The Journal of Korean Institute of Communications and Information Sciences '25-07 Vol.50 No.07
https://doi.org/10.7840/kics.2025.50.7.1085

Unveiling Kubernetes CNI: An In-Depth Analysis of Networking Performance and Resource Efficiency

Wonmi Choi◆, Juyoung Ahn*, Yeonho Yoo*, Zhixiong Niu**, Gyeongsik Yang°, Chuck Yoo°°

ABSTRACT

Kubernetes relies heavily on its networking performance. It offers four representative networking plugins: Flannel, Calico, Cilium, and Kube-router. However, their performance differences are not well understood. This study evaluates these plugins using real-world workloads like Memcached, Nginx and Kafka, examining throughput, latency, and CPU usage in 10 GbE and 100 GbE environments. Results reveal significant performance differences due to each plugin's architecture. Kube-router excels in CPU- and network-intensive scenarios but complicates network management. Among overlay plugins Flannel performs best in CPU-intensive tasks, while Cilium is superior for network-intensive tasks. This analysis provides insights for selecting suitable plugin based on workload characteristics.

Key Words : Cloud networking, Performance profiling, Container network interface, CPU usage, Kubernetes

Ⅰ. Introduction

Kubernetes is the de-facto container orchestrator for cloud services by automating deployment, scaling, and management of containerized applications across clusters[1]. Cloud services[2,3], like distributed deep learning and blockchain platforms, rely on Kubernetes networking. Containers in Kubernetes are deployed as pods, and multiple pods communicate heavily among themselves for these cloud services. Therefore, the network performance between pods (containers) significantly impacts the user experience, directly determining the quality of services[4-6].

The networking configuration of containers is managed by the container network interface (CNI) plugin, which performs two key tasks: 1) creation and deletion of container network components (e.g., veth pairs for connecting containers and hosts) and 2) assignment of unique IP addresses to containers. CNI plugins automate these configurations to enhance the scalability and flexibility of container-based services. There are four widely-used open source CNI plugins: Flannel, Calico, Cilium, and Kube-router. Cloud services can select a CNI plugin for their services, and because of the internal architecture differences of plugins, the network performance varies upon the CNI plugin selection[7]. Thus, it is important to understand the performance of plugins for better service quality.

Previous studies have compared the performance of CNI plugins. However, they are quite limited in

※ This work was partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2021R1A6A1A13044830), by the NRF grant funded by the Korea government (MSIT) (NRF-2023R1A2C3004145, RS-2024-00336564), by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (RS-2024-00405128), by ICT Creative Consilience Program (IITP-2025-RS-2020-II201819), and by the Google Cloud Research Credits program.

◆ First Author : Korea University Department of Computer Science and Engineering, ymcui@os.korea.ac.kr, 학생회원
° Corresponding Author : Korea University Department of Computer Science and Engineering, g_yang@korea.ac.kr, 정회원
°° Corresponding Author : Korea University Department of Computer Science and Engineering, hxy@korea.ac.kr, 종신회원
* Korea University Department of Computer Science and Engineering, jyahn@os.korea.ac.kr; yhyoo@os.korea.ac.kr, 정회원
** Microsoft Research Asia, zhniu@microsoft.com

논문번호 : 202412-313-C-RU, Received December 9, 2024; Revised January 9, 2025; Accepted January 27, 2025

the following aspects. First, the CNI plugins have different internal architectures[8-12], but it has not been analyzed how such differences affect the network performance (e.g., throughput and latency) and resource usage (e.g., CPU usage). Second, the evaluations of CNI plugins[13-15] only cover micro-benchmarks such as netperf or iperf that perform iterative packet transmissions but do not reflect the realistic aspects of realworld workloads. As containers today are used for a wide range of workloads, from network-intensive to CPU-intensive ones, it is essential that evaluations reflect this diversity of workloads. Third, existing studies[7,13-18] mostly focus on analyzing the throughput and latency or give only a rough summary of CPU overhead. They failed to comprehend the correlation between kernel-level CPU usage and network throughput to identify the root causes of network performance differences between CNI plugins.

This paper aims to provide a comprehensive characterization of CNI plugins, overcoming the limitations of previous studies. We first detail the architecture and internal dataflow of each CNI plugin (§II). In particular, we focus on the packet transmission path because it is the primary challenge for timely and responsive cloud services. For example, web servers of Google, Baidu, Yahoo, and Microsoft Bing serve answers to hundreds of millions of user requests each day[19], which rely heavily on packet transmission.

Second, we employ Memcached, Nginx and Kafka as real-world workloads for the performance analysis (§III). These workloads are chosen due to their popularity[20-22], for the following reasons. Cloud services are distributed systems and typically have a distributed storage for storing data, a web server to send and deliver required data to users and a synchronization service for the containers to maintain consistent states. To reflect these components, we choose Memcached, Nginx and Kafka, which address the three components mentioned above. These workloads exhibit distinct network traffic and respective resource usage patterns, but to our knowledge, the behavior with CNI plugins has not been investigated yet.

Third, existing studies measure the performance with ~25 GbE[7,16]. However, current datacenters are already equipped with servers in Kubernetes clusters that communicate at up to 100 GbE[23]. This means that we cannot determine if the analysis from previous studies is applicable to higher bandwidths. Therefore, we conduct the extensive measurement of CNI plugins in both 10 GbE and 100 GbE NICs (§IV).

Specifically, we analyze the network performance differences between the plugins in terms of network throughput, CPU usage, and latency. In addition, we perform detailed profiling of CPU usage for packet processing (*Softirq*) to identify the root causes of performance differences depending on workload characteristics. Through the comparative analysis between plugins, this study provides insights into choosing the best possible plugin depending on the characteristics of the workloads. We make the following novelties:

- First in-depth **structural analysis of four CNI plugins**, which offers guidelines to understand CNI performance.
- Analysis using three **real-world applications** for both 10 GbE and 100 GbE settings.
- Extensive experiments and **kernel-level investigations** to investigate performance differences and assess CPU resource efficiency, which provides meaningful insights for cloud service deployment.

Ⅱ. Background

2.1 Kubernetes

A Kubernetes cluster consists of nodes that are classified into worker or master. The worker node runs pods, the basic container deployment units in Kubernetes. A pod consists of a set of containers. The containers of a pod share the network and storage resources with the others belonging to the same pod. For simplicity, we assume that one pod maintains one container.

The master node runs control plane, such as 1) etcd, 2) kube-apiserver, 3) kube-scheduler, and 4) kube-proxy. etcd is a key-value store to save container network configurations and IP addresses. Containers can retrieve the etcd data through APIs provided by kubeapiserver. Kube-scheduler receives container creation requests from users via kube-apiserver and assigns the newly created containers to appropriate worker nodes. When a Kubernetes cluster is formed, kube-proxy creates new iptables to manage Kubernetes traffic.

