IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 24 January 2025, accepted 17 February 2025, date of publication 20 February 2025, date of current version 27 February 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3543841

RESEARCH ARTICLE

Exploring Security Enhancements in Kubernetes CNI: A Deep Dive Into Network Policies

BOM KIM[1], JINWOO KIM[2], AND SEUNGSOO LEE[1]
[1]Incheon National University, Incheon 22012, Republic of Korea
[2]Kwangwoon University, Seoul 01897, Republic of Korea

Corresponding authors: Jinwoo Kim (jinwookim@kw.ac.kr) and Seungsoo Lee (seungsoo@inu.ac.kr)

This work was supported by Incheon National University (International Cooperative) Research Grant, in 2024.

ABSTRACT With the explosive growth of Kubernetes adoption, Container Network Interfaces (CNIs) have become critical components for configuring and securing container networks, but a comprehensive analysis of their security capabilities and performance impact is noticeably lacking. Our study conducts a comprehensive security analysis of the major CNI plugins (Cilium, Calico, WeaveNet, Kube-router, and Antrea) in cloud-native environments with Kubernetes through extensive evaluation of Layer 3/4 policy processing, policy complexity scaling, pod scalability, and Layer 7 policy processing. The experimental results show that eBPF-based Cilium maintains 8.9K Mbps throughput under complex L3/4 policies, but drops to 94 Mbps with L7 processing, while Antrea achieves 6.6K Mbps at L7 through HTTP filtering, with performance degrading as policy complexity increases. Under high concurrent pod loads, iptables-based CNIs show a 60-70% reduction in throughput, while Cilium maintains performance within 10% of baseline. These results reveal critical trade-offs between architectural choices and security capabilities, and provide practical guidelines for CNI selection based on specific operational and security requirements in cloud-native environments.

INDEX TERMS Container network interface, cloud security, container security, network policy.

I. INTRODUCTION

The rapid adoption of cloud computing and container technologies is fundamentally changing today's IT landscape. According to IDC's 2024 report [1], global public cloud services revenue is expected to exceed $800 billion in 2024. This represents a 20.5% increase from 2023. The growth is particularly strong in the IaaS and PaaS segments, driven by increasing demand for AI and high-performance computing applications. As the cloud market expands, cloud-native computing is becoming increasingly important to enterprises' digital transformation initiatives. Cloud-native architectures emphasize flexibility, scalability, and automation, with microservices, containers, and orchestration tools emerging as foundational technology components [1], [2].

However, this accelerated growth has been accompanied by a concurrent rise in security concerns [3], [4]. As documented in the AlgoSec 2024 State of Network Security Report [5], 97% of respondents cited security and compliance as a top concern when selecting cloud platforms. In particular, the complex network configurations inherent in hybrid and multi-cloud environments have emerged as a significant source of security challenges.

In container environments, traditional network security approaches are inadequate due to three factors: the dynamic scaling of workloads, the ephemeral nature of containers, and the complex communication patterns between microservices. The Container Network Interface (CNI) [6] plays a pivotal role in standardizing and managing network connections between containers, thereby facilitating the integration of a variety of networking solutions into container orchestration platforms such as Kubernetes [7]. However, the default configuration of Kubernetes allows unrestricted communication between all pods without isolation, creating potential security vulnerabilities that CNI cannot adequately address on its own.

Thus, network policies serve as a security mechanism to address such a vulnerability, providing sophisticated capabilities for granular traffic control within container environments. When implemented, these policies enable precise regulation of inter-pod communication patterns,

The associate editor coordinating the review of this manuscript and approving it for publication was Mehdi Sookhak.

minimizing network exposure vulnerabilities and enforcing the principle of least privilege within microservice architectures. Additionally, the extensive monitoring and logging inherent in network policies facilitates rapid detection and response to potential security breaches, improving overall system resilience. As sophisticated threat vectors such as side-channel attacks, container hijacking, and network sniffers become more prevalent, granular access control and traffic monitoring through CNI and network policies have become critical. However, network policy handling and security characteristics vary among CNIs, and the impact of these differences on security effectiveness has not been systematically analyzed. Therefore, designing an effective security architecture requires careful analysis of each CNI's security features and how they support network policies.

Existing analyses of CNIs have predominantly focused on basic features or performance metrics [8], [9], [10], [11], [12], [13], with only cursory mentions of policy support mechanisms. Even studies that do address security policies tend to focus narrowly on specific L3/L4 technologies such as eBPF and iptables, rather than providing a holistic examination of how CNIs implement and enforce network policies across all layers (L3/L4/L7). Despite emerging research interests in policy management and automation in Kubernetes [14], [15], [16], [17], [18], analytical research on container network security remains fragmented and incomplete. This gap creates an urgent need for a more thorough analysis of how different CNIs manage and process security policies.

To address these limitations, this paper presents a comprehensive analysis of the network policy support and security features of the major CNI plugins used in the Kubernetes environment, specifically Flannel [19], WeaveNet [20], Calico [21], Kube-router [22], Cilium [23], and Antrea [24]. We present a quantitative and qualitative analysis of the relationships between architectural design choices, security capabilities, and performance characteristics for each CNI. Our results show that Kubernetes policies take precedence over CNI policies due to the positioning of the kernel stack, allowing for complementary security enforcement.

In L3/L4 testing, eBPF-based Cilium achieves 8.9K Mbps throughput even with 1,000 policy rules, while OVS-based Antrea shows significant degradation to 1.2K Mbps. In concurrent pod scaling tests, iptables-based CNIs experience 60-70% throughput reduction at 200 pods, while Cilium maintains performance within 10% of baseline. At L7, architectural choices lead to an interesting reversal - Cilium's extensive protocol support leads to significant performance drops (94 Mbps), while Antrea's HTTP filtering maintains 6.6K Mbps throughput. Finally, this paper presents the results of an empirical analysis to support optimal CNI selection based on operational characteristics when building cloud-native environments based on these overall findings.

The primary contributions of this paper are as follows:

- **In-depth analysis of security and network policy features across CNIs**: We conduct a comprehensive investigation into security-related feature variations among Container Network Interfaces (CNIs) to evaluate how each CNI's architectural characteristics and policy support mechanisms contribute to enhancing network security.
- **Quantitative performance evaluation of policy processing overhead**: We present a rigorous quantitative assessment of the impact of Layer 3/4 and Layer 7 network policies on CNI workload performance. This analysis includes detailed measurements and evaluations of throughput, latency, and system resource utilization (CPU and memory) under varying policy enforcement conditions.
- **Tailored guidelines for CNI selection**: We offer detailed guidelines to assist in selecting appropriate CNIs and network policies suited to different cloud-native environments and operational requirements. Based on our findings, we propose optimized network policies for specific scenarios, establishing selection criteria that effectively balance security improvements with performance optimization.

The remainder of this paper is organized as follows: Section II introduces the background and motivation. Sections III and IV present an analysis of the baseline and security features of each CNI, including its network policy. Section V provides an experimental analysis of the performance impact of network policy enforcement. Section VI reviews related work and its limitations. Section VII discusses our key findings and current limitations. Finally, Section VIII concludes with insights and future research.

II. BACKGROUND AND MOTIVATION

A. CONTAINERIZATION AND KUBERNETES

Containerization [25] packages applications and their dependencies into independent units, allowing them to run consistently across environments. Unlike traditional virtualization, which requires a separate guest operating system for each VM, containers share the host operating system kernel. This approach maintains isolation while providing improved resource efficiency and faster deployment times. The efficiency and portability of containerization have made it a fundamental part of cloud-native application development. Kubernetes [7] has emerged as the leading container orchestration platform, where applications run in pods - the smallest deployable units that can contain one or more containers. Kubernetes manages these pods across distributed environments. It automates deployment, scaling, and load balancing. It also provides application resiliency and service discovery. This orchestration enables organizations to efficiently manage containerized applications at scale while maintaining operational consistency.

B. CONTAINER NETWORK INTERFACE

There are several challenges to pod networking across multiple nodes in Kubernetes clusters. The main issues include IP address duplication when container networks share the same IP range. Also, communication difficulties

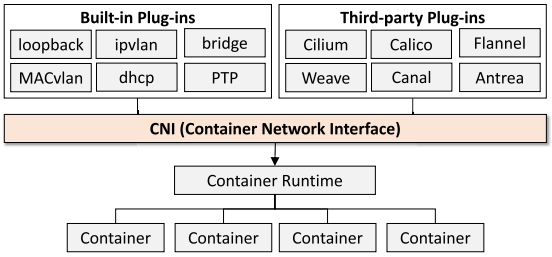

FIGURE 1. Various container network interfaces in Kubernetes.

between nodes occur when pod IP addresses cannot identify physical locations. To address these challenges, Kubernetes introduced the Container Network Interface (CNI) [6].[1] CNI provides a standardized approach that abstracts the complexity of networking while enabling efficient resource management and optimization.

Kubernetes provides 'Kubenet' as the default CNI, but its functionality is severely limited and lacks even basic support for cross-node networking. To address these limitations, several CNI solutions have emerged, as shown in Figure 1. Built-in plug-ins (loopback, bridge, etc.) provide basic networking capabilities. For example, Bridge facilitates container communication by creating a Linux bridge to connect container interfaces on the same node. However, to meet the needs of complex orchestration environments, third-party plug-ins (Cilium, Calico, etc.) have emerged that go beyond basic networking to provide advanced security features such as deep packet inspection, encrypted communications, and dynamic access control, becoming essential components in modern containerized environments.

C. MOTIVATION

As container environments become an integral part of modern infrastructure, ensuring secure network isolation and policy enforcement through CNIs has become critical to protecting containerized applications and data. Most existing studies have focused on evaluating basic network capabilities and performance benchmarks, while the interaction between security policies and system performance remains largely unexplored. There is a notable lack of a perspective that comprehensively considers architectural aspects, security features, and their performance implications together. Thus, we organize the three main limitations as follows.

L1. Lack of analysis of gaps in CNI security features: According to the 2023 CNCF Annual Survey [26], 40% of organizations using cloud services identify security as a major challenge. Monitoring and visibility into network traffic, especially in container environments, is a growing concern. To establish an effective security posture that addresses the multifaceted need for visibility, it is essential to examine both the practical implementation and effectiveness of advanced CNI security features, such as encrypted communications and support for real-time monitoring.

L2. Lack of comparison between network policy capabilities: The 2024 Cloud Security Report from Fortinet [27] shows that 45% of organizations struggle with consistent security policy management in multi-cloud environments. Specifically, differences in policy scope, priority schemes, and enforcement mechanisms across different CNIs add complexity to policy management and often lead to security misconfigurations. Furthermore, the 2024 State of Kubernetes Security Report from Red Hat [28] shows that 40% of organizations have encountered misconfigurations in container environments, with 60% expressing concern about the potential security risks associated with these issues. This highlights the need for a comprehensive analysis of the security mechanisms provided by each CNI, including how their policies are enforced and managed, to ensure optimal policy configuration tailored to organizational needs.

