is that we did a smaller number of runs (from 5 to 2), as we did extensive prior testing and concluded that it makes negligible difference to the overall evaluation scores.

### *5.3. UDP Bandwidth Efficiency Comparison with the Performance of Physical Network without CNIs*

The following results concern the UDP bandwidth efficiency comparison between CNIs and non-CNI scenarios. When dealing with the default performance profile, the drop in performance as we move to larger packages grows to more than 5 Gbit/s, as we can see in Figure 18:

 doi:10.20944/preprints202408.0970.v1



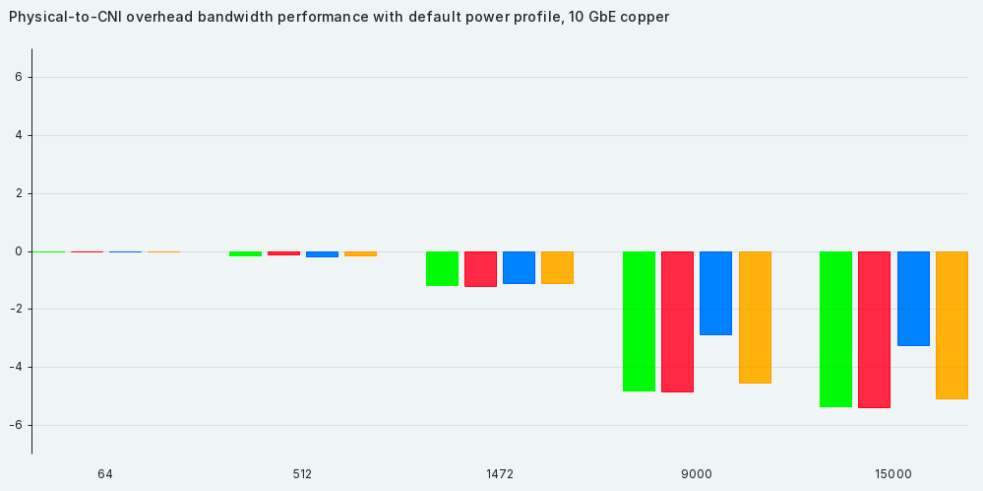


**Figure 18.** Physical to CNI comparison for UDP bandwidth in default tuned performance profile.

Switching our tuned profile to the accelerator-performance profile gets us marginally better results but still well below what we'd consider to be good for larger package sizes, as can be seen in Figure 19:

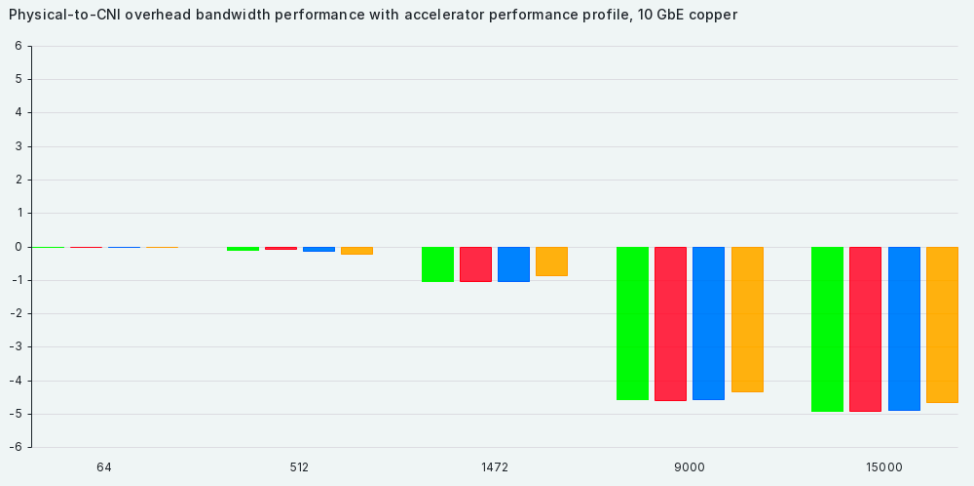


**Figure 19.** Physical to CNI comparison for UDP bandwidth accelerator-performance profile.

In the hpc-compute profile, results are a percent or two better, as can be seen in Picture 20:

 doi:10.20944/preprints202408.0970.v1



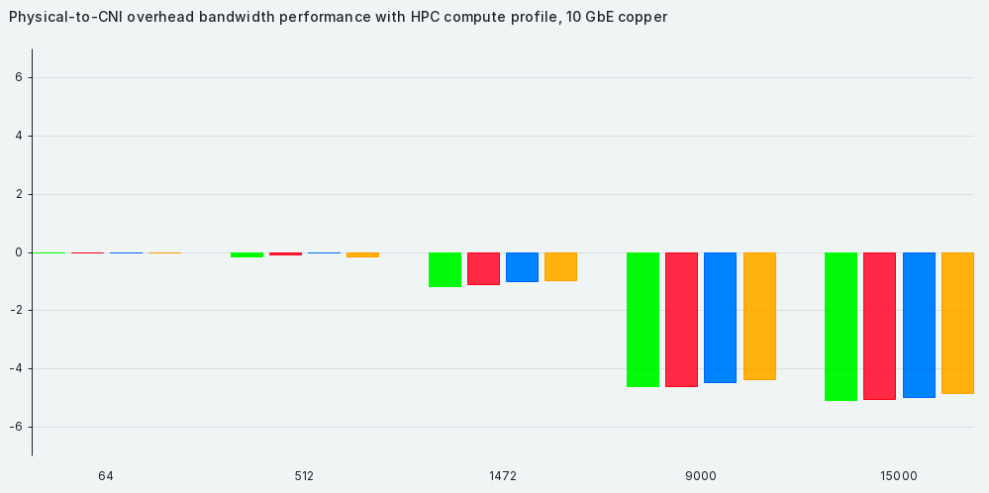


**Figure 20.** Physical to CNI comparison for UDP bandwidth in the hpc-compute performance profile.

Switching to the kernel optimizations profile gets us nowhere, and at least one CNI has a significant performance improvement (Calico), as can be seen in Figure 21:

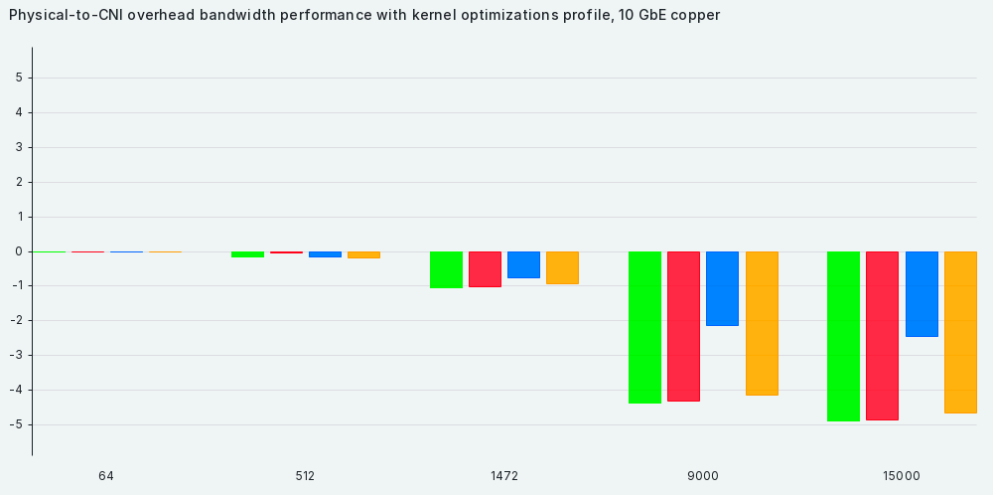


**Figure 21.** Physical to CNI comparison for UDP bandwidth in kernel optimization performance profile.

When we switched our Kubernetes cluster to a latency performance profile, the results got back to roughly 50% efficiency across the board for the larger package sizes, as can be seen in Figure 22:

 doi:10.20944/preprints202408.0970.v1



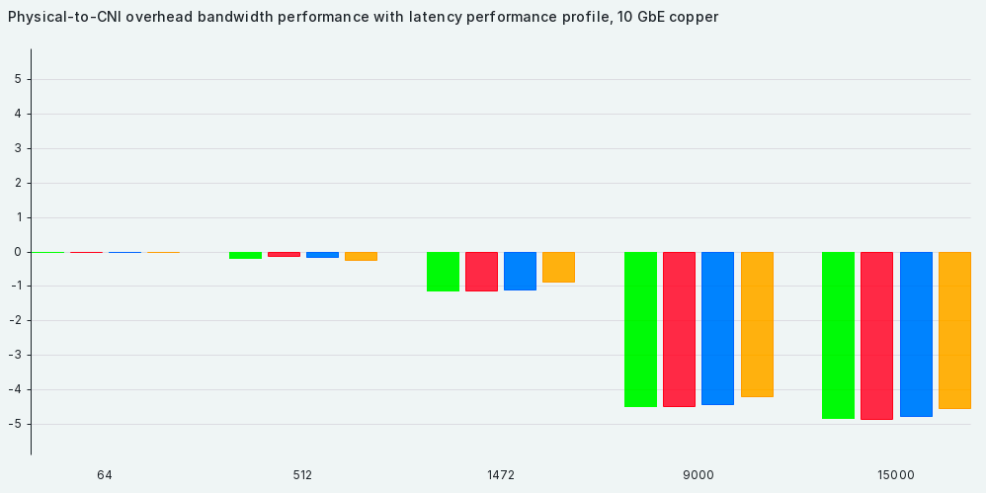


**Figure 22.** Physical to CNI comparison for UDP bandwidth in the latency performance profile.

In the network latency profile, results are almost the same as with the latency performance profile, as can be seen in Figure 23:

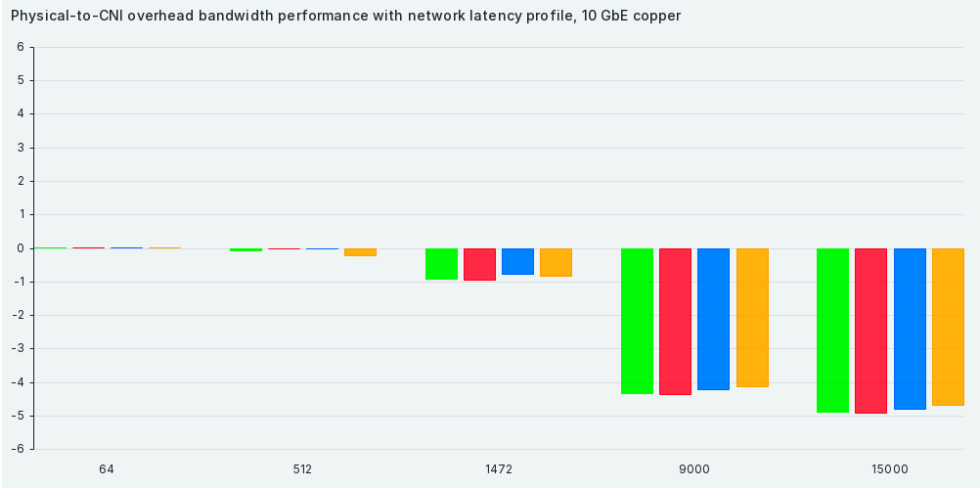


**Figure 23.** Physical to CNI comparison for UDP bandwidth in network latency performance profile.

When we switch to the network throughput profile, results are almost the same, as can be seen in Figure 24:

 doi:10.20944/preprints202408.0970.v1



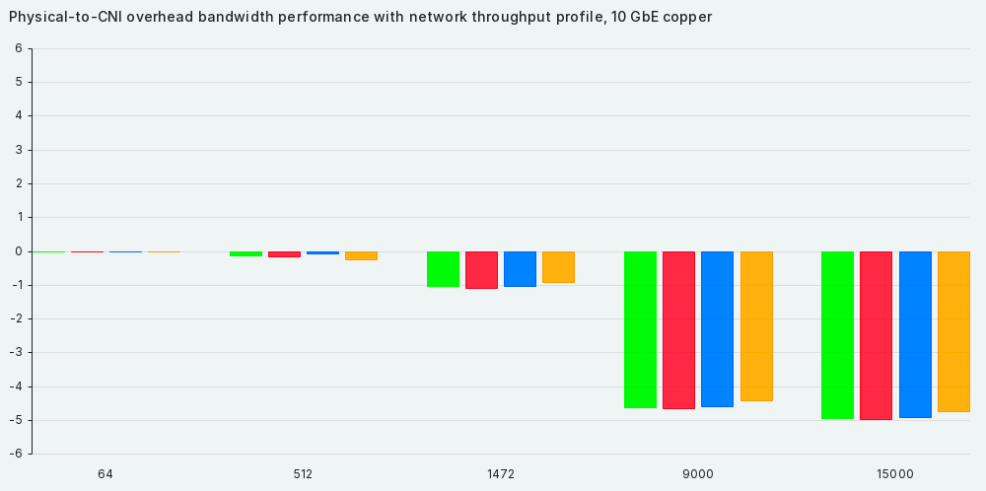


**Figure 24.** Physical to CNI comparison for UDP bandwidth in network throughput performance profile.

In the nic optimizations profile, performance drops for most of the CNIs, except for Antrea, which suddenly jumps in performance by almost 20% when compared to previous tuned profiles, as can be seen in Figure 25:

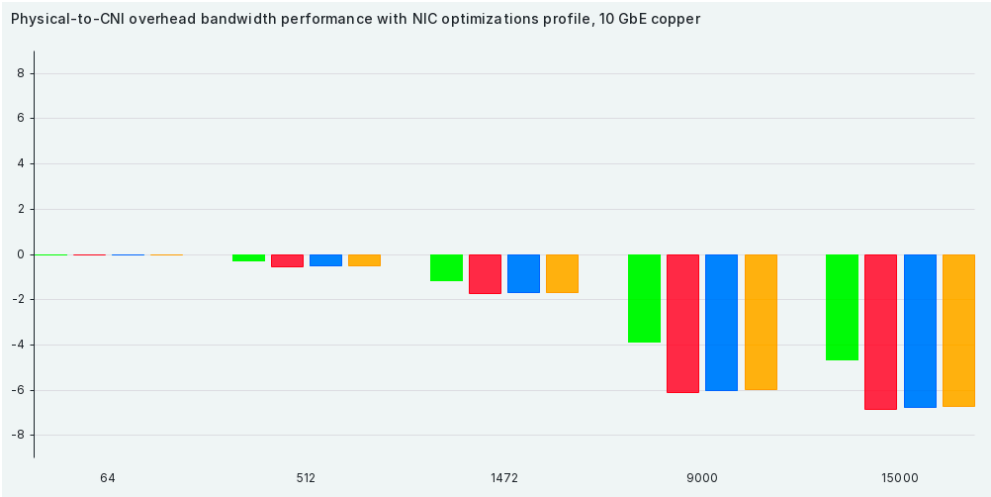


**Figure 25.** Physical to CNI comparison for UDP bandwidth in the nic-optimization performance profile.

Considering all these results, there's only one conclusion—when we use standard-size packages (1472 and 9000), the bandwidth performance drop is significant - for the most common package size (1472), the performance drop is around 20%, while at 9000, the performance drop is almost universally around 50%. We can only imagine what this would mean when using HPC for many streaming-type data (for example, video analysis). Regarding UDP bandwidth, with a couple of exceptions, Antrea seems to be the best option.

### *5.4. UDP Latency Efficiency Comparison with the Performance of Physical Network without CNIs*

The following results concern the UDP latency efficiency comparison between CNIs and non-CNI scenarios. When dealing with the default performance profile, the increase in latency is very noticeable, as can be seen in Figure 26:

 doi:10.20944/preprints202408.0970.v1



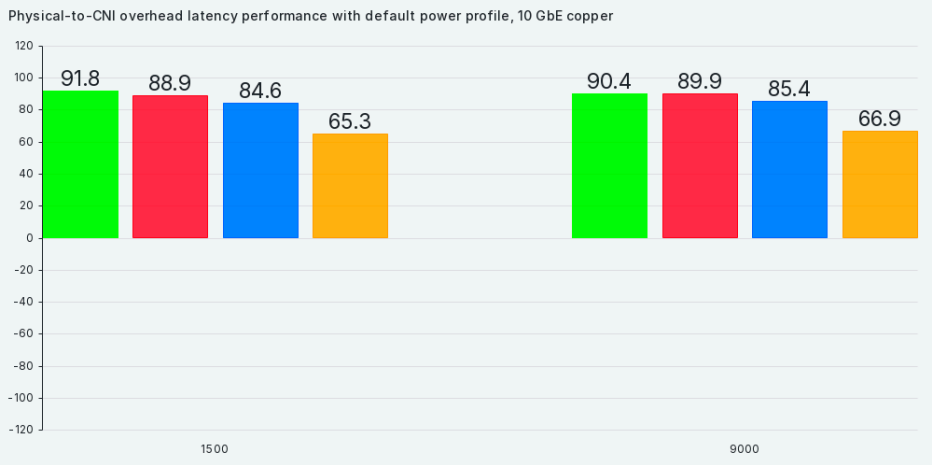


**Figure 26.** Physical to CNI comparison for UDP latency in default tuned performance profile.

Switching to the accelerator performance profile, latencies drop 15-45%, which is encouraging, considering the simplicity of the UDP protocol. We can see the results in Figure 27:

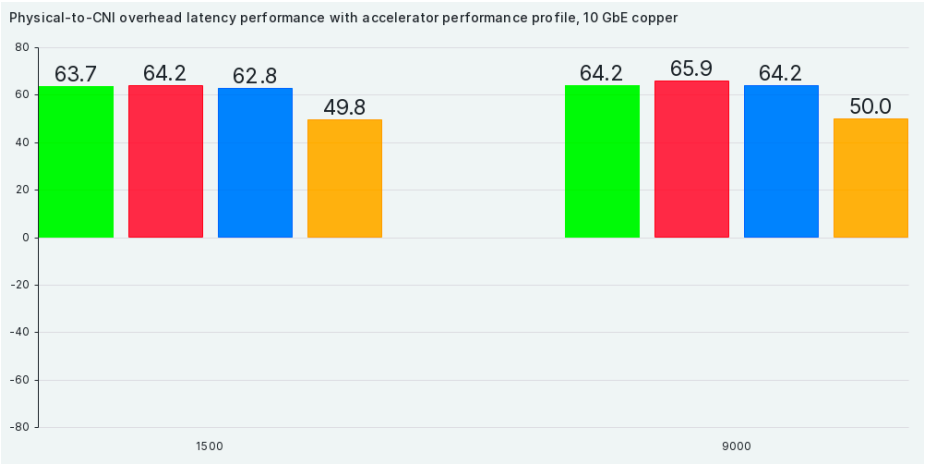


**Figure 27.** Physical to CNI comparison for UDP latency accelerator-performance profile.

The change to the hpc-compute profile does nothing good for our evaluation, as the latency increases by 15-20% across the board, as can be seen in Figure 28:

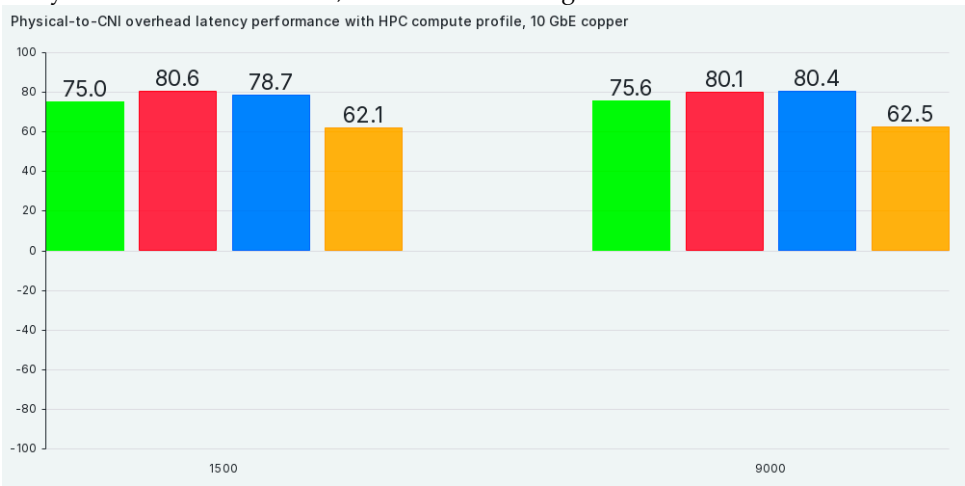


**Figure 28.** Physical to CNI comparison for UDP latency in the hpc-compute performance profile.

 doi:10.20944/preprints202408.0970.v1



In the kernel optimizations profile, latency remains almost the same, as we can see in Figure 29:

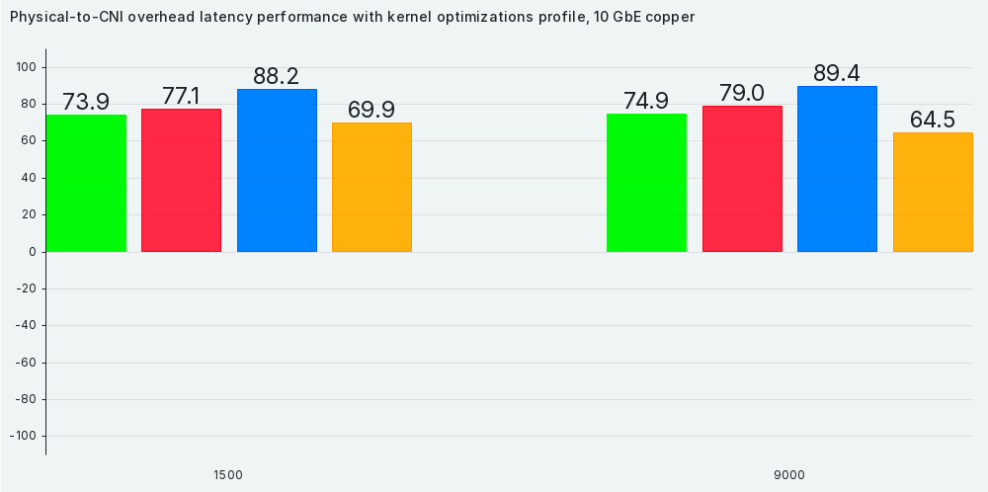


**Figure 29.** Physical to CNI comparison for UDP latency in kernel optimization performance profile.

The next profile on our list is the latency performance profile, which drops the latency significantly to the point of being the best, as can be seen in Figure 30:

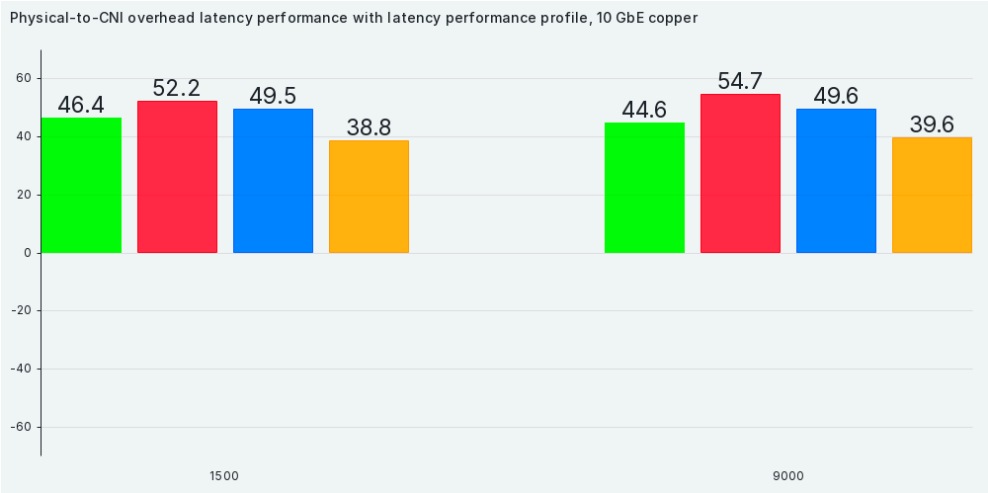


**Figure 30.** Physical to CNI comparison for UDP latency in the latency performance profile.

In the network latency profile, latency increases to the accelerator performance level, as can be seen in Figure 31:

 doi:10.20944/preprints202408.0970.v1



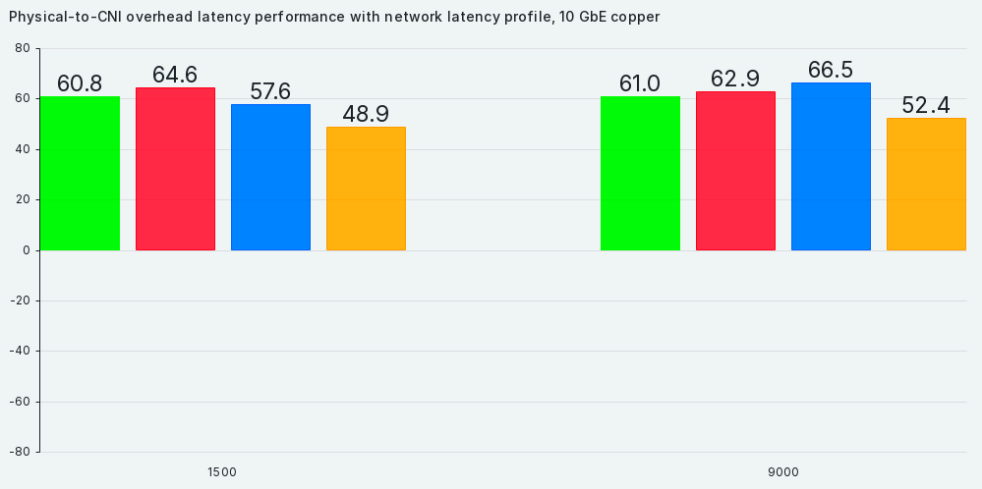


**Figure 31.** Physical to CNI comparison for UDP latency in network latency performance profile.

In the network throughput profile, latency remains broadly the same as in the network latency profile, as can be seen in Figure 32:

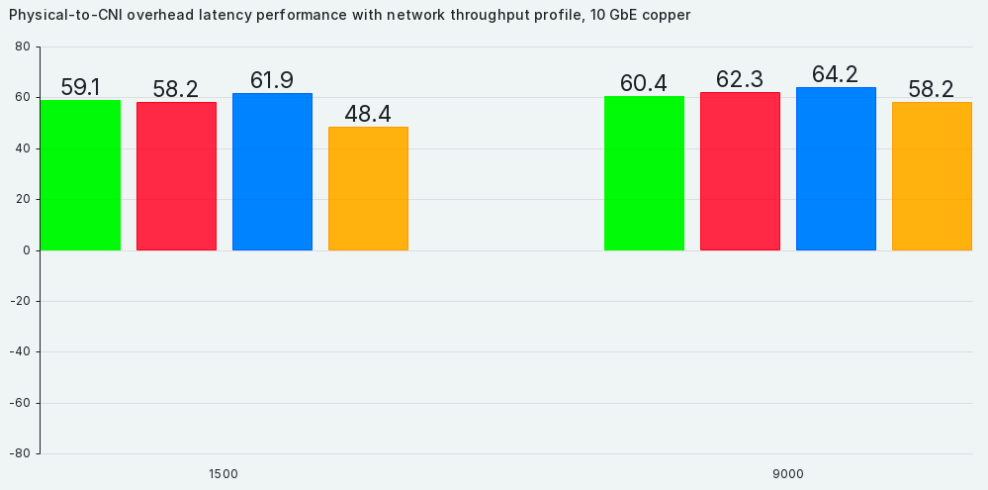


**Figure 32.** Physical to CNI comparison for UDP latency in network throughput performance profile.

Switching to the nic-optimizations profile, latency almost increases to the default profile level, as we can see in Figure 33:

 doi:10.20944/preprints202408.0970.v1



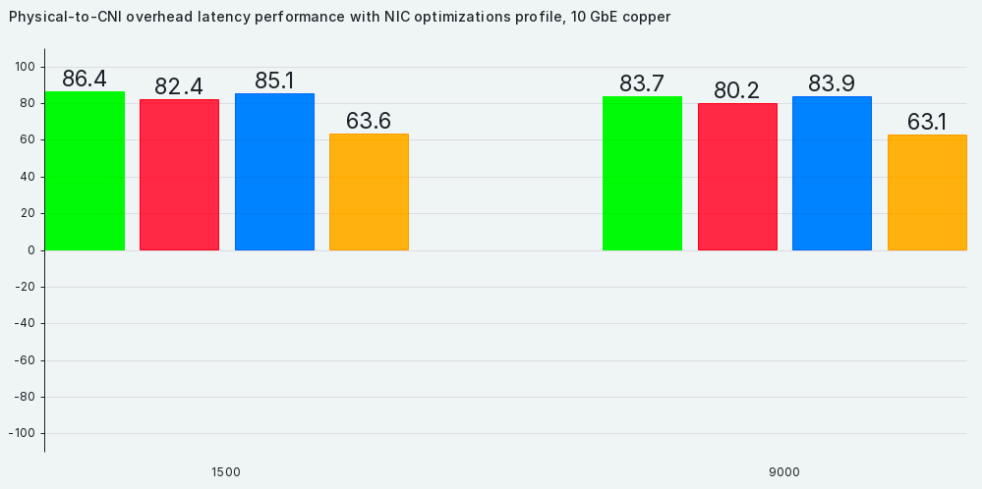


**Figure 33.** Physical to CNI comparison for UDP latency in the nic-optimization performance profile.

Our UDP latency efficiency evaluation suggests that CNIs suffer from a significant latency increase across the board. It almost universally doesn't matter which tuned profile we select, which package size, and which CNI plugin. However, Flannel seems to be the clear performance leader when discussing UDP latency, often by a large margin. However, that still points to the overall problem: CNIs add significant overhead when used for UDP traffic.

## 6. Discussion

Overall, bandwidth evaluations look very good in package sizes up to 1472 bytes and shockingly bad in larger package sizes. For latency, all results are sub-optimal. We can also call these results counter-intuitive, as it's usually the reverse – networks usually perform better when dealing with larger package sizes. From the efficiency perspective, compared to a physical network scenario that doesn't use CNIs, we're still way off in performance and latency. And it's not only that – we would expect UDP results to be markedly better than TCP across the board, which wasn't necessarily the case. We can only attribute this to overall CNI inefficiency, which means there's much room for improvement – both in how CNIs handle Kubernetes networking and how that gets scheduled on the OS side and network interfaces. We'd be curious to see what these results would look like if we used FPGAs or other offloaded network adapters as the underlying physical interfaces.