Each node in the cluster runs a kubelet that is responsible for initiating and running containers. Also, each node starts a CNI daemon that interacts with the kubeapiserver to allocate IP addresses to each container and create network interfaces per container. Containers communicate using the network interfaces configured by the CNI daemon.

2.2 Packet processing in Kubernetes cluster

The packet processing of containers can be divided into *User*, *System*, and *Softirq* contexts. First, the *User* context is in charge of executing the application that runs within the container. Second, when the application executes a system call (e.g., *sendmsg* to transmit a network packet), the *System* context handles the system call in the kernel space. At last, the *Softirq* context supports asynchronous packet delivery between different network interfaces (e.g., *VXLAN* and *eth*) and virtual interfaces (e.g., *veth*). This is because the container networking in a Kubernetes cluster consists of multiple network interfaces. When a packet is delivered between the interfaces, a software interrupt (NET_RX_SOFTIRQ) is raised and handled by the *Softirq* handler (NET_RX_ACTION) that processes the packet through the network protocol stack. The detailed operations of the *Softirq* contexts vary depending on the type of CNI plugins, which are explained in the next subsection.

2.3 CNI Plugin Comparison

Fig. 1 shows the detailed architectures of CNI plugins. The solid line represents the packet processing workflow in the *Softirq* context. The green boxes are networking method with interfaces. The dashed line indicates the use of table structures during the workflow. We explain the details below.

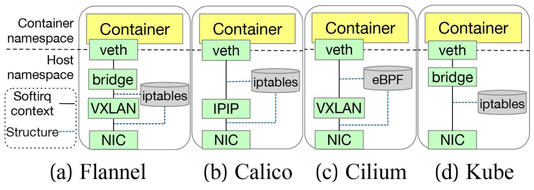


Fig. 1. CNI plugin architectures.

Networking method. We explain the networking method of *Softirq* by green boxes in Fig. 1. Packets from the container go to veth in the host namespace. For Flannel (Fig. 1a), packets are delivered to the Linux bridge. When the bridge receives the packets, it forwards packets to VXLAN that encapsulates packets with Layer-2 (L2, Ethernet) headers for overlay networking. The packets are then forwarded to the NIC, which finally transmits them out of the host machine. For Calico (Fig. 1b), the packets received at veth are forwarded to IPIP. Unlike VXLAN of L2 overlay, IPIP provides L3-based overlay by encapsulating packets with IP headers. Cilium (Fig. 1c) has a similar workflow of Calico but uses VXLAN instead of IPIP. In summary, Flannel, Calico, and Cilium employ overlay networking (e.g., VXLAN and IPIP). An advantage of the overlay is that it provides flexibility in security by enforcing security policies on containers that communicate within the same overlay network.

However, these methods may impact network performance due to the overhead of performing encapsulation. Moreover, it is known that overlay-based plugins of VXLAN and IPIP cannot utilize TCP segmentation offload (TSO) functionality of native Linux, which may further reduce performance[24].

In contrast, Kube-router (Fig. 1d) uses underlay networking. When packets arrive veth, Kube-router delivers the packets to the bridge and then directly forwards packets to the NIC by the container IP address without overlay. Kube-router can forward packets without overlay because it exchanges routing information of containers through BGP. Also, Kuberouter can leverage the TSO feature in Linux, which offloads packet segmentation to the NIC. Although Kube-router avoids overlay, it requires the

BGP processing for adding or modifying routing information. These frequent changes are known to possibly disrupt packet forwarding due to the outdated routing information not being updated promptly[25].

Packet forwarding: The gray cylinders in Fig. 1 are table structures for packet forwarding. Flannel, Calico, and Kube-router forward packets using Linux iptables. Specifically, each CNI plugin performs multiple lookups in iptables (e.g., *mangle*, *nat*, and *filter*) to carry out packet processing, such as time-to-live and type-of-service configurations, network address changes, and packet filtering. The iptables include processing policies that are registered at five hooks: *prerouting*, *input*, *output*, *forward*, and *post-routing*. On the other hand, Cilium forwards packets using the eBPF program by registering the packet processing policies into the eBPF map. Each network interface uses an eBPF program that finds the proper policy from the eBPF map instead of iptables, allowing Cilium to avoid traversing the IP layer, unlike other plugins.

CNI plugins also offer different levels of packet filtering by iptables or eBPF map. Calico provides filtering configurations for both ingress and egress traffic by iptables rules. Cilium implements filtering configurations for ingress and egress traffic using eBPF programs. On the other hand, Kube-router only supports the ingress traffic control by iptables rules. Flannel does not support any packet filtering configurations, meaning that it cannot filter any traffic for containers.

Ⅲ. Workloads

We select Memcached and Kafka to assess the network plugins. Table 1 presents the characteristics of the workloads.

Memcached is a key-value store mainly used to boost web applications by caching data from databases or remote storage into memory[20]. Memcached communicates with clients by its own binary protocol that creates TCP packets with operations (SET and GET) and key-value data. In our experiments, Memcached presents nearly 100% CPU usage while network bandwidth is only 22% utilized

Table 1. Characteristics of workloads.

Workload	Protocol	Major payload size	Resource usage		Primary resource
			CPU (%)	Network (%)	
Memcached	Memcached binary	key : 64 B, value : 1024 B	99	22	CPU
Nginx	HTTP	1 KB	96	12	CPU
		1 MB	9	90	Network
Kafka	Kafka wire	1 KB	82	34	CPU
		1 MB	21	91	Network

(experiment setup to be explained in §IV). This reveals that the networking performance of Memcached is limited by the CPU, meaning that Memcached is CPU-intensive, which aligns with Yoann *et al.*[26].

Nginx is a web server that transmits HTML, CSS, and Javascript pages to web browsers[21]. Unlike Memcached's binary protocol, Nginx uses HTTP to interact with clients. When it receives an HTTP request, Nginx generates a response containing a status code (success or failure of the request), metadata headers, and the requested file. For resource usage, we test 1 KB and 1 MB payload sizes because the web server processes payloads from small to large. With 1 KB payloads, CPU usage is at 96%, while network bandwidth utilization is only 12%. For 1 MB payloads, CPU usage is only 9% while network utilization is 90%. The results show that the major resource consumption of Nginx differs per payload size—for small payloads (1 KB), it is CPU-intensive, and for large payloads (1 MB), it is network-intensive.

Kafka is a publish-subscribe messaging system that handles real-time message collection and provision[22]. It consists of producer, broker, and consumer. A producer sends requests to one or more brokers to store new messages. A broker stores the messages, and a consumer then sends requests to brokers to receive messages. Kafka uses its wire protocol to create TCP packets for operations (e.g., produce, fetch, create, and delete). For resource usage, we test small (1 KB) and large (1 MB) as Kafka handles various sizes. For 1 KB payloads, CPU and network usage are 82% and 34%, each. For 1 MB payloads, they are 21% and 91%. Thus, we can say Kafka is

CPU-intensive for small payloads and network-intensive for large payloads.