L3. Lack of assessments of policy impacts on system performance: The 2024 Cloud Native Security and Usage Report from Sysdig [29] found that over 50% of container environments lack alerts or limits on system resource usage. This lack of monitoring raises concerns about the ability to accurately predict and manage the performance impact of security policies. This issue is particularly pertinent because each CNI processes policies differently from L3 to L7, and as policy complexity increases, there are notable variations in throughput, latency, and resource utilization patterns. Therefore, empirical criteria are needed to quantitatively assess the impact of each CNI's policy handling mechanisms on overall system performance. Such criteria would enable organizations to select the most appropriate CNI based on their specific infrastructure characteristics and security requirements.

To address those challenges, this study poses the following key research questions:

- **RQ1.** *How do security features across CNIs differ in their implementation, and what are their implications for network policy operations? (Section III)*
- **RQ2.** *How do CNIs differ in their policy extensibility and what capabilities do they provide for advanced policy control? (Section IV)*
- **RQ3.** *How do performance metrics vary with policy complexity and processing layer (L3/L4 vs. L7), and what are the performance and resource trade-offs when selecting a CNI? (Section V)*

III. SECURITY FEATURES ANALYSIS IN CNI

A. UNDERSTANDING CNI ARCHITECTURES

In order to evaluate their security capabilities and performance implications, it is essential to understand the basic architectural features of different CNIs. This section provides a brief overview of the basic features and architectures of each CNI based on the Table 1.

1) FLANNEL

Flannel [19] operates at L2/L3 and implements UDP-based overlay networks through multiple network modes, including VXLAN [30], [31], [32], Host-GW [33], and direct routing.

[1] hereafter referred to as 'CNI' in this paper.

TABLE 1. Summary of basic feature analysis for major CNI plugins within a layered network architecture.

CNI	Distributors	OSI Layer	Network Architecture (Layer-wise)	Data Storage	Multi-cloud Support
Flannel	CoreOS	L2, L3	L2: VXLAN L3: UDP-based Overlay, VXLAN, Host-GW, Direct Routing	etcd	No
WeaveNet	None	L2, L3	L2: Overlay (VXLAN) L3: Overlay, Routed Mesh, VXLAN, Host-GW, Direct Routing	No	Yes (Multi-Hop Routing)
Kube-router	CloudNativeLabs	L3	L3: BGP-based routing	No	No
Calico	Tigera	L3, L4, L7	L3: IP-in-IP tunneling, BGP routing, VXLAN support (optional), eBPF-based datapath L4: eBPF-based kube-proxy replacement (load balancing) L7: Application layer visibility and traffic control through eBPF	etcd (or K8s API)	Yes (Federation)
Cilium	Isovalent	L3, L4, L7	L3: eBPF-based Overlay L4: eBPF Socket Filtering L7: Layer-aware routing through eBPF	etcd (or consul)	Yes (ClusterMesh)
Antrea	VMware	L3, L4, L7	L3: Open vSwitch-based Overlay, VXLAN, GENEVE L4: Flow-based packet filtering via OVS L7: Advanced policy control through Suricata	etcd (or CRD)	Yes (Multi-cluster)

As shown in Figure 2 (A), Flannel's architecture centers around the Flanneld daemon running on each node, which configures local routing tables based on the network configuration stored in etcd. In VXLAN mode, Flannel installs a bridge interface (cni0) with a unique subnet assignment (e.g., 10.42.1.1) for each node, and encapsulates traffic between nodes through a VXLAN interface (flannel.1). This architecture not only prevents network contention through subnet-level isolation, but also simplifies IP address management across the cluster.

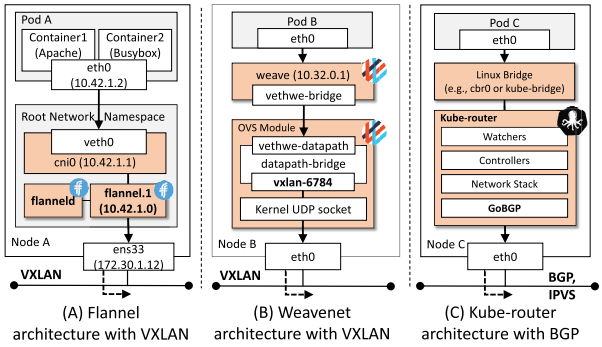

FIGURE 2. Comparison of CNI architectures: Flannel (A) with simple VXLAN-based overlay networking, WeaveNet (B) leveraging VXLAN with OVS for advanced packet processing, and Kube-router (C) utilizing IPVS and BGP for efficient routing and load balancing.

2) WEAVENET

WeaveNet [20] adopts a unique mesh network architecture for container-to-container communications. Operating at L2/L3, it combines overlay networks and a routing mesh topology that can be configured without requiring a separate data storage infrastructure. As shown in Figure 2 (B), the core component of WeaveNet is the Weave router running on each node. These routers interconnect to form a network mesh and provide a bridge interface (vethwe-bridge) on each node for container traffic management.

In VXLAN mode, it has a notable two-tiered structure. The vethwe-datapath and vxlan-6784 interfaces are layered to efficiently handle packet encapsulation and decapsulation. vxlan-6784, WeaveNet's proprietary interface, not only encapsulates inter-node traffic, but also integrates with iptables rules to implement security and routing policies. This architecture provides network encryption capabilities and resilience to network partitioning. In addition, WeaveNet supports efficient container name resolution through its built-in DNS server and enables reliable packet delivery through multiple intermediate nodes via multi-hop routing.

3) KUBE-ROUTER

Kube-router [22] operates primarily at the L3 level, using BGP to exchange routing information between nodes and enable direct communication. It uses IPVS (IP Virtual Server) for network traffic processing, which eliminates the performance overhead associated with inbound source network address translation (SNAT) that has traditionally degraded kube proxy performance. Although IPVS cannot handle packet filtering and certain SNAT operations, it requires minimal use of iptables.

As shown in Figure 2 (C), Kube-router is centered around a single Kube-router daemon running on each node. This daemon communicates directly with the Kubernetes API server through watchers to detect changes in cluster state, and updates network configurations through controllers. The network stack provides high-performance service proxy functionality using IPVS, while built-in GoBGP allows each node to act as a BGP router. This integrated structure enables efficient pod-to-pod communication without overlay networks and optimizes overall network performance along with pod network management through Linux Bridge.

4) CALICO

Calico [21] is a solution that provides advanced capabilities through BGP-based routing, making it suitable for deployment in large cluster environments. As depicted in Figure 3, the Calico architecture consists of a Calico agent running on each node, which includes Felix, BIRD (BGP client), CNI plugin, and IPAM plugin.

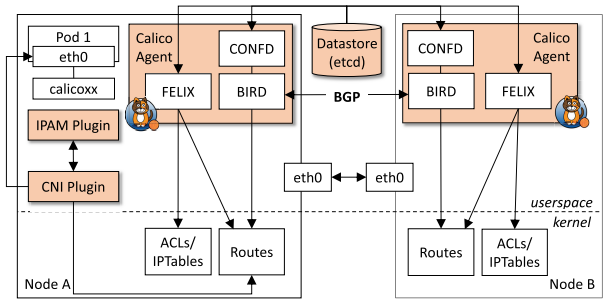

FIGURE 3. Calico CNI architecture with BGP.

As the main agent, Felix configures routes for each pod and implements network policies using IPtables rules to control traffic flow. The BIRD daemon facilitates the exchange of routing information between nodes via BGP, enabling efficient routing without overlay networks. The IPAM handles IP address allocation and management. Felix and BIRD operate in userspace, with the actual packet processing and routing taking place in the kernel-space network stack. The architecture uses either etcd or the Kubernetes API as the data store for maintaining network policies and configuration information.

Recent architectural enhancements include support for eBPF [34], [35], [36], which enables direct kernel-level packet processing and provides more efficient service load balancing by replacing kube-proxy. In addition, Calico's Federation [37] enables centralized network policy management and synchronization across geographically distributed clusters in multi-cloud environments, ensuring consistent security policy enforcement.

5) CILIUM

Cilium [23] is a CNI that adopts eBPF to provide advanced networking, security, and observability. As shown in Figure 4, the architecture uses the Cilium Agent as the central component in the user space. This agent interacts with the CNI plugin, the Cilium CLI, and the libbpf (Go library) to generate eBPF program source code. The resulting byte code is then loaded into the kernel after passing the eBPF verifier.

In kernel space, loaded eBPF programs are executed at multiple hook points, including network device-level XDP hooks, TC ingress/egress hooks, and socket-level hooks. The eBPF maps facilitate data exchange between the Cilium Agent in user space and the eBPF programs in kernel space, enabling dynamic policy updates and state management.

The Cilium Operator interfaces with the Kubernetes API to manage resources and policies at the cluster level. In addition, Cilium's ClusterMesh [38] extends service namespaces across multiple clusters to behave like a single unified cluster, enabling seamless microservice communication and global load balancing for high availability in multi-clouds.

6) ANTREA

Antrea [39] is based on the Open vSwitch (OVS) [39] to provide high performance packet processing. As shown in

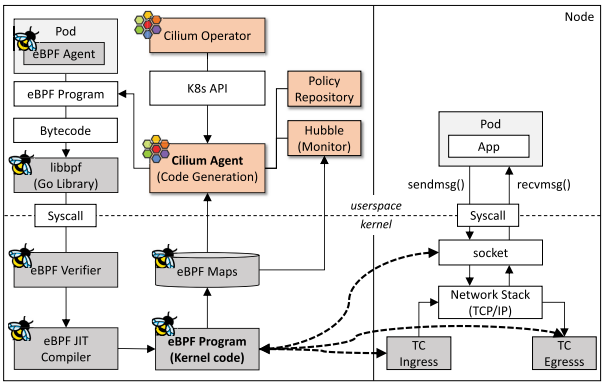

FIGURE 4. Cilium CNI architecture with eBPF.

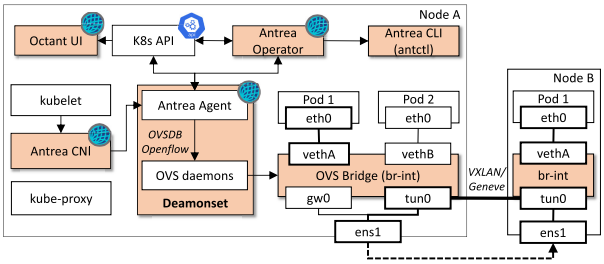

FIGURE 5. Antrea CNI architecture with OVS.