Looking at the actual results, 40ms of added latency in the best case and 90ms in the worst case is a huge performance problem for any distributed application scenario, especially HPC/AI applications. The performance drop would be significant if we ran a set of HPC workloads on thousands of containers across hundreds of HPC nodes with these latencies. Bandwidth-wise, CNIs have the same problem. Our performance evaluations could not reach 50% link saturation in TCP or UDP scenarios with larger package sizes, which is very surprising, especially for UDP. And we didn't try to cut corners with cheap network cards, either – we're running Intel and Mellanox enterprise-grade network cards across these evaluations. Furthermore, we had no issues reaching close to 100% saturation when we performed baselining on the physical network level or with smaller package sizes, so these results can only be attributed to CNI performance. This also shows that there's much research to be done in this area as improvements need to be made everywhere – CNI efficiency, data center design, etc. We will discuss some potential research avenues in the next section.

## 7. Future Work

We see much potential for future research in terms of scalability across even more recent Ethernet standards (25 Gbit/s, 50 Gbit/s, 100 Gbit/s, 200 Gbit/s, etc.), as it's difficult to predict how bandwidth increases will influence pure CNI performance. But even with these newer networking standards, we need to see how an increase in pure bandwidth on the physical level will help. A brute-

 doi:10.20944/preprints202408.0970.v1



force way of solving problems like network efficiency issues isn't the way to go; there needs to be much research done on improving the architecture and efficiency of these plugins.

Future research must be done on offload technologies for various network functions, especially considering the increase in bandwidth and the fact that CNIs have much overhead. FPGAs and network adapters using Intel's DPDK are good candidates for this research. Furthermore, network adapters using offloading techniques for TCP (like TOE, TCP Offload Engine) and UDP (like UOE, UDP Offload Engine) might help alleviate a considerable portion of the measured overheads presented by this paper. Misconfiguration of any of these technologies will significantly impact CNIs' performance in real-life scenarios.

As mentioned in Section 3, one of the most fundamental challenges created by the Kubernetes-based HPC environments is that CNIs have much overhead and need to be incorporated into the initial design. How this challenge is solved is a substantial area of future research. From what we learned in this paper, if we focus on one area of the design, which points to the fact that large-scale HPC environments for Kubernetes will have to use, for example, DPDK-enabled adapters with offload engines, or FPGAs – fundamentally changes how we plan and design our HPC environments. This design change also comes with a significant increase in cost.

## 8. Conclusions

This paper evaluates the performance and latency efficiency of Kubernetes CNIs across different power profiles and custom configurations. Our methodology employs Ansible for automated and orchestrated testing to manage various CNIs in Kubernetes clusters, ensuring consistent and repeatable performance assessments. The most used CNIs, such as Antrea, Flannel, Cilium, and Calico, are examined, highlighting their strengths and weaknesses in handling different workload types, including bandwidth and latency-sensitive workloads. The paper underscores the importance of selecting the appropriate CNI based on specific network requirements and workloads to optimize Kubernetes environments. Various tuned performance profiles are tested, revealing significant differences among CNIs, especially in bandwidth and latency metrics.

The TCP and UDP bandwidth and latency testing results reveal significant performance disparities among CNIs and tuned performance profiles. TCP bandwidth efficiency drops significantly for larger packet sizes, with Antrea showing the best performance overall. Flannel performs best for TCP latency but still lags behind physical networks. UDP testing shows a similar trend, with Antrea excelling in bandwidth efficiency while latency performance varies across profiles. Also, Calico has a very pronounced sensitivity to kernel-level settings, as proved in our TCP and UDP bandwidth tests. Calico's efficiency increased dramatically when used with our custom kernel-tuned parameters. These results highlight the importance of selecting and tuning CNIs based on specific workload requirements to optimize Kubernetes network performance.

This research emphasizes the necessity for automated methodologies to efficiently evaluate CNIs, particularly in HPC scenarios, to achieve optimal network configuration and performance. Manually performing these testing procedures and dealing with the human factor afterward is too complicated. Repeatable results are the key here, as our results clearly show.

**Author Contributions:** Conceptualization, V.D.; Methodology, V.D.; Validation, M.B. and L.Z.; Formal analysis, M.B. and D.R..; Investigation, M.B., and L.Z.; Resources, V.D.; Writing—original draft, V.D. and M.B.; Writing—review & editing, V.D. and D.R.; Supervision, V.D.; Project administration, V.D. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** The authors declare no conflict of interest.