Ⅳ. Experiment Settings and Results

This section shows the results of extensive experiments. Our experiments evaluate the network throughput, CPU usage, and latency of the network plugins in different environments. We first explain the analysis method and then the detailed results.

4.1 Analysis Methodology

4.1.1 Testbed

We use two network settings: 1) 10 GbE servers from our on-premise clusters and 2) 100 GbE servers from CloudLab[27]. Table 2 summarizes specifications. The software versions are the latest ones that can run on each server and testbed.

10 GbE servers. We use two servers, each running a master node and a worker node of the Kubernetes cluster, similar to previous studies[13,14]. The master node also runs load generators (e.g., generating cache requests to Memcached servers). The worker node runs two containers for Memcached, Nginx or Kafka. All servers are connected by 10 GbE with receive flow steering (RFS) of NICs enabled.

100 GbE servers. We use 100 GbE NIC servers from CloudLab[27]. The increased network bandwidth causes the load generator to suffer from CPU shortage with a single server, so we add one more server for the load generator. Thus, we use three nodes for Mecached, Nginx and Kafka. All servers are connected by 100 GbE with RFS enabled.

Table 2. Specification of the servers.

Specification		10 GbE servers	100 GbE servers
HW	CPU	One Intel Xeon E5-2650 v3 10-core CPU at 2.30GHz	One Intel Xeon Silver 4314 16-core CPU at 2.40 GHz
	RAM	256GB	128GB
SW	OS	Ubuntu 20.04, Linux kernel 5.4	
	Runtime	containerd 1.6.12, Kubernetes 1.27.2	containerd 1.6.32, Kubernetes 1.24.17
	Plugins	Flannel v.0.22.0, Calico v.3.26.1, Cilium v.1.13.0, Kube-router v.1.5.4	

4.1.2 Measurement Methodology

For each workload, we measure network throughput (MB/s) and latency (ms) by the widely-used benchmarks. We use Memaslap for Memcached, wrk[28] for Nginx servers, and kafka-consumer-perf-test[29] for Kafka, which are the official benchmarks for each workload. The benchmark configurations are set to generate a sufficient amount of load, and the results are explained in the following subsections. We also measure the CPU usage by mpstat, the de-facto benchmark in Linux. We conduct in-depth profiling of CPU usage, categorizing it into *User*, *System*, and *Softirq*. Since *Softirq* is a key CPU usage for packet processing, the *Softirq* results are presented. We repeat each experiment five times and plot the average values.

4.2 Memcached

First, we present the experiment results of the Memcached for network throughput, latency, and CPU usage. We run two containers executing the Memcached workloads on a worker node. Then, we execute two Memaslap instances on a server to generate requests to the Memcached containers in 10 GbE servers. For 100 GbE servers, we execute two Memaslap instances on two separate servers to create the sufficient amount of requests. Each Memaslap instance generates 10 million requests to Memcached containers for every experiment trial.

4.2.1 Network Throughput and Latency

Fig. 2 shows the results of Memcached throughput (left y-axis) and latency (right y-axis) with 10 GbE and 100 GbE servers. Kube-router, the underlay network-based plugin, provides higher throughput and lower latency than Flannel, Calico, and Cilium, that are all based on overlay networking. Specifically, in the 10 GbE servers, Kube-router outperforms Flannel, Calico, and Cilium with 22%, 42%, and 42% higher throughput. Also, Kube-router shows 18%, 29%, and 30% lower latency than the three. This trend is similar in 100 GbE servers—Kube-router achieves 23%, 46%, and 51% higher throughput and 18%, 30%, and 34% lower latency than the three, respectively. We find that

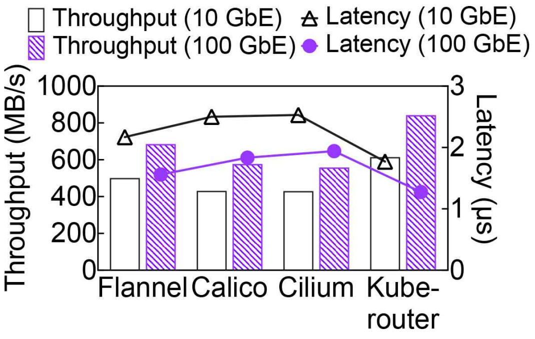


Fig. 2. Memcached throughput and latency.

this is because the overlay-based plugins require additional computations for packet en/de-capsulation.

Specifically, when a packet is transmitted through an overlay-based plugin, it needs to be delivered to the additional network interface (i.e., VXLAN of Flannel, IPIP of Calico, and VXLAN of Cilium) at the host kernel. The network interface then creates a new packet header with the host address and encapsulates the container's packet as the payload. The reverse operations are also needed at the destination, which decreases the network throughput of overlay-based plugins. On the other hand, Kube-router eliminates the need for packet en/decapsulation by utilizing the container's IP address directly to communicate with containers on different hosts. This reduces per-packet processing overhead and allows Kube-router to achieve higher network throughput than the overlay-based plugins.

Next, we analyze the results of overlay-based plugins. Flannel outperforms Calico in both 10 GbE and 100 GbE servers with 16% and 19% higher throughout and 13% and 15% lower latency, respectively. We find that the reason for the low throughput of Calico is due to the high number of iptables rules. Table 3 demonstrates that Calico generates three times more iptables rules than Flannel, resulting in high overhead. Calico has iptables rules required for packet filtering (both ingress and egress), while Flannel does not support any filtering; thus, the corresponding rules are not configured, which is why Flannel has much fewer iptables rules. As a result, Calico requires a long time to traverse iptables rules for per-packet processing and degrades the network

Table 3. Number of iptables rules.

	Flannel	Calico	Cilium	Kube-router
# of rules	56	141	-	99

throughput.

Also, Cilium uses the eBPF program for packet forwarding instead of iptables and does not configure any iptables rules. Although Cilium avoids the packet forwarding overhead from iptables, Fig. 2 shows that Cilium has lower network throughput (14% and 19% in 10 GbE and 100 GbE servers) and higher latency (16% and 24% in 10 GbE and 100 GbE servers) compared to Flannel. This is due to the additional eBPF programs attached to each network interface. Cilium filters ingress and egress traffic in those eBPF programs, resulting in higher packet forwarding overhead than Flannel.