Figure 5, Antrea is built around the Antrea Agent, which is deployed as a daemon set on each node to manage the OVS and handle local Pod networking requirements. Under the supervision of the Antrea Controller, the Agent interacts with the OVS daemon to establish OpenFlow rules and manage the OVS configuration via the OVSDB. The Antrea communicates with the Kubelet to configure the pod network interfaces and connect them to the OVS bridge (br-int), with each pod connected to the OVS bridge via veth pairs. Communication between nodes is enabled using either VXLAN or GENEVE tunnels carried over physical network interfaces such as ens1.

Antrea also integrates with the Octant UI [40], which provides visualization capabilities for network topology and policy management. Antrea uses either etcd or Kubernetes Custom Resource Definitions (CRD) [41] to maintain network configuration information. In addition, the multi-cluster controller [42] uses a leader-member cluster architecture based on the ClusterSet. Leader clusters are responsible for managing network policies and resource synchronization, while member clusters apply these resources locally. This architecture is enhanced by a Common Area shared repository that enables efficient resource and namespace sharing across clusters.

B. SECURITY FEATURE COMPARISON ACROSS CNIS

This section presents an overall comparative analysis of the major CNIs based on security features and investigates the impact of CNI architecture selection on security features. The analysis is summarized in Table 2.

TABLE 2. Comparison of security feature analysis for major CNI plugins.

CNI	Network Policy	Encryption	Authentication	Multi-tenancy	Traffic Visibility
Flannel	No	No	No	No	No
WeaveNet	Yes	IPsec (built-in), TLS	Certificates	Basic overlay isolation	Weave Scope
Kube-router	Yes	No	No	Basic partitioning (namespace)	No
Calico	Yes	WireGuard, IPsec	Certificates, Tokens	RBAC and hierarchical policies	Flow Logs
Cilium	Yes	IPsec, TLS, WireGuard	mTLS	Identity-based (eBPF micro-segmentation)	Cilium Hubble
Antrea	Yes	IPsec, WireGuard	Certificates	Policy-based (namespace + OVS)	Flow Exporter

1) SUPPORT FOR NETWORK POLICY

Support for network policies provides a security mechanism for controlling inter-pod communication within clusters. Although Flannel does not provide native policies, most other CNIs support Kubernetes network policies [43], with some providing advanced policy capabilities.

Notably, Cilium uses its eBPF to enforce policies directly at the kernel level. Policy processing occurs sequentially at multiple hook points, including XDP hooks that handle initial packet filtering at the network device level, TC ingress/egress hooks that handle L3/L4 policies, and socket-level hooks that implement L7 policies. Antrea implements policy control through a hierarchical OpenFlow pipeline structure within the OVS. The policy rules are converted into OpenFlow entries that form distinct tables for different policy types, with connections tracked through the OVS conntrack module for stateful policy enforcement.

Calico, WeaveNet, and Kube-router employ iptables-based policy enforcement, although each implements it differently: Calico through Felix's structured pipeline with distinct ingress/egress chains, WeaveNet through its mesh network's synchronized routing tables, and Kube-router through direct integration with its BGP routing system. The detailed analysis of network policies and their impact on performance can be found in Section IV and V.

2) ENCRYPTION AND AUTHENTICATION MECHANISMS

The encryption and authentication mechanisms are indispensable elements in guaranteeing the confidentiality and integrity of network traffic in cloud-native environments. Flannel and Kube-router's reliance on basic Kubernetes API server authentication reflects their focus on simplicity, which consequently limits their ability to offer fine-grained authentication policies. In contrast, other CNIs implement comprehensive security protocols. WeaveNet integrates IPsec within its mesh network for peer-to-peer encryption and implements certificate-based authentication for peer verification.

Calico combines IPsec for direct packet encryption in the data plane with WireGuard for lightweight VPN tunneling, supporting both certificate and token-based authentication mechanisms. Cilium implements multi-layer encryption using IPsec and WireGuard in its BPF-based data plane, while also providing TLS for service-level encryption and mTLS for service-to-service authentication with unique workload identities. Antrea implements both IPsec and WireGuard in its OVS-based data plane, using IPsec for tunneled node-to-node communications and WireGuard for encrypted peer networking, while centrally managing certificates centrally for OVS component authentication.

Encryption and authentication mechanisms have a direct impact on the security and efficiency of policy enforcement. Calico's WireGuard integration automatically encrypts traffic allowed by L3/L4 policies, enabling unified policy and data protection management in a single location. In particular, Cilium's mTLS implementation combines fine-grained, service-level access control with encryption, enhancing the effectiveness of L7 policies. This integrated approach strengthens end-to-end security while reducing the complexity of policy management.

3) SUPPORT FOR MULTI-TENANCY

In cloud-native environments, multi-tenancy is a security requirement that ensures the isolation of multiple tenants sharing cluster resources. To effectively implement multi-tenancy, several isolation mechanisms are required, including namespace separation, resource isolation, network isolation, storage isolation, and granular access control. The level of multi-tenancy support in a CNI can be analyzed based on the comprehensiveness of these isolation mechanisms.

Cilium implements identity-based access control using workload identities instead of IP addresses, leverages Kubernetes namespaces for tenant separation, and enables network traffic isolation through its Multi-Network. Calico reinforces namespace isolation through RBAC-based policies and BGP-based network segmentation and supports resource and network isolation through a hierarchical policy model. Antrea provides namespace-based isolation but lacks granular control over tenant resource sharing, while WeaveNet provides only basic namespace isolation with encrypted communications and Kube-router provides basic network partitioning without additional tenant isolation. Flannel lacks any multi-tenancy support, making it unsuitable for environments that require tenant networking.

Multi-tenancy capabilities enable effective network policy separation and enforcement. Calico's hierarchical policy model combines RBAC with network segmentation, allowing systematic configuration of top-level baseline policies and tenant-specific custom policies.

4) TRAFFIC VISIBILITY AND MONITORING CAPABILITIES

Network traffic monitoring enables network activity observation and control through packet capture, real-time analysis, and anomaly detection. While basic CNIs such as Flannel

and Kube-router lack self-monitoring capabilities, other CNIs provide comprehensive monitoring solutions. Hubble [44] integrates with Cilium to provide comprehensive visibility into service-to-service communication patterns, thereby supporting advanced policy verification and bottleneck identification through Prometheus [45]. Calico's Flow Logs [46] facilitate traffic analysis by integrating with tools such as Elasticsearch [47] or Splunk [48] for rapid anomaly detection.

Furthermore, Antrea's Flow Exporter [49] employs the IPFIX protocol and utilizes OVS with the conntrack module for flow capture, periodically polling network connection states through the conntrack and integrating with visualization tools such as the ELK Stack [50] or Grafana [51]. WeaveNet's WeaveScope [52] offered visualization capabilities comparable to Hubble, facilitating precise modeling of communications between deployments, stateful sets, and external entities with real-time container network monitoring. However, due to severe scalability issues in large environments, development was discontinued in 2021, and then completely stopped when Weaveworks was shut down in February 2024. This left a void in the market for topology-centric monitoring solutions, particularly those seeking an alternative to resource-intensive approaches.

Traffic monitoring plays a critical role in validating and optimizing policy operations. CNI monitoring tools provide the visibility to detect policy violations and anomalous traffic patterns in real-time. This goes beyond simple monitoring and provides a foundation for measuring and validating policy effectiveness to continuously improve security policies. This visibility is especially critical in complex microservice environments to quickly identify and resolve policy configuration errors or unintended side effects.

Insight 1. Our analysis reveals that essential security features, such as encryption and authentication, are absent in some CNIs. Surprisingly, multi-tenancy, another critical feature for isolation, is either unsupported or only partially implemented in existing CNIs. This suggests that CNI contributors prioritize network functionality over security.

IV. NETWORK POLICY PROFILING IN CNI

This section presents a detailed network policy analysis for each CNI, based on Table 3. To analyze the strengths and limitations of each CNI's policy features, our analysis examines policy types, scope, supported layers, and priority.

A. EVOLUTION AND CURRENT TRENDS

The evolution of network policies has advanced from basic network isolation to supporting complex cloud-native architecture requirements. As shown in Figure 6, the Kubernetes Network Policy [43] structure consists of hierarchical components: Metadata containing basic policy information, Selectors using labels and Common Expression Language (CEL) [53] for pod targeting, and Rules comprising Policy Type, Action, Target specifications, and Traffic Rules. These components can be combined to create diverse policy configurations, such as an *allow-db-egress* policy that restricts egress traffic to a specific CIDR range (192.168.0.0/16) and TCP port 80 for pods labeled with `app: db`.

Although the fundamental network policy model is centered on single-cluster and constrained traffic control, the growing intricacy of security threats has revealed the limitations. This has necessitated an evolution toward extended policy schemes through Custom Resource Definitions (CRD) [41], which enable a wider scope and more precise control. For example, Calico's IPPool [54], when integrated with BGP peering information, ensures that traffic can traverse is only possible through authorized routing paths, mitigating IP spoofing attempts through precise IP range control and routing validation. Moreover, Antrea's `NodeStatsSummary` CRD [55] provides comprehensive monitoring of policy enforcement status at the node level for real-time detection of anomalous traffic patterns, such as a sudden spikes in connection attempts.

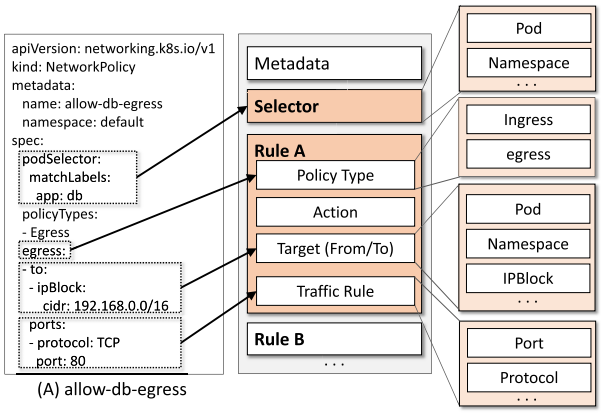

FIGURE 6. The overview of the default network policy structure provided by Kubernetes.

B. POLICY SCOPE AND LAYER-SPECIFIC RULE CHARACTERISTICS

Network policies in container networking differ in their scope and layer-specific rule support. Policy scope is divided into namespace-level and cluster-wide policies. While WeaveNet and Kube-router are limited to namespace-level Kubernetes network policies, advanced CNIs support both Kubernetes network policies and extend their policy scope through custom resource definitions (CRDs). Namespace-level policies provide fine-grained control within specific namespace boundaries, enabling independent management of security rules. In contrast, cluster-wide policies (Calico's GlobalNetworkPolicy [56], Cilium's ClusterwideNetworkPolicy [57], and Antrea's ClusterNetworkPolicy [58], etc.) apply uniformly across all namespaces, ensuring consistent security standards across the cluster.

At Layer 3/4, CNIs extend the Kubernetes network policy beyond IP/CIDR-based filtering and port/protocol restrictions. Typically, CNIs extend policy controls with comprehensive ingress/egress port ranges and ICMP/ICMPv6 type management. Cilium network policy facilitates the

TABLE 3. Comparison of network policy features across major CNI plugins.