 doi:10.20944/preprints202408.0970.v1



## References


1. Dakić, V.; Kovač, M.; Slovinac, J. Evolving High-Performance Computing Data Centers with Kubernetes, Performance Analysis, and Dynamic Workload Placement Based on Machine Learning Scheduling. Electronics 2024, 13, 2651. https://doi.org/10.3390/electronics13132651.
2. S. Zhu, C. Zhou, and Y. Wang, "Highly Efficient Multicast over Surface Wave in Hybrid Wireless-Optical On-Chip Networks for IoT HPC," Wireless Communications and Mobile Computing, vol. 2022, 2022, doi: 10.1155/2022/3882894.
3. Y. Shang, X. Guo, B. Guo, H. Wang, J. Xiao, and S. Huang, "Automatically Reconfigurable Optical Network for HPC System Based on Deep Reinforcement Learning," 2022 Asia Communications and Photonics Conference (ACP), pp. 1163-1167, 2022, doi: 10.1109/ACP55869.2022.10088485.
4. H. Veerla, V. Chindukuri, Y. J. Mohammed, V. Palakonda, and R. R. Chintala, "HPC Driven Innovations in Network Management for Greater Efficiency and Productivity," 2023 5th International Conference on Inventive Research in Computing Applications (ICIRCA), pp. 1047-1053, 2023, doi: 10.1109/ICIRCA57980.2023.10220775.
5. M. Y. Teh, Z. Wu, M. Glick, S. Rumley, M. Ghobadi, and K. Bergman, "Performance Trade-offs in Reconfigurable Networks for HPC," Journal of Optical Communications and Networking, vol. 14, pp. 454-468, 2022, doi: 10.1364/jocn.451760.
6. P. Y. Su, D. Ho, J. Pu, and Y. P. Wang, "Chiplets Integrated Solution with FO-EB Package in HPC and Networking Application," 2022 IEEE 72nd Electronic Components and Technology Conference (ECTC), pp. 2135-2140, 2022, doi: 10.1109/ectc51906.2022.00337.
7. P. Haghi, A. Guo, T. Geng, A. Skjellum, and M. C. Herbordt, "Workload Imbalance in HPC Applications: Effect on Performance of In-Network Processing," 2021 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1-8, 2021, doi: 10.1109/HPEC49654.2021.9622847.
8. S. Christgau, D. Everingham, F. Mikolajczak, N. Schelten, B. Schnor, M. Schroetter, B. Stabernack, and F. Steinert, "Enabling Communication with FPGA-based Network-attached Accelerators for HPC Workloads," Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, 2023, doi: 10.1145/3624062.3624540.
9. S. Varrette, H. Cartiaux, T. Valette, and A. Olloh, "Aggregating and Consolidating Two High Performant Network Topologies: The ULHPC Experience," Practice and Experience in Advanced Research Computing, 2022, doi: 10.1145/3491418.3535159.
10. Y. Zhang, B. Aksar, O. Aaziz, B. Schwaller, J. Brandt, V. Leung, M. Egele, and A. Coskun, "Using Monitoring Data to Improve HPC Performance via Network-Data-Driven Allocation," 2021 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1-7, 2021, doi: 10.1109/HPEC49654.2021.9622783.
11. T. Islam and C. Phelps, "HPC@SCALE: A Hands-on Approach for Training Next-Gen HPC Software Architects," 2021 IEEE 28th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW), pp. 29-34, 2021, doi: 10.1109/HiPCW54834.2021.00011.
12. F. Yan, C. Yuan, C. Li, and X. Deng, "FOSquare: A Novel Optical HPC Interconnect Network Architecture Based on Fast Optical Switches with Distributed Optical Flow Control," Photonics, vol. 8, no. 1, 2021, doi: 10.3390/photonics8010011.
13. J. He, C. Wu, H. Li, Y. Guo, and T. Li, "Editorial for the Special Issue on Reliability and Power Efficiency for HPC," CCF Transactions on High Performance Computing, vol. 3, pp. 1-3, 2021, doi: 10.1007/s42514-020-00062-5.
14. S. Bhowmik, N. Jain, X. Yuan, and A. Bhatele, "A Simulation Study of Hardware Parameters for Future GPU-based HPC Platforms," 2021 IEEE International Performance, Computing, and Communications Conference (IPCCC), pp. 1-10, 2021, doi: 10.1109/IPCCC51483.2021.9679359.
15. M. Chadha, N. Krueger, J. John, A. Jindal, M. Gerndt, and S. Benedict, "Exploring the Use of WebAssembly in HPC," Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, 2023, doi: 10.1145/3572848.3577436.
16. Y. Wang, H. Liu, and T. Zhao, "Hybrid-TF: A New Hybrid Interconnection Network Topology for High Performance Computing," 2021 13th International Conference on Communication Software and Networks (ICCSN), pp. 62-68, 2021, doi: 10.1109/ICCSN52437.2021.9463619.
17. E. Suvorova, "An Approach to Designing Heterogeneous Networks for High Performance Computing Systems Based on a Unified Reconfigurable Router Core," 2022 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF), pp. 1-11, 2022, doi: 10.1109/weconf55058.2022.9803490.
18. P. Maniotis, L. Schares, D. Kuchta, and B. Karaçali, "Toward Higher-Radix Switches with Co-Packaged Optics for Improved Network Locality in Data Center and HPC Networks [Invited]," Journal of Optical Communications and Networking, vol. 14, pp. C1-C10, 2022, doi: 10.1364/jocn.451449.
19. H. Lin, X.-Z. Hsu, L.-Y. Wang, Y.-H. Hsu, and Y. Wang, "High Thermal Assembly Solution for Large FOEB Chiplets Integrated Package in HPC Application," 2022 23rd International Conference on Electronic Packaging Technology (ICEPT), pp. 1-4, 2022, doi: 10.1109/ICEPT56209.2022.9872620.

20. N. Contini, B. Ramesh, K. Kandadi Suresh, T. Tran, B. Michalowicz, M. Abduljabbar, H. Subramoni, and D. Panda, "Enabling Reconfigurable HPC through MPI-based Inter-FPGA Communication," Proceedings of the 37th International Conference on Supercomputing, 2023, doi: 10.1145/3577193.3593720.
21. J. Kim, M. Cafaro, J. Chou, and A. Sim, "SNTA'22: The 5th Workshop on Systems and Network Telemetry and Analytics," Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing, 2022, doi: 10.1145/3502181.3535108.
22. N. Kapočius, "Overview of Kubernetes CNI Plugins Performance," Mokslas - Lietuvos Ateitis, vol. 12, pp. 1-5, 2020, doi: 10.3846/mla.2020.11454.
23. N. Kapočius, "Performance Studies of Kubernetes Network Solutions," 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), pp. 1-6, 2020, doi: 10.1109/eStream50540.2020.9108894.
24. S. Novianti and A. Basuki, "The Performance Analysis of Container Networking Interface Plugins in Kubernetes," Proceedings of the 6th International Conference on Sustainable Information Engineering and Technology, 2021, doi: 10.1145/3479645.3479700.
25. S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Assessing Container Network Interface Plugins: Functionality, Performance, and Scalability," IEEE Transactions on Network and Service Management, vol. 18, pp. 656-671, 2021, doi: 10.1109/TNSM.2020.3047545.
26. M. M. Iordache-Sica, T. Kraiser, and O. Komolafe, "Seamless Hardware-Accelerated Kubernetes Networking," Proceedings of the 2nd ACM SIGCOMM Workshop on Future of Internet Routing & Addressing, 2023, doi: 10.1145/3607504.3609292.
27. T. P. Nagendra and R. Hemavathy, "Unlocking Kubernetes Networking Efficiency: Exploring Data Processing Units for Offloading and Enhancing Container Network Interfaces," 2023 4th IEEE Global Conference for Advancement in Technology (GCAT), pp. 1-7, 2023, doi: 10.1109/GCAT59970.2023.10353542.
28. S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Understanding Container Network Interface Plugins: Design Considerations and Performance," 2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pp. 1-6, 2020, doi: 10.1109/LANMAN49260.2020.9153266.
29. Z. Kang, K. An, A. Gokhale, and P. N. Pazandak, "A Comprehensive Performance Evaluation of Different Kubernetes CNI Plugins for Edge-based and Containerized Publish/Subscribe Applications," 2021 IEEE International Conference on Cloud Engineering (IC2E), pp. 31-42, 2021, doi: 10.1109/ic2e52221.2021.00017.
30. R. Kumar and M. Trivedi, "Networking Analysis and Performance Comparison of Kubernetes CNI Plugins," Benchmark results of Kubernetes network plugins (CNI) over 10 Gbit/s network, 2020, doi: 10.1007/978-981-15-4409-5_9.
31. A. Ayub, M. Ishaq, and M. Munir, "Enhancement in Multus CNI for DPDK Applications in the Cloud Native Environment," 2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), pp. 66-68, 2023, doi: 10.1109/ICIN56760.2023.10073475.
32. A. Garbugli, L. Rosa, A. Bujari, and L. Foschini, "KuberneTSN: a Deterministic Overlay Network for Time-Sensitive Containerized Environments," ArXiv, vol. abs/2302.08398, 2023, doi: 10.48550/arXiv.2302.08398.
33. Y. Park, H. Yang, and Y. Kim, "Performance Analysis of CNI (Container Networking Interface) based Container Network," 2018 International Conference on Information and Communication Technology Convergence (ICTC), pp. 248-250, 2018, doi: 10.1109/ICTC.2018.8539382.
34. F. Lombardo, S. Salsano, A. Abdelsalam, D. Bernier, and C. Filsfils, "Extending Kubernetes Networking to make use of Segment Routing over IPv6 (SRv6)," ArXiv, vol. abs/2301.01178, 2023, doi: 10.48550/arXiv.2301.01178.
35. L. Larsson, W. Tarneberg, C. Klein, E. Elmroth, and M. Kihl, "Impact of etcd deployment on Kubernetes, Istio, and application performance," Software: Practice and Experience, vol. 50, pp. 1986-2007, 2020, doi: 10.1002/spe.2885.
36. Y. Ou, "Optimizing Container CNI Metrics for Improved Service Level Indicators and Objectives in Container Networking," Science and Technology of Engineering, Chemistry and Environmental Protection, 2023, doi: 10.61173/bfvps044.
37. N. Jain, V. Mohan, A. Singhai, D. Chatterjee, and D. Daly, "Kubernetes Load-balancing and related network functions using P4," Proceedings of the Symposium on Architectures for Networking and Communications Systems, 2021, doi: 10.1145/3493425.3502768.
38. D. Milroy et al., "One Step Closer to Converged Computing: Achieving Scalability with Cloud-Native HPC," 2022 IEEE/ACM 4th International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), pp. 57-70, 2022, doi: 10.1109/CANOPIE-HPC56864.2022.00011.
39. K.-H. Kim, D. Kim, and Y.-h. Kim, "Open-Cloud Computing Platform Design based on Virtually Dedicated Network and Container Interface," 2020, doi: 10.35940/ijitee.e2176.039520.