4.2.2 CPU Usage

Fig. 3 depicts that plugins with higher throughput have lower *Softirq* in both 10 GbE and 100 GbE servers. For example, the underlay-based Kube-router shows the lowest *Softirq* CPU usage compared to the overlaybased plugins. The CPU usage of *Softirq* in Kuberouter is 25%, 29%, and 29% smaller than Flannel, Cilium, and Calico, respectively, in 10 GbE servers, and 9%, 17%, and 19% smaller, respectively, in 100 GbE servers. This indicates that the underlay-based plugin requires less CPU usage for *Softirq* processing than overlay-based plugins, as it saves additional packet en/de-capsulation. As a result, with the same number of CPUs (e.g., 8 cores in our experiments), the underlay-based plugin can spare

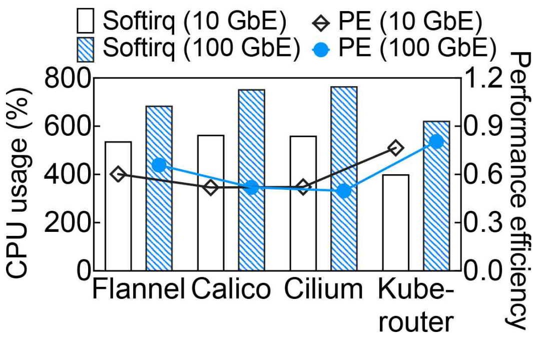


Fig. 3. *Softirq* CPU usage of Memcached containers.

more time to generate packets (e.g., *User* and *System*), which increases the network throughput compared to the overlay-based plugins.

Among overlay-based plugins, Calico consumes more CPU in *Softirq* than Flannel, with an increase of 5% and 10% in 10 GbE and 100 GbE servers, respectively. This is due to the larger size of iptables for Calico, which incurs more packet forwarding overhead in *Softirq*. Thus, Flannel spends more time for *User* and *System* to generate packets, thereby increasing network throughput than Calico. Similar to Calico, Cilium also consumes more *Softirq* CPU resources than Flannel, with 5% and 12% more in 10 GbE and 100 GbE servers. We find that the extra hook points for eBPF programs increase the packet processing in *Softirq*.

In addition, we calculate the performance efficiency that divides the network throughput by the total CPU usage, which represents the network throughput per unit of CPU. Thus, the high performance efficiency means that the network plugin achieves high network throughput given the CPU usage. The reason why we introduce the performance efficiency is to help select suitable network plugins for cloud services. The lines with symbols (right y-axis) in Fig. 3 depict the performance efficiency of each plugin, and Kube-router shows the highest efficiency in both 10 GbE and 100 GbE servers. Specifically, Kube-router achieves 27%, 46%, and 47% higher than Flannel, Cilium, and Calico, respectively, in 10 GbE servers and 22%, 55%, and 62% higher in 100 GbE servers. This is because underlay-based Kube-router reduces the overhead in the overlay-based plugins, notably, *Softirq*.

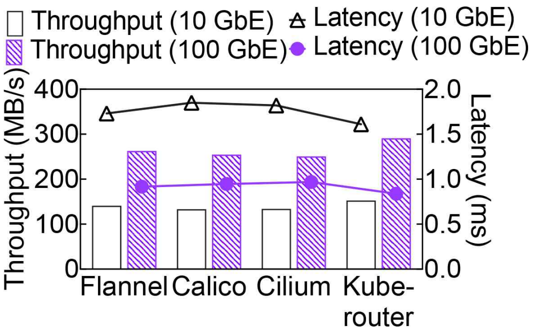
Also, Flannel shows the second highest performance efficiency in both 10 GbE and 100 GbE servers. This indicates that network performance efficiency increases as *Softirq* usage for iptables lookup decreases. Also, despite using the eBPF, Cilium lags behind Flannel due to the additional hook points, which increase *Softirq* CPU usage.

4.3 Nginx

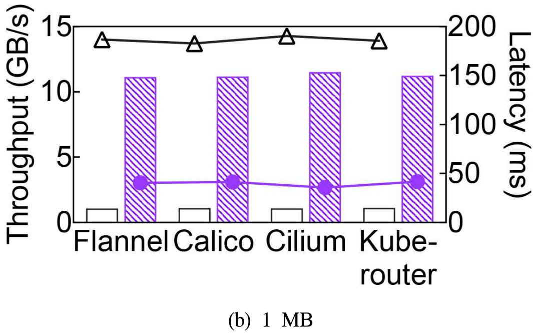
Next, we evaluate the Nginx workload. We create two Ngnix containers on a worker node. Then, we execute wrk as load generator that transmits HTTP requests to retrieve files continuously. For the experiment, each wrk instance generates 10 threads and 100 connections toward each Nginx container. Our experiments measure two file sizes: 1 KB and 1 MB.

4.3.1 Network Throughput and Latency

Fig. 4a depicts the network throughput (MB/s) and latency (ms) of Nginx containers for delivering 1 KB files. For 10 GbE servers, Kube-router outperforms overlay-based plugins. In particular, Kube-router achieves 8%, 14%, and 15% higher network throughput and 7%, 12%, and 13% lower latency compared to Flannel, Cilium, and Calico, respectively. However, the difference in performance between underlay-based and overlay-based plugins decreases compared to Memcached. For example, in Memcached, Kube-router has 19% higher network throughput than Flannel but only 8% higher in Nginx with 1 KB. This is because Nginx uses HTTP packets, which takes longer to send and receive than the regular TCP packet-based Memcached. This increased



(a) 1 KB



(b) 1 MB

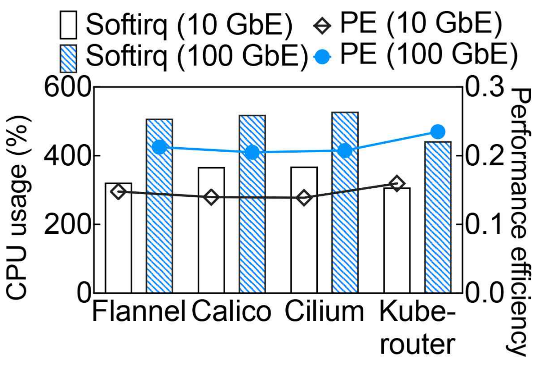
Fig. 4. Nginx throughput and latency.

time leads to the reduction in the number of processed packets (e.g., from 558 MB/s in Memcached to 152 MB/s in Nginx with 1 KB for Kube-router), resulting in lower performance difference. For the same reason, Flannel also presents only 5% higher throughput than Calico and Cilium. Also, 100 GbE servers show similar trends in throughput and latency between plugins.

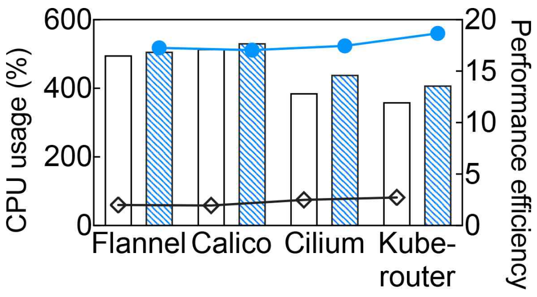
When the file increases to 1 MB (Fig. 4b), the CNI plugins have little performance difference among them—within 4% and 3% difference in throughput and 4% and 1% latency, respectively. Moreover, we find that the containers fully utilize the network bandwidth capacity independent of the type of plugins. We measure the network throughput, categorizing it into TX and RX by vnstat. In 10 GbE servers, the TX path averages at 9.85 Gbps, which includes response packets from the containers, and the RX path averages at 0.1 Gbps, which includes the reception of requests from clients. Also, in 100 GbE servers, the TX path averages at 99.6 Gbps and the RX path at 0.19 Gbps. This is because Nginx with 1 MB files is network-intensive, causing all plugins to saturate the network bandwidth, resulting in similar throughput among plugins.