CNI	Types	Scope	Layers	L3/L4	L7	Target	Actions	Priority
Flannel	No	No	No	No	No	No	No	No
WeaveNet	Kubernetes Policy	Namespace	L3, L4	Ingress, Egress	No	Pod, Namespace	Allow, Deny	No
Kube-router				Ingress				
Calico	Kubernetes Policy, Calico Policy	Namespace, Cluster-wide	L3, L4, L7	Ingress, Egress	HTTP	Pod, Namespace, Label, ServiceAccount, IPBlock	Allow, Deny, Log, Pass	Order
Cilium	Kubernetes Policy, Cilium Policy	Namespace, Cluster-wide	L3, L4, L7	Ingress, Egress	HTTP, gRPC, Kafka	Pod, Namespace, Label, ServiceAccount, IPBlock, DNS, Node, Endpoint	Allow, Deny	No
Antrea	Kubernetes Policy, Antrea Policy	Namespace, Cluster-wide	L3, L4, L7	Ingress, Egress	HTTP	Pod, Namespace, IPBlock	Allow, Drop, Reject, Pass	Tiers

definition of policies through the use of DNS-queryable domain names for endpoints, providing IP address handling from DNS responses analogous to CIDR-based rules.

Calico network policy implements CIDR block pattern matching and integrates IPAM to offer policies based on IP pools and subnets, thereby enhancing stateful protocol filtering mechanisms. Antrea network policy features topology-aware policy enforcement, taking into account physical infrastructure layouts and zone boundaries, while supporting stateful connection tracking integrated with its filtering.

At Layer 7, Cilium Network Policy handles DNS request filtering itself and implements comprehensive application layer control including HTTP (method, path, headers), gRPC (service/method calls, metadata), and Kafka (topics, consumer groups) communications. Antrea Network Policy focuses primarily on HTTP traffic control through URL filtering and method-based access rules, and integrates these capabilities into its OpenFlow pipeline structure. While Calico also supports Layer 7 policies, it is limited to HTTP traffic control and is only available in the enterprise version.

C. TARGET AND ACTION SPECIFICATION

While Kubernetes policies are limited to pod and namespace selectors, CNIs extend with sophisticated target specifications and diverse policy enforcement actions. Calico extends beyond basic selectors through the use of advanced label expressions using Common Expression Language (CEL), enabling sophisticated multi-dimensional targeting with service account selectors and complex label combinations.

Cilium implements a targeting system through endpoints and entities. Its endpoint abstraction serves as the fundamental unit for policy application, while entity-based selectors enable broader scope targeting from hosts to external networks (via the 'world' entity) and cluster-wide resources. Antrea enhances targeting flexibility through its ClusterGroup abstraction, which consolidates selectors - including pod, namespace, and node - into policy targets.

With respect to the mechanisms of action, Kubernetes network policy implements a whitelist approach, permitting only explicitly permitted traffic while blocking all others by default. Similarly, Cilium provides explicit `Allow` and `Deny` actions, following the 'most restrictive rule takes precedence' principle for policy conflict resolution.

In contrast, Calico offers more granular control through `Allow`, `Deny`, `Log`, and `Pass` actions. The `Log` action enables continuous monitoring of traffic flow for security audits and troubleshooting, while the `Pass` action terminates policy evaluation at the current layer and moves to the next priority layer or other network policies. If no policies exist, traffic is allowed by default. Antrea provides `Allow`, `Drop`, `Reject`, and `Pass`. Antrea's `Pass` action, similar to Calico's `Pass`, enables hierarchical policy evaluation. The `Drop` action silently discards packets, enhancing security through opacity. The Reject action generates explicit denial responses, such as TCP Reset (RST) or ICMP 'administratively prohibited' messages.

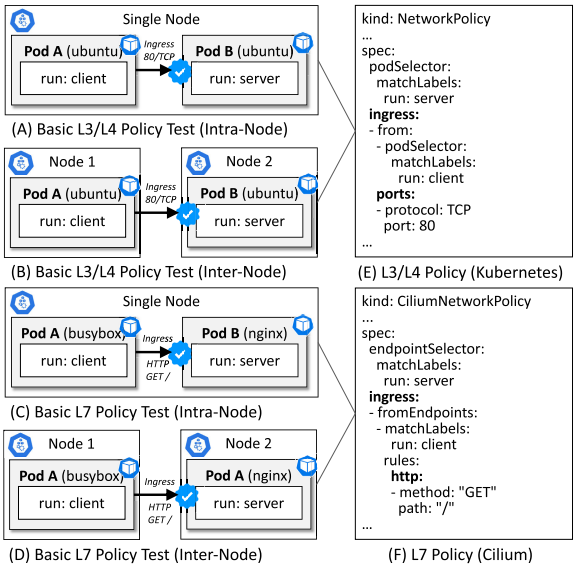

FIGURE 7. Test scenarios for enforcing L3/L4 and L7 network policies.

D. POLICY PRIORITY ANALYSIS

1) EXPERIMENTAL METHODOLOGY

The management of policy priorities is of critical importance in resolving policy conflicts and maintaining consistent enforcement in complex cloud-native environments. To evaluate policy priorities, two types of analysis were conducted.

First, an examination of the processing order between Kubernetes network policies and CNI policies was conducted by applying them simultaneously and observing which policy takes precedence.

Second, we investigated how different CNIs implement policy precedence through their built-in priority mechanisms, such as Calico's `order` and Antrea's `priority` and `tier` fields, by testing various priority configurations. For these two analyses, we employed the Basic L7 Policy Test scenario depicted in Figure 7 (D), wherein we deployed an Nginx server pod and a Busybox client pod across disparate nodes and evaluated the efficacy of HTTP GET requests between them.

2) CNI-SPECIFIC IMPLEMENTATION CHARACTERISTICS

As illustrated in Figure 8, two pods (pod-a and pod-b) were deployed on distinct worker nodes (192.168.184.146 and 192.168.184.147), and three pivotal test scenarios were conducted. The experimental results obtained revealed that when Calico's Allow policy was applied alone (B-1), pod-to-pod communication was successful, with pod-a successfully downloading index.html from pod-b. Conversely, when Kubernetes' default deny policy was applied (B-2), communication was blocked as expected. Notably, in case (C), where both policies were applied concurrently, the communication remained blocked despite Calico's allow policy. These findings demonstrate a clear precedence of the Kubernetes policy over the CNI-specific policy.

A architectural analysis reveals that Calico's Felix translates policies into iptables rules in chains that are distinct from those utilized by Kubernetes. However, these Calico-specific chains are positioned subsequent to the Kubernetes' iptables processing flow. Additionally, both Cilium, which employs TCP, XDP, and eBPF maps, and Antrea, which converts to OVS Flow Rules applied to node bridges, operate their processing hooks or OVS pipelines after the point where Kubernetes evaluates network policies, despite their advanced processing capabilities. Consequently, when packets enter the network stack, Kubernetes policies are applied first due to the iptables processing flow, resulting in packets being dropped before reaching any CNI-specific processing points if traffic is blocked.

This sequential processing, while appearing as a structural limitation, actually enables a complementary relationship between Kubernetes and CNIs. Through this processing approach, CNIs respect Kubernetes' networking foundation and add essential components for modern cloud-native environments through extensibility, rather than replacing or overriding the underlying infrastructure.

3) PRIORITY HIERARCHY ANALYSIS

As shown in Table 3, among the primary CNIs, only Calico and Antrea implement an explicit priority field. Calico implements a single-tier priority system through its `order` field, where numerical values determine policy precedence. Policies with lower `order` values (e.g., 500) take precedence over those with higher values (e.g., 1000). This numerical ordering allows administrators to explicitly define the processing sequence of network policies.

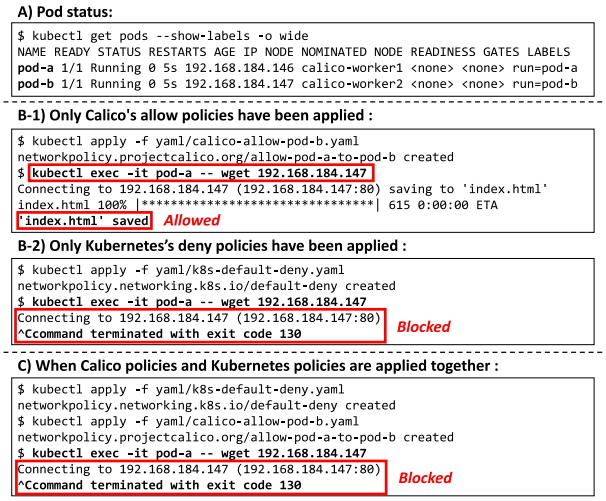

FIGURE 8. The results of priority testing between Kubernetes and Calico policies.

Antrea implements a hierarchical three-level priority consisting of `tier`, `policy`, and `rules`. At the highest level, the `tier` system comprises six preconfigured tiers: emergency, securityops, networkops, platform, application, and baseline, with emergency having the highest priority. Within each `tier`, policies are ordered using `policy` priority values ranging from 1.0 to 10000.0. `rule` is employed to determine the sequence of rule application within a single policy. At the lowest level, rules within each policy are processed sequentially.

Our experimental analysis confirmed this hierarchical enforcement: when a traffic-blocking policy in the securityops tier (priority 5.0) competed with a traffic-allowing policy in the application tier (priority 10.0), the securityops tier policy prevailed, thereby demonstrating the precedence of `tier` over `policy` priority values. To ensure system stability and manageability, Antrea enforces constraints by limiting the number of user-defined tiers to 20 and restricting unique priority values within the baseline tier to 150.

Insight II. We found that most CNI policies are built on Kubernetes Policy, extending functionalities across various network layers. In addition, we confirmed that all CNIs have a processing order, which is processed after the Kubernetes policy is developed, to support CNI-specific policies together while reducing the likelihood of conflict with the underlying policy. Cloud administrators should consider these details, such as processing order and layers for consistent policy enforcement.

V. PERFORMANCE EVALUATION

A. EXPERIMENTAL SETUP AND SCENARIO OVERVIEW

To evaluate the network policy enforcement performance of each CNI plugin, experiments were conducted in a three-node Kubernetes cluster (v1.30) running on a server equipped with

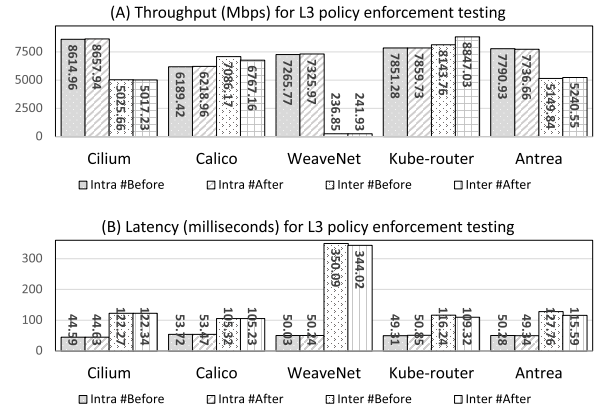

FIGURE 9. The results of throughput (Mbps) and latency (milliseconds) for L3 testing pre- and post-policy enforcement across different CNIs.