40. V. Medel Gracia, R. Tolosana-Calasanz, J. A. Banares, U. Arronategui, and O. Rana, "Characterising resource management performance in Kubernetes," Comput. Electr. Eng., vol. 68, pp. 286-297, 2018, doi: 10.1016/j.compeleceng.2018.03.041.
41. D. T. Nguyen et al., "Enhancing CNF performance for 5G core network using SR-IOV in Kubernetes," 2022 24th International Conference on Advanced Communication Technology (ICACT), pp. 501-506, 2022, doi: 10.23919/ICACT53585.2022.9728817.
42. Budigiri, G.; Baumann, C.; Muhlberg, J.T.; Truyen, E.; Joosen, W. Network Policies in Kubernetes: Performance Evaluation and Security Analysis. 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit) 2021. https://doi.org/10.1109/EuCNC/6GSummit51104.2021.9482526.
43. Jain, N.; Mohan, V.K.C.; Singhai, A.; Chatterjee, D.; Daly, D. Kubernetes Load-Balancing and Related Network Functions Using P4. Proceedings of the Symposium on Architectures for Networking and Communications Systems 2021. https://doi.org/10.1145/3493425.3502768.
44. Zeng, H.; Wang, B.; Deng, W.; Zhang, W. Measurement and Evaluation for Docker Container Networking. 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC) 2017. https://doi.org/10.1109/CyberC.2017.78.
45. Nagendra, T.P.; Hemavathy, R. Unlocking Kubernetes Networking Efficiency: Exploring Data Processing Units for Offloading and Enhancing Container Network Interfaces. 2023 4th IEEE Global Conference for Advancement in Technology (GCAT) 2023. https://doi.org/10.1109/GCAT59970.2023.10353542.
46. Brewer, E.A. Kubernetes and the Path to Cloud Native. Proceedings of the Sixth ACM Symposium on Cloud Computing 2015. https://doi.org/10.1145/2806777.2809955.
47. Haja, D.; Szalay, M.; Sonkoly, B.; Pongracz, G.; Toka, L. Sharpening Kubernetes for the Edge. Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos 2019. https://doi.org/10.1145/3342280.3342335.
48. Haragi L, D.; S, M.; B, Prof.S. Infrastructure Optimization in Kubernetes Cluster. JUSST 2021, 23, 546–555. https://doi.org/10.51201/JUSST/21/05292.
49. Telenyk, S.; Sopov, O.; Zharikov, E.; Nowakowski, G. A Comparison of Kubernetes and Kubernetes-Compatible Platforms. 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS) 2021. https://doi.org/10.1109/IDAACS53288.2021.9660392.
50. Brewer, E. Kubernetes and the New Cloud. Proceedings of the 2018 International Conference on Management of Data 2018. https://doi.org/10.1145/3183713.3183725.
51. Kim, E.; Lee, K.; Yoo, C. On the Resource Management of Kubernetes. 2021 International Conference on Information Networking (ICOIN) 2021. https://doi.org/10.1109/ICOIN50884.2021.9333977.
52. Poniszewska-Marańda, A.; Czechowska, E. Kubernetes Cluster for Automating Software Production Environment. Sensors 2021, 21, 1910. https://doi.org/10.3390/s21051910.
53. Perera, H.C.S.; De Silva, T.S.D.; Wasala, W.M.D.C.; Rajapakshe, R.M.P.R.L.; Kodagoda, N.; Samaratunge, U.S.S.; Jayanandana, H.H.N.C. Database Scaling on Kubernetes. 2021 3rd International Conference on Advancements in Computing (ICAC) 2021. https://doi.org/10.1109/ICAC54203.2021.9671185.
54. Dakic, V.; Redzepagic, J.; Basic, M.; Zgrablic, L. Methodology For Automating And Orchestrating Performance Evaluation Of Kubernetes Container Network Interfaces. Preprints 2024, 2024071592. https://doi.org/10.20944/preprints202407.1592.v1.