4.3.2 CPU Usage

Fig. 5 illustrates the *Softirq* CPU usage of the server and performance efficiency running Nginx containers. Fig. 5a shows that the underlay-based Kube-router shows the lowest *Softirq* CPU usage in both 10 GbE and 100 GbE servers. In 10 GbE servers, the *Softirq* CPU usage of Kube-router is reduced by 4%, 16%, and 16% compared to overlay-based Flannel, Calico, and Cilium, respectively. So the low CPU usage for *Softirq* of Kube-router allows room for *User* and *System*, which increases the number of requests processing in the containers and improves the network throughput. As a result, Fig. 5a demonstrates that Kube-router achieves the highest efficiency while maintaining similar total CPU usage to other plugins. Similarly, Flannel reduces the CPU usage of *Softirq* by 12% compared to Calico in 10 GbE servers. This is due to the low packet forwarding overhead resulting from the smaller number of iptables rules. The



(a) 1 KB



(b) 1 MB

Fig. 5. *Softirq* CPU usage of Nginx containers.

reduction in the *Softirq* CPU usage results in a 5% increase in performance efficiency. The results of 100 GbE servers also follow the trends observed in the 10 GbE servers.

When the file size increases to 1 MB, Fig. 5b exhibits that the underlay-based Kube-router shows the lowest *Softirq* in both 10 GbE and 100 GbE servers. Specifically, the *Softirq* CPU usage of Kube-router decreases by 6%, 27%, and 30% compared to Cilium, Flannel, and Calico in 10 GbE servers, and by 7%, 19%, and 23% in 100 GbE servers. We find that Kube-router reduces the *Softirq* overhead by offloading packet segmentation to hardware (i.e., TSO), while overlay-based plugins cannot utilize TSO functionality, so they segment packets in the kernel, increasing *Softirq*.

When we compare the performance efficiency, Kube-router shows the highest value in both 10 GbE and 100 GbE servers. This is because Kube-router consumes the least *Softirq* CPU, which implies it consumes the least total CPU while achieving similar throughput close to the network capacity.

Next, among overlay-based plugins, Cilium consumes the lowest CPU for *Softirq* in 10 GbE and 100 GbE servers. The *Softirq* of Cilium decreases by 22% and 25% compared to that of Flannel and Calico in 10 GbE servers, respectively, and by 13% and 17%, in 100 GbE servers. This implies that Cilium can effectively reduce *Softirq* in network-intensive workloads by utilizing eBPF to bypass packet forwarding at the IP layer. As a result, Cilium provides higher performance efficiency than Calico and Flannel.

On the other hand, Flannel in 10 GbE and 100 GbE servers shows 3% and 5% lower *Softirq* CPU usage compared to Calico. The gap between the two is reduced when compared to Nginx with 1 KB. This is due to the iptables lookup being performed for each packet. Therefore, the number of packets that perform iptables lookup decreases as the number of requests processed per second decreases. Thus, the impact of packet forwarding overhead on CPU usage is reduced, and Flannel and Calico have similar performance efficiency.

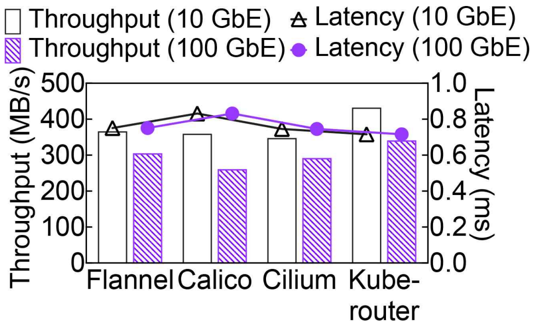
Also, in 100 GbE servers, Fig. 5b shows that CNI plugins achieve 7.6× higher performance efficiency on average compared to 10 GbE servers. This is because Nginx with 1 MB is network-intensive, so as the network capacity increases to 100 GbE, the plugins achieve 10× higher network throughput while CPU usage increases much less (only 56% on average), resulting in better performance efficiency.

4.4 Kafka

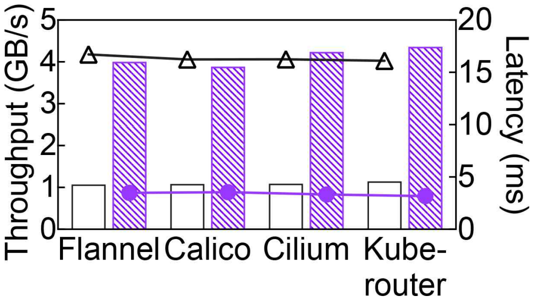
We present the Kafka evaluation results. We run two containers on a server, and each container acts as a Kafka broker with 4 partitions and 1 replication factor for 1 topic. Kafka-consumer-perf-test is executed as load generator to generate requests to Kafka containers and receive published messages in the containers. We vary the message size and the number of requests to (1 KB, 100M) and (1 MB, 1M).

4.4.1 Network Throughput and Latency

Fig. 6a depicts the network throughput (MB/s) and latency (ms) of Kafka containers for processing 1 KB messages. For 10 GbE servers, Kube-router achieves higher network throughput than overlay-based plugins. Specifically, Kube-router achieves 18%, 20%, and 24% higher network throughput and 4%, 4%, and 14% lower latency compared to Flannel, Cilium, and Calico, respectively. Also, 100 GbE servers show similar trends in throughput and latency between plugins.



(a) 1 KB



(b) 1 MB

Fig. 6. Kafka throughput and latency.

However, despite the network capacity increasing from 10 GbE to 100 GbE, the average throughput of the four plugins (in Fig. 6a) decreases from 376 MB/s to 300 MB/s on average. We find this decrease is due to the lower file I/O throughput of the 100 GbE server testbed (CloudLab). We additionally measure the file I/O throughput (iops) using ioping. The 10 GbE servers achieve 159 MiB/s, while 100 GbE servers achieve only 130 MiB/s. This lower file I/O performance in the 100 GbE servers affects Kafka operations, such as file I/O for reading messages in brokers (explained in §III). As a result, the network throughput of 100 GbE servers is reduced compared to 10 GbE servers.