Intel(R) Xeon(R) Silver 4210R CPU 2.40 GHz and 256 GB RAM. The cluster consists of one control plane node and two worker nodes, each allocated 4 vCPUs and 8GB RAM, with containerd serving as the container runtime. For pod configurations, widely-used base images were strategically selected: Ubuntu-based pods for L3/L4 policy testing with netperf traffic generation, Nginx for server-side HTTP services, and Busybox for client-side HTTP requests. Each test scenario operates in dedicated namespaces with defined network policies to ensure consistent and isolated environments.

As shown in Figure 7, four test scenarios were designed to evaluate both intra-node and inter-node policy enforcement capabilities. Scenarios (A) and (B) focus on L3/L4 policy performance using Ubuntu-based pods with netperf traffic generation. For L7 policy evaluation, scenarios (C) and (D) use Nginx server pods to handle HTTP GET requests from Busybox client pods. These scenarios control traffic using two types of policies: (E) shows the standard Kubernetes NetworkPolicy, which allows TCP traffic on port 80 between labeled client and server pods for L3/L4 control, and (F) demonstrates CNI-specific L7 policies (e.g., CiliumNetworkPolicy) that allow HTTP GET requests to the root path (“/”). These L7 policies are implemented to provide consistent behavior across different CNI implementations.

The performance evaluation examines four key metrics: network throughput, end-to-end latency, system resource utilization (CPU and Memory) during policy enforcement, and scalability as policy and pod complexity increase. To ensure statistical reliability, each measurement cycle runs at ten-second intervals for five minutes, and this process is repeated ten times to derive the average values.

B. PERFORMANCE OVERHEAD OF LAYER 3 POLICY PROCESSING

1) BASIC PERFORMANCE ANALYSIS OF POLICY ENFORCEMENT

To evaluate the fundamental impact of network policies on CNI performance, we analyzed four key metrics. Throughput (Mbps) measures the actual data transfer capacity between containers, which has a direct impact on application performance. Latency (ms) reflects network responsiveness and is critical for time-sensitive cases. CPU(%) and memory (GB) utilization indicate the resource overhead of policy enforcement, which is essential for understanding the operational costs of each CNI. The experiments involved two tools: netperf [59] for performance metrics and the System Activity Report (sar) [60] for resource utilization.

As shown in Figure 9 through throughput and latency measurements, in intra-node environments, Cilium maintains the highest throughput at 8.6K Mbps both before and after policy enforcement, while other CNIs show stable performance in the 6.0K-7.0K Mbps range. However, inter-node testing exposed significant differences. Kube-router exhibited superior performance, with an 8.1K Mbps pre-policy throughput that increased to 8.8K Mbps post-policy.

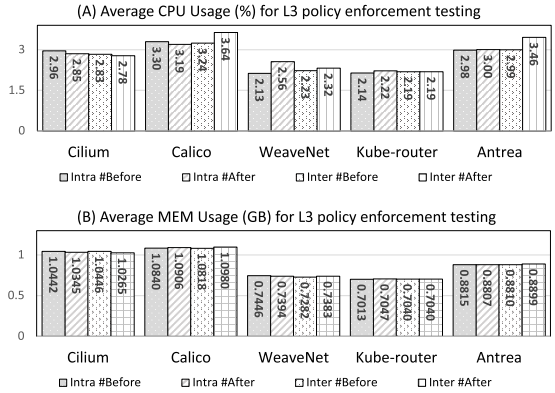

FIGURE 10. The results of average CPU (%) and memory (GB) usage pre- and post-policy enforcement across different CNIs in L3 testing.

Conversely, WeaveNet’s performance dropped from 7.3K Mbps in intra-node scenarios to 240 Mbps in inter-node scenarios, a 97% reduction. Both Cilium and Antrea showed a moderate decrease in throughput, reaching approximately 5.0K Mbps in inter-node environments, a 42% decrease compared to their intra-node performance. The latency measurements showed a similar trend, with all CNIs maintaining consistent response times between 44 and 53 ms in intra-node scenarios. However, in inter-node environments, WeaveNet’s latency increased significantly to 350 ms, nearly triple the 105-127 ms range observed for the other CNIs after policy enforcement.

As illustrated in Figure 10, Calico’s CPU utilization increased by 0.45 %p from 3.24% to 3.64% following policy enforcement in inter-node scenarios, exhibiting the most substantial increase among all CNIs. In inter-node environments, WeaveNet and Kube-router exhibited the most efficient CPU utilization at 2.32% and 2.19%, respectively. Following the enforcement of the policy, the results pertaining to memory usage indicate that Calico utilizes approximately 1.08 GB in intra-node environments and 1.09 GB in inter-node environments. In contrast, WeaveNet and Kube-router maintain significantly lower memory footprints of approximately 0.73 GB and 0.70 GB, respectively.

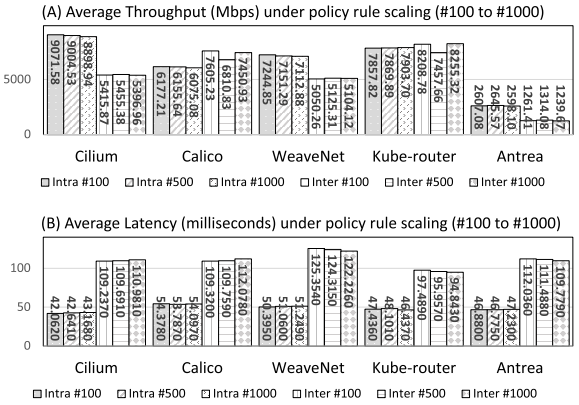

FIGURE 11. The impact of policy complexity (100, 500, 1000 policies) on throughput and latency in L3 testing.

These performance patterns reflect specific architectural design choices of each CNI. WeaveNet's dual-layered VXLAN architecture requires sequential packet processing through the vethwe-datapath and vxlan-6784 interfaces, causing significant overhead in inter-node communication. Calico's BGP daemon demands higher CPU resources due to continuous routing table maintenance and policy rule enforcement through iptables. Cilium leverages eBPF programs in kernel space, eliminating context switching overhead and enabling efficient intra-node packet processing. Kube-router's direct BGP peering approach without overlay encapsulation minimizes processing overhead, resulting in superior inter-node performance.

The minimal memory growth observed during policy enforcement is due to the efficient state management of modern CNIs. Typically, rules are stored as compact data structures in kernel space (e.g., eBPF maps, iptables rules), with only policy metadata maintained in userspace memory. This configuration allows packet processing to operate efficiently without continuous memory allocation.

2) PERFORMANCE DEGRADATION DUE TO POLICY COMPLEXITY

To understand the impact of increasing network policy rules on CNI performance, we scaled policy rules from 100 to 500, and finally to 1,000 rules. By isolating policy complexity as the sole variable, we measured the changes in throughput, latency, CPU usage, and memory consumption in both intra-node and inter-node environments. The experiment maintained a 1:1 pod communication pattern while increasing the number of port ranges between them.

As shown in Figure 11, in throughput measurements, Cilium's intra-node performance dropped only 2.2% from 9.0K Mbps at 100 rules to 8.8K Mbps at 1000 rules. Similarly, Kube-router demonstrated stable inter-node performance, maintaining approximately 8.2K Mbps across all policy scales. However, Antrea exhibited significant performance degradation, with intra-node throughput dropping by 58% from 2.6K Mbps to 1.2K Mbps as rules increased from 100 to 1000. The latency measurements showed corresponding trends, with Cilium maintaining stable response times around 43ms in intra-node scenarios across all policy scales. In contrast, WeaveNet showed the highest latency increase of 245% in inter-node communication, rising from 50.39ms at 100 rules to 122.26ms at 1000 rules.

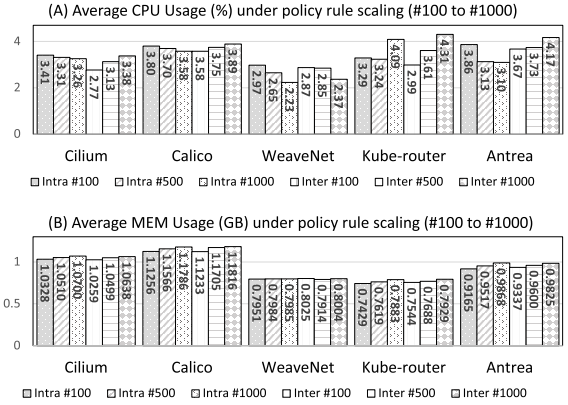

FIGURE 12. The impact of policy complexity (100, 500, 1000 policies) on CPU and memory usage in L3 testing.

As demonstrated in Figure 12, Calico showed the highest CPU sensitivity to policy scaling, with usage increasing from 3.80% at 100 rules to 3.89% at 1000 rules in inter-node environments, representing a 2.4% increase. Antrea followed a similar pattern, showing a 13.6% increase from 3.67% to 4.17%. In terms of memory consumption, Calico consistently maintained the highest usage, reaching 1.18GB at 1000 rules, while WeaveNet and Kube-router remained below 0.8GB even at maximum policy complexity. Notably, Cilium's memory usage increased from 1.03GB to 1.07GB when scaling from 100 to 1000 rules.

An analysis of the policy scaling results revealed that Cilium's throughput and latency showed performance decline due to accumulated overhead from eBPF map lookups and program execution in large policy sets. Antrea's significant throughput decline stems from increased overhead in Open vSwitch flow table management, where each packet requires sequential verification against an expanding number of flow entries. WeaveNet's rising latency with policy scaling signifies that its VXLAN overlay becomes a bottleneck when packets must traverse multiple network abstractions while checking against expanded rule sets.

Concurrently, Calico's increased CPU utilization pattern reflects the escalating complexity of rule chain processing, as each packet must be checked against an expanding rule list. In contrast, Kube-router's IPVS-based hash processing approach maintains consistent performance regardless of the number of rules because it avoids the linear scaling problems associated with traditional chain-based packet processing.