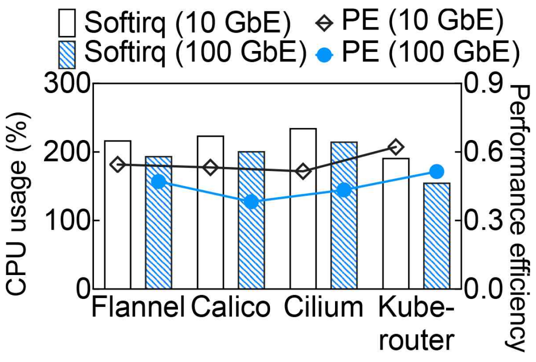
Next, for 1 MB messages (Fig. 6b), the average throughput of four plugins in 100 GbE servers is 4.12

GB/s, which is 3.7× higher than 10 GbE servers (1.13 GB/s on average). This is because Kafka is network-intensive for 1 MB messages so the throughput of Kafka is largely determined by network throughput. The increased network bandwidth to 100 GbE prevents Kafka from suffering from networking bottlenecks, resulting in higher throughput. Also, the lower I/O performance of the file system in 100 GbE servers in CloudLab, as discussed in the previous paragraph, does not have a significant impact because this workload is network-intensive.

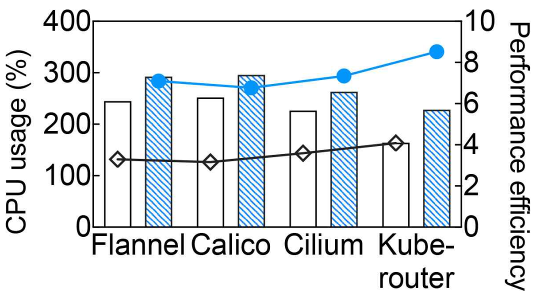
4.4.2 CPU Usage

Fig. 7 illustrates the *Softirq* CPU usage of the worker node and performance efficiency for 1 KB and 1 MB. In 10 GbE servers, Fig. 7a shows Kube-router uses 12%, 15%, and 19% lower *Softirq* than Flannel, Calico, and Cilium, respectively, and Kube-router achieves the highest efficiency. In addition, Flannel uses 5% less *Softirq*, compared to Calico and Cilium, and Flannel brings performance efficiency improvement by 5%. For 100 GbE servers, the *Softirq* CPU usage is reduced by 12% on average because the network throughput is lower than that of 10 GbE servers by the lower file I/O throughput (§4.4.1).

When we increase the message size to 1 MB, Fig. 7b shows Kube-router consumes the least *Softirq* CPU due to its minimal packet processing overhead. As a result, Kube-router achieves the highest performance efficiency, which is 14%, 23%, and 29% higher than Cilium, Flannel, and Calico, respectively. Among overlay-based plugins, Cilium shows the lowest *Softirq* CPU usage, lower than Flannel and Calico by 7% and 10%, respectively. The reduced CPU usage in *Softirq* leads to the decrease in the total CPU usage for Cilium. As a result, Cilium exhibits the highest performance efficiency that outperforms Flannel and Calico by 14%. For 100 GbE servers, the performance efficiency is twice as high as that of 10 GbE servers due to the 3× increase in throughput and 22% increase in *Softirq* usage.



(a) 1 KB



(b) 1 MB

Fig. 7. *Softirq* CPU usage of Kafka containers.

V. Summary of Results

Based on our analysis, we summarize the results as follows:

CPU-intensive workloads. When running CPU-intensive workloads, such as Memcached and Kafka 1 KB, all plugins saturate the given CPU resources in both 10 GbE and 100 GbE servers. However, the difference in the amount of *Softirq* usage makes performance efficiency vary significantly between plugins. The results show that the underlay-based Kube-router plugin consumes the fewest *Softirq* usage, achieving the highest performance efficiency. Among overlay-based plugins, Flannel exhibits the highest performance efficiency due to its lowest iptables lookup overheads. However, Flannel lacks support for containerlevel packet filtering configurations.

Network-intensive workloads. For network-intensive workloads (Kafka 1 MB), all plugins saturate the network bandwidth resources, revealing small differences (4-5%) in network performance. However, the CPU resources to achieve the network throughput vary significantly between plugins. In the overlay-based plugins, Cilium shows the highest performance efficiency with the lowest CPU usage. The reason is that Cilium reduces the time required to process these packets in the IP layer by its eBPF-based packet forwarding.

Insight on plugin selection. In short, the underlay-based Kube-router plugin is the best choice for both CPU-intensive and network-intensive workloads in terms of performance efficiency. However, many cloud service providers utilize overlay networks for container management when they need to isolate container networks from host namespaces, or when BGP is not available which is necessary to realize underlaybased networks. For CPU-intensive workloads, Flannel is recommended. For network-intensive workloads, Cilium is a suitable option.

Ⅵ. Related Work

We summarize the related studies in Table 4 and compare them with this study. First, to our knowledge, existing studies evaluate CNI plugins with limited workloads. For example, most existing studies evaluate CNI plugins with only micro-benchmarks that can have artificial and simple traffic structures. For instance, Koukis *et al.*[30] measure network throughput of several CNI plugins using micro-benchmark, iperf, and demonstrate that Calico achieves the highest throughput. However, our study,

Table 4. Related work comparison.

Study	Realistic workloads	Analysis metrics			Servers
		Latency	Performance efficiency	CPU usage profiling	
[13]	×	○	×	×	1 GbE
[14]	×	○	×	×	10 GbE
[7], [16]	Nginx	○	×	△	10 GbE, 25 GbE
[15]	×	○	×	×	10 GbE
[17]	Nginx	×	×	×	Unknown
[18]	×	×	×	×	1-5 GbE
[30]	×	×	×	△	Single server
[31]	×	○	×	×	1-10 GbE
This study	Memcached, Nginx, Kafka	○	○	○	10 GbE, 100 GbE

which measures network throughput using real-world applications, shows that Calico performs poorly due to excessive CPU usage, which provides new insights into their performance and resource usage. Furthermore, some other studies include Nginx, a real-world application, in their evaluations, but they only examine it in CPU-intensive scenarios. The results for CPU-intensive scenario under identical configurations (e.g., MTU size) align with our experiment results (Fig. 4a). However, Nginx also works as a network-intensive webserver, where our experiment results (Fig. 4b) show different results. In addition, no studies consider real-world workloads like Memcached and Kafka that we cover, which represent both CPU and network-intensive tasks.

Second, existing research mostly focuses on analyzing their throughput and latency or provides only a rough summary of CPU overheads. For example, Qi *et al.*[7,16] study the architecture of different CNI plugins and measure their throughput, latency, and CPU cycles in intra- and inter-host environments. Suo *et al.*[14] evaluate the performance of CNI plugins and investigate various factors (e.g., packet size and the number of containers) that affect the network performance of containers. However, they do not analyze the kernel-level CPU resource usage for packet processing (i.e., *Softirq*). CPU usage plays a critical role in determining the CNI plugin efficiency. For example, even though a plugin achieves high throughput, it can be inefficient if it requires heavy CPU usage.

Lastly, existing studies evaluate performance limited to 25 GbE[7,16]. In contrast, we cover 10 GbE and 100 GbE and show how plugins behave differently.