3) PERFORMANCE IMPACT OF NETWORK POLICY UNDER CONCURRENT CONNECTION SCALING

To analyze how concurrent pod connections affect network policy performance in inter-node environments, we evaluated performance variations by increasing the number of client

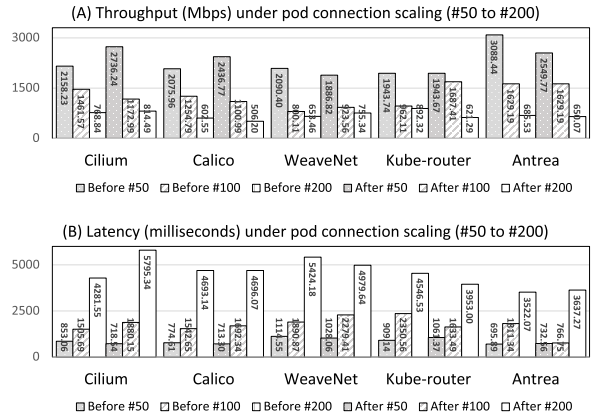

FIGURE 13. The impact of policy enforcement under increasing concurrent connections (50 to 200 pods) across different CNIs.

pods under a single policy. The scenario depicted in Figure 7 (B) was utilized to conduct the experiment, wherein the number of client pods accessing the server pod was incrementally scaled from 50 to 200. The throughput and latency changes before and after applying the policy are shown in Figure 13.

The throughput measurements revealed that before policy enforcement, Cilium's performance decreased by 64% from 2.2K Mbps at 50 pods to 0.8K Mbps at 200 pods. After policy enforcement, similar degradation of 70% was observed. WeaveNet exhibited the most severe throughput reduction, declining by 88% from 1.9K Mbps to 0.2K Mbps before policy enforcement and 89% after policy enforcement. Notably, Antrea showed the highest initial throughput of 3.1K Mbps at 50 pods both before and after policy enforcement, but experienced substantial degradation of 59% and 79% respectively when scaling to 200 pods. In contrast, Kube-router maintained relatively stable performance until 100 pods with minimal difference between pre and post-policy states (approximately 1.9K Mbps), but showed significant degradation of 68% when scaled to 200 pods.

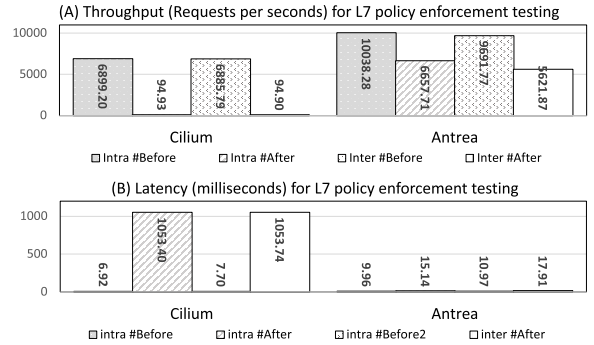

FIGURE 14. The comparison of L7 policy processing overhead between Cilium and Antrea.

Latency measurements demonstrated that before policy enforcement, Antrea showed the most dramatic latency increase of 1,410% from 896ms at 50 pods to 13.5K ms at 200 pods. After policy enforcement, WeaveNet exhibited the highest relative latency increase of 321% from 1.1K ms to 4.7K ms when scaling from 50 to 200 pods. Interestingly, Cilium maintained the most consistent latency characteristics after policy enforcement, with the smallest relative increase of 236% from 719ms to 2.4K ms between 50 and 100 pods, although this advantage diminished at 200 pods with a final latency of 5.8K ms. Calico and Kube-router showed similar latency patterns, both experiencing approximately 300% increases when scaling from 50 to 200 pods under policy enforcement.

Beyond the above measures, we observed variations in session stability as pod counts increased. All CNIs had stable connectivity up to 100 pods, but once this threshold was crossed, we saw significant differences. Cilium demonstrated remarkable connection stability, maintaining approximately 183 successful connections out of 200 attempted connections even after policy enforcement, although connection establishment times increased slightly.

In stark contrast, WeaveNet's stability deteriorated significantly, dropping to about 114 successful connections out of 200 before policy enforcement and further declining to 86 after policy enforcement. This instability was manifested by frequent connection reset events during VXLAN tunnel establishment. In particular, Calico and Antrea exhibited unexpected behavior at 200 pods, where initial policy synchronization phases resulted in complete connection failures across multiple test sets. These CNIs required considerable time over ten minute to stabilize. This behavior revealed underlying limitations in their connection tracking mechanisms and policy synchronization processes, particularly as their respective tracking tables approached capacity limits.

C. PERFORMANCE OVERHEAD OF LAYER 7 POLICY PROCESSING

To evaluate the performance impact of L7 policies, we conducted experiments using (C) and (D) shown in Figure 7, which represent intra-node and inter-node environments respectively. Performance measurements focused on two key metrics: throughput measured in requests per second (RPS) and latency, measured in milliseconds. Tests were conducted using Apache Benchmark (ab) [61], a tool designed specifically for HTTP server benchmarking, by generating 100 concurrent connections over a 20-second period. The results comparing performance before and after L7 policy deployment for Cilium and Antrea are shown in Figure 14.

The results demonstrated that Cilium experienced a significant degradation within the node, with throughput declining from 6.8K RPS to 94 RPS and latency increasing from 6.9 ms to 1.0K ms. This considerable decline can be attributed to the logic of Cilium's L7 policy processing. While Cilium demonstrates superior performance in kernel-level packet processing through the use of eBPF for L3/L4 policies, the enforcement of L7 policies necessitates the redirection of packets to a userspace proxy for the purpose of deep packet inspection of application layer protocols (such as HTTP

and DNS). This processing causes significant overhead due to context switching between the kernel and userspace and additional memory copies of packet data, as packets that would normally be processed completely within the kernel must now cross the kernel-userspace boundary for L7 inspection.

However, Antrea has shown relatively stable performance even after policy enforcement. In intra-node communication, throughput decreased from 10K RPS to 6.6K RPS, accompanied by a slight increase in latency, from 9.9 ms to 15.1 ms. The relatively favorable performance can be attributed to Antrea's application-aware engine for L7 policy processing. While Antrea's flow rules are primarily designed to handle L3/L4 traffic, for the enforcement of L7 policies, it integrates with Suricata [62] for the filtering of HTTP traffic. The TrafficControl API selectively redirects HTTP traffic to Suricata for application-layer inspection, supporting fine-grained filtering based on specific HTTP URIs and methods. Antrea still requires userspace switching for Suricata processing, obviously, but because it specializes in HTTP filtering only, it has less overhead than the Cilium approach, which supports multiple L7 protocols.

Insight III. The most critical layer in our experiments is L7, as few CNIs demonstrate adequate performance at this level. Cloud administrators must carefully select and deploy CNIs when prioritizing security enforcement at L7, where many cloud-targeted attacks, such as APTs and lateral movement, commonly occur.

VI. RELATED WORK

In this section, we review recent CNI research in performance-oriented studies and security-oriented analyses, as summarized in Table 4.

A. PERFORMANCE-ORIENTED RESEARCH FOR CNI

Recent research on CNIs has primarily focused on evaluating fundamental performance characteristics in various deployment scenarios. Kang et al. [63] conducted a comprehensive analysis of Flannel, WeaveNet, and Kube-router in edge computing environments, focusing on DDS (Data Distribution Service) applications' performance under different network conditions. Their evaluation metrics included throughput, latency, and resource utilization, though security aspects were limited to basic encryption overhead analysis.

Sekigawa et al. [64] examined CNI performance in telecom, evaluating Flannel, Calico, Cilium, and Kube-OVN with emphasis on latency-sensitive metrics and packet processing efficiency. While they addressed the impact of overlay versus underlay networking approaches, their security analysis remained peripheral. Dakić et al. [13] presented an extensive performance evaluation of Antrea, Flannel, Calico, and Cilium in high-performance computing and AI workloads. Their study particularly focused on the impact of network tuning parameters such as MTU sizes and packet configurations, though it did not address security implications or policy enforcement mechanisms.

These studies have provided valuable insights into basic network capabilities of CNIs, but demonstrate a critical limitation: the lack of comprehensive security analysis, particularly in policy processing and enforcement capabilities. While these studies thoroughly evaluate basic performance metrics, they do not address how architectural characteristics affect security features and policy processing, which are increasingly crucial in modern cloud environments.

Our research addresses these issues by first analyzing the relationship between CNI security features and policy implementations, and then examining their architectural connections across different security and policy configurations. We also evaluate the performance implications of these relationships. This comprehensive approach provides a thorough understanding of CNI security features, policy implementations, and their architectural characteristics in cloud-native environments, with a particular focus on policy processing.

B. SECURITY-ORIENTED RESEARCH IN KUBERNETES

Prior research explicitly addressing the security features and network policy aspects of CNIs has been more limited compared to performance-focused studies. Qi et al. [65] provided an early analysis of CNI security features, evaluating basic encryption capabilities and policy enforcement mechanisms across various plugins. Their follow-up study [10] expanded this analysis to include more detailed examination of eBPF-based security features and policy processing overhead, though primarily focusing on L3/L4 implementations. Budigiri et al. [12] conducted a focused investigation of eBPF-based network policies in Kubernetes, analyzing both security implications and performance characteristics of Calico and Cilium. While their study provided valuable insights into eBPF-based policy enforcement, it was limited to L3/L4 policies and did not address the broader spectrum of security features available in modern CNIs.

Recent research has expanded beyond CNI security analysis to address the challenges of policy management and verification. Lee and Nam [15] introduced Kunerva, which leverages network logs to automatically discover minimal policy sets, integrating with Kubernetes Gatekeeper for policy validation. In the context of microservices, Li et al. [17] developed AutoArmor, focusing on automated generation of inter-service access control policies through code-based analysis of service interactions.

Moreover, recent studies are considering AI-based approaches like Jacobs et al. [18], which uses natural language processing for intent-based network management. Li et al. [66] proposed Kano, a framework for efficient verification of cloud-native network policies using simulation-based validation and learned models to ensure policy correctness and security compliance. In this way, interest is growing in utilizing machine learning,

TABLE 4. Comparison of CNI research approaches.

References	Environment	Focus	Analyzed Layer	Security Features	Policy Analysis	Policy-related Performance	Performance Metrics
Kang et al. [63]	Kubernetes (Edge Cloud)	Performance Comparison	L3, L4	IPSec Overhead	None	None	Throughput, Latency, CPU
Sekigawa et al. [64]	Kubernetes (Telecom)	Performance Comparison	L3	Overlay/Underlay	None	None	Latency, Packet Loss
Dakić et al. [13]	Kubernetes (HPC + AI)	Performance Tuning Analysis	L3, L4	None	None	None	Throughput, Latency
Qi et al. [65]	Kubernetes	Design and Performance Analysis	L3, L4	Encryption, Limited Monitoring	Netfilter, iptables	Limited	Pod Startup, Throughput, Latency
Qi et al. [10]	Kubernetes	Design and Performance Analysis	L3, L4	Encryption, Monitoring	iptables, eBPF	Limited	Latency, Startup, Scalability
Budigiri et al. [12]	Kubernetes (5G Edge)	eBPF-based Policy Evaluation	L3, L4	eBPF, WireGuard	eBPF policies	Yes	Latency, Resources
This Work	Kubernetes (Cloud-native)	Comprehensive Analysis (Design, Security, Policy)	L3, L4, L7	Encryption, Multi-tenancy, Monitoring	Network Policy comparison, priority	Yes	Throughput, Latency, Resources, Scalability

natural language techniques, and even Large Language Models (LLMs) to automate and streamline network policy enforcement.

The emergence of these studies demonstrates growing interest in security policy management, yet existing analytical studies have two key limitations. First, they either briefly mention policy support or lack detailed analysis of policy mechanisms. Second, when such analysis exists, it primarily focuses on L3/L4 implementation technologies like eBPF and iptables, failing to examine the comprehensive network policies provided by modern CNIs.

This research addresses these gaps by providing an in-depth analysis of policy implementation mechanisms and processing methods across all network layers (L3/L4/L7). Our study examines previously unexplored aspects of CNI policy features, including policy priority systems, L7 protocol handling. This comprehensive analysis of policy establishes a foundation for future research in policy automation and verification in cloud-native environments.

VII. DISCUSSION

The experimental results of this study demonstrate that CNI selection is a complex decision-making process that goes beyond simple performance or functionality comparisons, requiring a comprehensive consideration of security requirements and operational environment characteristics.

A. SECURITY, PERFORMANCE, AND SCALABILITY TRADE-OFFS

In cloud-native environments, balancing security, performance, and scalability is critical when selecting a CNI. Each organization must carefully evaluate 'What is the necessary level of security, and what is the acceptable range of performance degradation?.' Our analysis reveals the intricate trade-offs among these elements.

Regarding the relationship between advanced security features and performance, as observed in L7 policy processing, the more sophisticated security features are applied, the more complex the packet inspection becomes, inevitably leading to processing overhead. The difference in L7 implementation between Cilium and Antrea particularly illustrates this trade-off. While Antrea secured efficiency through HTTP protocol-specific processing, Cilium takes a generic approach to support various L7 protocols, resulting in relatively significant performance degradation.

There was also a clear trade-off between policy scalability and processing performance. The eBPF-based Cilium maintained stable performance even as the number of policies increased through kernel-level optimization. In contrast, the OVS-based Antrea experienced a sharp drop in performance due to accumulated flow table search overhead as policy complexity increased. This demonstrates that the policy processing architecture has a direct impact on scalability.

Differences in concurrent connection scalability were also noticeable between the CNI architectures. iptables-based CNIs experienced dramatic performance degradation as concurrent connections increased due to the capacity limitations of the connection tracking table. While the Kube router using IPVS maintained stable performance up to 100 pods, it experienced performance degradation beyond 200 pods due to connection tracking limitations. These differences suggest that the efficiency of connection management mechanisms is a critical consideration in large-scale environments.

These trade-off patterns suggest that three key elements must be considered when selecting a CNI: the balance between comprehensiveness of security features and performance, processing efficiency with policy scaling, and stability in handling large numbers of connections. These elements are closely interrelated, and improving one will inevitably require compromising the others. In particular, as more advanced security features are applied, processing performance decreases, and as greater scalability is pursued, security feature limitations follow. Selection therefore requires careful consideration of the complementary relationship between these three elements.

B. PRACTICAL GUIDELINES FOR CNI SELECTION

Flannel was included in the baseline analysis of this study, but excluded from the performance evaluation. While Flannel provides only basic network connectivity and lacks security features or policy support, it is often used as the default CNI in small clusters that start their Kubernetes with a minimal environment. This serves as an important reference point for understanding functional differences when transitioning to other CNIs for advanced security features.

For throughput-sensitive microservice environments, Cilium may be the optimal choice. It maintains a high throughput of 8.9K Mbps at L3/L4 levels, with particularly excellent performance in single-node environments. However, performance can drop to 94Mbps when L7 policies are applied, so it is advisable to selectively apply L7 security only to specific critical services. For environments with predominantly HTTP-based applications, Antrea can be considered. With HTTP-specific L7 policy processing, it maintains a high throughput of 6.6K Mbps. However, Antrea's performance degradation due to sequential OVS flow table scans as policy size increases must be considered.

For complex microservice architectures that require granular traffic analysis, Cilium's Hubble provides valuable traffic analysis that can detect subtle security anomalies in microservice communications. However, these advanced monitoring features can generate CPU overhead, so storage capacity for monitoring data and log management strategies must be carefully considered.

In selecting CNIs for multi-tenancy environments, four criteria warrant consideration. Firstly, the level of isolation must be evaluated in order to ascertain whether tenant traffic is completely segregated or potentially exposed. Secondly, the support for inter-tenant encryption must be verified, which is of particular importance in environments that handle sensitive data. Finally, the support of authentication and authorization must be examined in order to ensure compatibility with the relevant policies, whether those are RBAC or identity-based approaches. In accordance with the aforementioned criteria, environments requiring high-level isolation and granular policy control are optimally served by Cilium or Calico. In addition, scenarios that require basic isolation may find Antrea or WeaveNet to be sufficient.

If you need to choose a CNI based on granular network security control requirements, each CNI's policy offer advantages. For basic security isolation, Kubernetes Network Policy provides essential L3/L4 controls with namespace and pod selectors, suitable for environments with straightforward needs. For environments needing various protocol-level traffic control, Cilium Network Policy provides extensive coverage across HTTP, gRPC, and Kafka, along with endpoint targeting. Where infrastructure-aware security controls are crucial, Calico Network Policy combines IPAM integration with flexible CIDR pattern matching, enabling sophisticated subnet and IP pool-based rules. Additionally, Calico and Antrea provide diverse action types, enabling detailed security monitoring and staged policy rollouts.

In environments with high policy management complexity, policy prioritization systems should be considered. Calico's single-tier approach is appropriate for medium policy management, while Antrea's three-tier hierarchical system is appropriate for large deployments with more complex policy requirements. In environments that require a large number of concurrent connections, connection stability must be carefully considered. Cilium maintained a connection success rate of approximately 90% at 200 concurrent connections, while other CNIs showed significant connection instability. Therefore, Cilium may be a better choice in environments that require a high number of concurrent connections.

In resource-constrained environments, WeaveNet or Kube-router may be reasonable alternatives. WeaveNet provides a few security features, but shows significant performance degradation as the number of policies and pods increases, making it suitable for simple environments. Kube-router, while more basic in functionality, maintains stable performance by leveraging IPVS technology, making it effective in high-traffic environments despite its simplicity.

Finally, organizations considering a move to multi-cloud environments should choose platform-neutral CNIs to ensure workload mobility and consistent application of security policies. In this context, Cilium with its ClusterMesh feature or Calico with its Federation feature would be appropriate choices as they provide unified policy management and seamless service connectivity across different cloud platforms while maintaining consistent security controls.

As a result, we recommend that the default CNIs from major cloud providers are not always the best choice in all situations. While the AWS VPC CNI, Azure CNI, and Google Cloud VPC-native are optimized for their respective platforms, combining or replacing them with alternative CNIs may be more appropriate depending on an organization's specific security requirements or operational patterns. Therefore, it is advisable to carefully analyze and consider which CNI and policy configuration best suits your needs.

VIII. CONCLUSION

In this study, we present an in-depth analysis of network policy mechanisms in Kubernetes CNIs, focusing particularly on the intricate relationships between architectural design choices, security policy features, and performance implications. While earlier research primarily focused on isolated performance metrics, our study uniquely demonstrates how architectural decisions—from Cilium's eBPF implementation to Antrea's OVS pipeline and Calico's BGP routing—fundamentally influence policy processing capabilities and enforcement patterns. Through both qualitative and quantitative analysis, we establish clear correlations between these three aspects, providing a comprehensive understanding of their interdependencies.

This comprehensive approach revealed critical insights into policy processing mechanisms and their performance implications in complex environments. Notably, we found that eBPF-based solutions experience significant

performance degradation during L7 policy enforcement due to kernel-userspace transitions, while overlay network CNIs face connection stability challenges under high policy complexity. We believe our findings provide essential foundations for both practitioners selecting CNIs and researchers advancing container network security.

Several limitations in our current study inform potential research directions. First, our evaluation environment could be expanded to include physical network infrastructure to validate performance impacts under real-world conditions, particularly examining phenomena like packet delays and losses. Second, our analysis could be extended to examine CNI behavior and policy consistency during failure scenarios and network interruptions. Third, exploring policy enforcement consistency across hybrid and multi-cloud environments would provide valuable insights for modern distributed architectures.

In addition, future research extending from this work includes: (1) leveraging machine learning techniques for dynamic policy optimization based on traffic patterns and threat detection; (2) developing automated approaches for policy generation and verification that can handle complex microservice interactions while maintaining performance; and (3) investigating novel approaches to achieve both comprehensive L7 security and high performance.

REFERENCES

[1] S. Deng, H. Zhao, B. Huang, C. Zhang, F. Chen, Y. Deng, J. Yin, S. Dustdar, and A. Y. Zomaya, "Cloud-native computing: A survey from the perspective of services," *Proc. IEEE*, vol. 112, no. 1, pp. 12–46, Jan. 2023.

[2] L. Patan, "Leveraging cloud-native architecture for scalable and resilient enterprise applications: A comprehensive analysis," *Int. J. Comput. Eng. Technol. (IJCET)*, vol. 15, no. 5, pp. 583–591, 2024.

[3] M. K. Sasubilli and R. Venkateswarlu, "Cloud computing security challenges, threats and vulnerabilities," in *Proc. 6th Int. Conf. Inventive Comput. Technol. (ICICT)*, Jan. 2021, pp. 476–480.

[4] R. Bundela, N. Dhanda, and K. K. Gupta, "Identification and analysis of security issues in cloud computing," in *Proc. 2nd Int. Conf. Disruptive Technol. (ICDT)*, Mar. 2024, pp. 1685–1690.

[5] Algosec. (2024). *The 2024 State of Network Security Report Reveals a Shift Towards Multi-cloud Environments, With a 47% Increase in Sd-wan and 25% Uptick in Sase Adoption*. [Online]. Available: https://www.algosec.com/press-release/the-2024-state-of-network-security-report

[6] (2024). *The Container Network Interface*. [Online]. Available: https://www.cni.dev/

[7] Kubernetes. (2024). *Kubernetes*. [Online]. Available: https://kubernetes.io/

[8] N. Kapočius, "Overview of kubernetes cni plugins performance," *Mokslas–Lietuvos ateitis/Science–Future Lithuania*, vol. 12, 2020.

[9] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman, "XI commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices," in *Proc. IEEE Secure Develop. (SecDev)*, Sep. 2020, pp. 58–64.

[10] S. Qi, S. G. Kulkarni, and K. K. Ramakrishnan, "Assessing container network interface plugins: Functionality, performance, and scalability," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 656–671, Mar. 2021.

[11] S. Böhm and G. Wirtz, "Profiling lightweight container platforms: MicroK8s and K3s in comparison to Kubernetes," in *Proc. ZEUS*, Jan. 2021, pp. 65–73.