Ⅶ. Discussion

Reasons for choosing four CNI plugins: To choose CNI plugins for analysis in this study, we review CNI plugins listed in the official Kubernetes documentation[32]. While various CNI plugins are available for Kubernetes, we choose and analyze four plugins, Flannel, Calico, Cilium, and Kube-Router, as the state-of-the-art (SOTA) plugins for the following reasons.

First, the four plugins in this study are continuously and actively updated and maintained. Second, they are the most commonly used plugins for networking in Kubernetes clusters[33-35]. Third, tools to support running CNI plugins, like Canal[36] and CNI-Genie[37], also utilize these four plugins.

On the other hand, some CNI plugins listed in the Kubernetes documentation, such as WeaveNet[38] and Romana[39], have not been maintained for the past five years or have officially ceased their updates, which means they are not SOTA methods. Also, plugins, such as Spiderpool[40], Multus[41], and Contiv[42], are designed to support hardware acceleration features like DPDK or RDMA. These plugins depend on specific hardware configurations, whereas this study focuses on plugins that operate without additional hardware. Additionally, Antrea[43] and kube-OVN[44] are plugins that require extra software switches or SDN controllers beyond the kernel's networking stack. As they operate under different network architectures, they do not fall within the scope of this study. In summary, the four plugins analyzed in this study represent the SOTA models among Kubernetes networking plugins.

Ⅷ. Conclusion

This study explores four widely used Kubernetes network plugins in terms of architecture, throughput, latency, and CPU usage. We first analyze the packet processing of the plugins in-depth and compare the structural differences. We then conduct extensive experiments to evaluate the impact of these differences on network performance. Unlike previous studies that presented only micro-benchmarks, our study includes realistic workloads such as Memcached, Nginx and Kafka. These workloads encompass CPU-intensive and network-intensive workloads, reflecting the real-world. The major contribution of this study is that it devises a plugin selection criteria by workload characteristics, which can help Kubernetes cluster performance significantly.

References

[1] *Kubernetes*, Retrieved Sep. 27, 2024, from https://kubernetes.io/docs/concepts/overview/what-is-kubernetes

[2] I. Ahmad, M. G. AlFailakawi, A. AlMutawa, et al., "Container scheduling techniques: A survey and assessment," *J. King Saud University-Computer and Inf. Sci.*, vol. 34, no. 7, pp. 3934-3947, 2022. (https://doi.org/10.1016/j.jksuci.2021.03.002)

[3] G. Yang, K. Lee, K. Lee, et al., "Resource analysis of blockchain consensus algorithms in hyperledger fabric," *IEEE Access*, vol. 10, pp. 74 902-74 920, 2022. (https://doi.org/10.1109/ACCESS.2022.3190979)

[4] J. Santos, C. Wang, T. Wauters, et al., "Dik- tyo: Network-aware scheduling in container- based clouds," *IEEE Trans. Netw. and Service Manag.*, 2023. (https://doi.org/10.1109/TNSM.2023.3271415)

[5] W. Choi, Y. Yoo, K. Lee, et al., "Intelligent packet processing for performant containers in IoT," *IEEE Internet of Things J.*, 2024. (https://doi.org/10.1109/JIOT.2024.3453410)

[6] E. J. Ko, W. Choi, G. Yang, and C. Yoo, "Analysis of workload performance and resource consumption of lightweight secure virtualization platforms in IoT/edge computing," *J. KICS*, vol. 49, no. 10, 2024. (https://doi.org/10.7840/kics.2024.49.10.1386)

[7] S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Assessing container network interface plug- ins: Functionality, performance, and scalabil- ity," *IEEE Trans. Netw. and Service Manag.*, 2020. (https://doi.org/10.1109/TNSM.2020.3047545)

[8] *Project calico*, Retrieved May, 13, 2024, from https://projectcalico.docs.tigera.io/networking/vxlan-ipip

[9] *Cilium configuration*, Retrieved May 16, 2024, from https://docs.cilium.io/en/v1.13/network/kubernetes/configuration

[10] *Kube-router*, Retrieved Aug. 12, 2024, from https://github.com/cloudnativelabs/kube-router

[11] Flannel, Retrieved Aug. 16, 2024, from https://github.com/flannel-io/flannel

[12] W. Choi, K. Lee, J. Lee, and C. Yoo, "An analysis

on the network plugins of Kubernetes," in *Proc. KIISE*, pp. 250-252, 2021.

[13] G. Atici and P. S. Boluk, "A performance analysis of container cluster networking alternatives," *2nd Int. Conf. Industrial Con- trol Netw. and Syst. Eng. Res.*, pp. 10-17, 2020. (https://doi.org/10.1145/3411016.3411019)

[14] K. Suo, Y. Zhao, W. Chen, et al., "An analysis and empirical study of container networks," *IEEE INFOCOM 2018-IEEE Conf. Computer Commun.*, pp. 189-197, 2018. (https://doi.org/10.1109/INFOCOM.2018.8485865)

[15] S. Novianti and A. Basuki, "The performance analysis of container networking interface plugins in Kubernetes," *6th Int. Conf. Sustainable Inf. Eng. and Technol.*, pp. 231-234, 2021.

[16] S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Understanding container network interface plugins: Design considerations and performance," *2020 IEEE Int. Symp. Local and Metropolitan Area Netw.*, pp. 1-6, 2020. (https://doi.org/10.1145/3479645.3479700)

[17] N. Kapočius, "Overview of Kubernetes CNI plugins performance," *Mokslas-Lietuvos ateitis/Science-Future of Lithuania*, vol. 12, 2020. (https://doi.org/10.3846/mla.2020.11454)

[18] R. Bankston and J. Guo, "Performance of container network technologies in cloud environments," *2018 IEEE Int. Conf. EIT*, pp. 277-283, 2018. (https://doi.org/10.1109/EIT.2018.8500285)

[19] A. Archer, K. Aydin, M. H. Bateni, et al., "Cache-aware load balancing of data center applications," *VLDB Endowment*, vol. 12, no. 6, pp. 709-723, 2019. (https://doi.org/10.14778/3311880.3311887)

[20] R. Nishtala, H. Fugal, S. Grimm, et al., "Scaling Memcache at Facebook," *10th USENIX Symp. Netw. Syst. Design and Implementation*, 2013.

[21] D. DeJonghe, *Nginx CookBook*, O'Reilly Media, 2020.

[22] J. Kreps, N. Narkhede, J. Rao, et al., "Kafka: A distributed messaging system for log processing," *NetDB*, pp. 1-7, 2011.