[12] G. Budigiri, C. Baumann, J. T. Mühlberg, E. Truyen, and W. Joosen, "Network policies in kubernetes: Performance evaluation and security analysis," in *Proc. Joint Eur. Conf. Netw. Commun. 6G Summit (EuCNC/6G Summit)*, vol. 2021, Jun. 2021, pp. 407–412.

[13] V. Dakić, J. Redžepagić, M. Bašić, and L. Žgrablić, "Performance and latency efficiency evaluation of kubernetes container network interfaces for built-in and custom tuned profiles," *Electronics*, vol. 13, no. 19, p. 3972, Oct. 2024.

[14] K. Dzeparoska, J. Lin, A. Tizghadam, and A. Leon-Garcia, "LLM-based policy generation for intent-based management of applications," in *Proc. 19th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2023, pp. 1–7.

[15] S. Lee and J. Nam, "Kunerva: Automated network policy discovery framework for containers," *IEEE Access*, vol. 11, pp. 95616–95631, 2023.

[16] S. Xu, Q. Zhou, H. Huang, X. Jia, H. Du, Y. Chen, and Y. Xie, "Log2Policy: An approach to generate fine-grained access control rules for microservices from scratch," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2023, pp. 229–240.

[17] X. Li, Y. Chen, Z. Lin, X. Wang, and J. H. Chen, "Automatic policy generation for inter-service access control of microservices," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 3971–3988.

[18] A. S. Jacobs, R. J. Pfitscher, R. H. Ribeiro, R. A. Ferreira, L. Z. Granville, W. Willinger, and S. G. Rao, "Hey, lumi! Using natural language for intent-based network management," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC 21)*, 2021, pp. 625–639.

[19] Flannel. (2024). *Flannel is a Network Fabric for Containers, Designed for Kubernetes*. [Online]. Available: https://github.com/flannel-io/flannel

[20] Weave. (2024). *Weave Scope—Troubleshooting & Monitoring for Docker & Kubernetes*. [Online]. Available: https://github.com/rajch/weave

[21] Tigera. (2024). *Project Calico*. [Online]. Available: https://www.tigera.io/project-calico/

[22] CloudNativeLabs. (2024). *Kube-router, a Turnkey Solution for Kubernetes Networking*. [Online]. Available: https://github.com/cloudnativelabs/kube-router

[23] Cilium. (2024). *Cilium—Ebpf-based Networking, Observability, Security*. [Online]. Available: https://cilium.io/

[24] Antrea. (2024). *Antrea—Kubernetes Networking Based on Open Vswitch*. [Online]. Available: https://github.com/antrea-io/antrea

[25] A. Bhardwaj and C. R. Krishna, "Virtualization in cloud computing: Moving from hypervisor to containerization—A survey," *Arabian J. Sci. Eng.*, vol. 46, no. 9, pp. 8585–8601, Sep. 2021.

[26] CNCF. (2024). *CNCF Annual Survey—Cloud Native 2023: The Undisputed Infrastructure of Global Technology*. [Online]. Available: https://www.cncf.io/reports/cncf-annual-survey-2023/

[27] Fortinet. (2024). *2024 Cloud Security Report*. [Online]. Available: https://global.fortinet.com/lp-en-2024-cloud-report?utm_source=Social&utm_medium=Blog&utm_campaign=Cloud-NAMER-U.S.&utm_content=AR-2024cloudsecreport-G&utm_term=Blog&lsci=701Hr0000011kADIAY&UID=ftnt-4032-225332

[28] RedHat. (2024). *The State of Kubernetes Security Report: 2024 Edition*. [Online]. Available: https://www.redhat.com/en/engage/state-kubernetes-security-report-2024

[29] Sysdig. (2024). *2024 Cloud-native Security and Usage Report*. [Online]. Available: https://sysdig.com/2024-cloud-native-security-and-usage-report/

[30] T. Muhammad, "Overlay network technologies in SDN: Evaluating performance and scalability of VXLAN and GENEVE," *Int. J. Comput. Sci. Technol.*, vol. 5, no. 1, pp. 39–75, 2021.

[31] A. Mehdizadeha, K. Suinggia, M. Mohammadpoorb, and H. Haruna, "Virtual local area network (VLAN): Segmentation and security," in *Proc. 3rd Int. Conf. Comput. Technol. Inf. Manage. (ICCTIM)*, vol. 78, 2017, p. 89.

[32] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks," Tech. Rep., 2014.

[33] H. Liu, Y. Luo, B. Chen, and Y. Yang, "Performance evaluation of container networking technology," in *Proc. IEEE 3rd Int. Conf. Inf. Technol., Big Data Artif. Intell. (ICIBA)*, vol. 3, May 2023, pp. 815–818.

[34] L. Song and J. Li, "EBPF: Pioneering kernel programmability and system observability—Past, present, and future insights," in *Proc. 3rd Int. Conf. Artif. Intell. Comput. Inf. Technol. (AICIT)*, Sep. 2024, pp. 1–10.

[35] H. Sharaf, I. Ahmad, and T. Dimitriou, "Extended Berkeley packet filter: An application perspective," *IEEE Access*, vol. 10, pp. 126370–126393, 2022.

[36] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior, and L. F. M. Vieira, "Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications," *ACM Comput. Surveys*, vol. 53, no. 1, pp. 1–36, Jan. 2021.
[37] Tigera. (2024). *Calico Cloud Documentation—Federation and Multi-cluster Networking*. [Online]. Available: https://docs.tigera.io/calico-cloud/multicluster/
[38] Cilium. (2024). *Cilium—Setting Up Cluster Mesh*. [Online]. Available: https://docs.cilium.io/en/stable/network/clustermesh/clustermesh/
[39] L. Foundation. (2024). *Production Quality, Multilayer Open Virtual Switch*. [Online]. Available: https://www.openvswitch.org/
[40] Octant. (2024). *Visualize Your Kubernetes Workloads*. [Online]. Available: https://octant.dev/
[41] Kubernetes. (2024). *Kubernetes —Custom Resources*. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
[42] Antrea. (2024). *Antrea Multi-cluster User Guide*. [Online]. Available: https://antrea.io/docs/v1.7.2/docs/multicluster/user-guide/
[43] (2024). *Network Policies*. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/network-policies/
[44] Cilium. (2024). *Hubble—Network, Service & Security Observability for Kubernetes Using Ebpf*. [Online]. Available: https://github.com/cilium/hubble
[45] Prometheus. (2024). *Prometheus—From Metrics to Insight*. [Online]. Available: https://prometheus.io/
[46] Tigera. (2024). *Calico Cloud Documentation—Flow Log Data Types*. [Online]. Available: https://docs.tigera.io/calico-cloud/visibility/elastic/flow/datatypes
[47] Elastic. (2024). *Elasticsearch*. [Online]. Available: https://www.elastic.co/elasticsearch
[48] CISCO. (2024). *Splunk—Fortune Favors the Resilient*. [Online]. Available: https://www.splunk.com/
[49] Antrea. (2024). *Network Flow Visibility in Antrea*. [Online]. Available: https://antrea.io/docs/v1.0.0/docs/network-flow-visibility/
[50] Elastic. (2024). *Elastic Stack—Meet the Search Platform that Helps You Search, Solve, and Succeed*. [Online]. Available: https://www.elastic.co/elasticsearch
[51] G. Labs. (2024). *Grafana: The Open Observability Platform*. [Online]. Available: https://grafana.com/
[52] Weave. (2024). *Weave Scope—-Monitoring, Visualisation & Management for Docker & Kubernetes*. [Online]. Available: https://github.com/weaveworks/scope
[53] (2024). *Common Expression Language in Kubernetes*. [Online]. Available: https://kubernetes.io/docs/reference/using-api/cel/
[54] Tigera. (2024). *Ip Pool—Calico Documentation—Tigera*. [Online]. Available: https://docs.tigera.io/calico/latest/reference/resources/ippool
[55] Antrea. (2024). *Antrea API Reference*. [Online]. Available: https://antrea.io/docs/main/docs/api-reference/
[56] Tigera. (2024). *Global Network Policy*. [Online]. Available: https://docs.tigera.io/calico/latest/reference/resources/globalnetworkpolicy
[57] Cilium. (2024). *Ciliumnetworkpolicy*. [Online]. Available: https://docs.cilium.io/en/stable/network/kubernetes/policy/
[58] Antrea. (2024). *Antrea Network Policy Crds*. [Online]. Available: https://antrea.io/docs/main/docs/antrea-network-policy/
[59] O. Olimov, G. Artikova, and M. Xatamova, "Iperf to determine network speed and functionality," *Web Technol., Multidimensional Res. J.*, vol. 2, no. 3, pp. 94–101, 2024.
[60] R. Hat. (2024). *2.5.4.4. the Sar Command*. [Online]. Available: https://docs.redhat.com/ko/documentation/red_hat_enterprise_linux/4/html/introduction_to_system_administration/s3-resource-tools-sar-sar#s3-resource-tools-sar-sar
[61] APACHE. (2024). *Ab—Apache Http Server Benchmarking Tool*. [Online]. Available: https://httpd.apache.org/docs/2.4/en/programs/ab.html
[62] Suricata. (2024). *Suricata—Observe. Protect. Adapt*. [Online]. Available: https://suricata.io/
[63] Z. Kang, K. An, A. Gokhale, and P. Pazandak, "A comprehensive performance evaluation of different kubernetes CNI plugins for edge-based and containerized publish/subscribe applications," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Oct. 2021, pp. 31–42.
[64] S. Sekigawa, C. Sasaki, and A. Tagami, "Toward a cloud-native telecom infrastructure: Analysis and evaluations of kubernetes networking," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2022, pp. 838–843.
[65] S. Qi, S. G. Kulkarni, and K. K. Ramakrishnan, "Understanding container network interface plugins: Design considerations and performance," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN*, Jul. 2020, pp. 1–6.
[66] Y. Li, X. Hu, C. Jia, K. Wang, and J. Li, "Kano: Efficient cloud native network policy verification," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 3, pp. 3747–3764, Mar. 2022.

BOM KIM is currently pursuing the M.S. degree with the Department of Computer Science and Engineering, Incheon National University. Her research focuses on the automation of intent-driven security policy generation and validation for advanced cloud-native environments.

JINWOO KIM received the B.S. degree in computer science and engineering from Chungnam National University, the M.S. degree from the Graduate School of Information Security, KAIST, and the Ph.D. degree from the School of Electrical Engineering, KAIST. He is currently an Assistant Professor with the School of Software, Kwangwoon University, Seoul, South Korea. His research interests include investigating security issues with software-defined networks and cloud systems.

SEUNGSOO LEE received the B.S. degree in computer science from Soongsil University, the M.S. degree in information security from KAIST, and the Ph.D. degree in information security from KAIST, in 2020. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Incheon National University. His research interests include developing secure and robust cloud/network systems against potential threats.