[23] *Amazon EC2 instance type*, Retrieved Sep. 21, 2024, from https://aws.amazon.com/ec2/instance-types

[24] J. Weerasinghe and F. Abel, "On the cost of tunnel endpoint processing in overlay virtual networks," *IEEE/ ACM 7th Int. Conf. Utility and Cloud Comput.*, 2014. (https://doi.org/10.1109/UCC.2014.123)

[25] A. Mitseva, A. Panchenko, and T. Engel, "The state of affairs in BGP security: A survey of attacks and defenses," *Computer Commun.*, vol. 124, pp. 45-60, 2018. (https://doi.org/10.1016/j.comcom.2018.04.013)

[26] Y. Ghigoff, J. Sopena, K. Lazri, et al., "BMC: Accelerating memcached using safe in-kernel caching and pre-stack processing," *USENIX Symp. Netw. Syst. Design and Implementation*, pp. 487-501, 2021.

[27] D. Duplyakin, R. Ricci, A. Maricq, et al., "The design and operation of CloudLab," *Annual Technical Conf.*, pp. 1-14, 2019.

[28] *Wrk - a http benchmarking tool*, Retrieved May, 17, 2024, from https://github.com/wg/wrk.git

[29] *Managing Apache Kafka*, Retrieved Sep. 12, 2024, from https://docs.cloudera.com/cdp-private-cloud-base/7.1.6/kafka-managing/topics/kafka-manage-cli-perf-test.html

[30] G. Koukis, S. Skaperas, I. A. Kapetanidou, L. Mamatas, and V. Tsaoussidis, "Evaluating CNI plugins features & tradeoffs for edge cloud applications," *2024 ISCC*, pp. 1-6, 2024. (https://doi.ieeecomputersociety.org/10.1109/ISCC61673.2024.10733657)

[31] V. Dakić, J. Redžepagić, M. Bašić, and L. Žgrablić, "Performance and latency efficiency evaluation of kubernetes container network in- terfaces for built-in and custom tuned profiles," *Electr.*, vol. 13, no. 19, p. 3972, 2024. (https://doi.org/10.3390/electronics13193972)

[32] *Kubernetes - Installing Addons*, Retrieved Jan, 13, 2025, from https://kubernetes.io/docs/concepts/cluster-administration/addons

[33] Y. Ma, S. Smith, B. Dai, et al., "Uninet: Accelerating the container network data plane in iaas clouds," *2024 IEEE 17th Int. Conf. Cloud Computing* (*CLOUD*), pp. 115-127, 2024. (https://doi.org/10.1109/CLOUD62652.2024.00023)

[34] *GKE networking overview*, Retrieved Jan, 13, 2025,

from https://cloud.google.com/kubernetes-engine/docs/concepts/network-overview

[35] H. Koziolek and N. Eskandani, "Lightweight kubernetes distributions: A performance com- parison of microk8s, k3s, k0s, and microshift," in *Proc. 2023 ACM/SPEC Int. Conf. Performance Eng.*, pp. 17-29, 2023. (https://doi.org/10.1145/3578244.3583737)

[36] *Install Calico for policy and flannel (aka Canal) for networking*, Retrieved Jan. 13, 2025, from https://docs.tigera.io/calico/latest/getting-started/kubernetes/flannel/install-for-flannel

[37] *CNI-Genie*, Retrieved Jan. 13, 2025, from https://github.com/cni-genie/CNI-Genie

[38] *Weave Net - weaving containers into applications*, Retrieved Jan. 13, 2025, from https://github.com/weaveworks/weave.git

[39] *Romana - network and security automation solution for cloud native applications*, Retrieved Jan. 13, 2025, from https://github.com/romana/romana.git

[40] *Spiderpool*, Retrieved Jan. 13, 2025, from https://github.com/spidernet-io/spiderpool

[41] *Multus CNI plugin*, Retrieved Feb, 18, 2024, from https://anywhere.eks.amazonaws.com/docs/clustermgmt/networking/cluster-multus/

[42] *Contivpp.io*, Retrieved Jan. 13, 2025, from https://contivpp.io

[43] *Antrea*, Retrieved Jan. 13, 2025, from https://antrea.io

[44] *Kube-OVN*, Retrieved Jan. 13, 2025, from https://github.com/kubeovn/kube-ovn

Wonmi Choi

Feb. 2021 : B.S. degree, Korea University, Seoul, Republic of Korea

Mar. 2021~Current : M.S.-Ph.D. course, Korea University, Seoul, Republic of Korea

<Research Interest> Container virtualization, Container orchestration, Kernel networking stack, Federated learning

[ORCID:0009-0001-3290-8262]

Juyoung Ahn

Feb. 2023: B.S. degree, Korea University, Seoul, Republic of Korea

Sep. 2023~Current: M.S. course, Korea University, Seoul, Republic of Korea

<Research Interest> Container virtualization, Container orchestration, Federated learning

[ORCID:0009-0000-5671-2988]

Yeonho Yoo

Feb. 2017 : B.S. degree, Kookmin University, Seoul, Republic of Korea

Aug. 2021 : M.S. degree, Korea University, Seoul, Republic of Korea

Feb. 2024 : Ph.D. dgree, Korea University, Seoul, Republic of Korea

Oct. 2023~Apr. 2023 : Research intern, Microsoft Research Asia, Beijing, China

Mar. 2024~Current : Postdoctoral researcher, Korea University, Seoul, Republic of Korea

<Research Interest> Network virtualization, Network softwarization, Datacenter systems, AI systems

[ORCID:0000-0002-2636-633X]

Zhixiong Niu

Aug. 2012 : B.E. degree, Dalian Maritime University, Dalian, China

Aug. 2014 : M.S. degree, The University of Hong Kong, Hong Kong SAR

Aug. 2019 : Ph.D. degree, City University of Hong Kong, Hong Kong SAR

Feb. 2019~Current : Senior researcher, Microsoft Research Asia, Beijing, China

<Research Interest> Systems, Networking

[ORCID:0000-0001-6947-9740]

Gyeongsik Yang

Feb. 2015 : B.S. degree, Korea University, Seoul, Republic of Korea

Feb. 2017 : M.S. degree, Korea University, Seoul, Republic of Korea

Aug. 2019 : Ph.D. degree, Korea University, Seoul, Republic of Korea

Mar. 2018~Jun. 2018 : Research intern, Microsoft Research Asia, Beijing, China

Sep. 2023~Current : Assistant professor, Korea University, Seoul, Republic of Korea

<Research Interest> Operating systems, AI systems, Datacenter systems, Network virtualization, Network softwarization

[ORCID:0000-0003-4560-2972]

Chuck Yoo

Feb. 1982 : B.S. degree, Seoul National University, Seoul, Republic of Korea

Feb. 1983 : M.S. degree, Seoul National University, Seoul, Republic of Korea

Aug. 1986 : M.S. degree, University of Michigan, Ann Arbor, USA

Aug. 1990 : Ph.D. degree, University of Michigan, Ann Arbor, USA

Aug. 1990~Feb. 1995 : Researcher, Sun Microsystems Lab., USA

Mar. 1995~Current : Professor, Korea University, Seoul, Republic of Korea

<Research Interest> Operating systems, Digital healthcare, AI infrastructure

[ORCID:0000-0002-1115-1862]