2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Uncovering Communication Bottlenecks in Scalable ROS 2 Deployments on Kubernetes for Cloud/Edge Robotics

Yongzhou Zhang[1,2], Oliver P. Waldhorst[1,2], Björn Hein[1,2]

***Abstract*— Containerization and orchestration using cloud-native technologies enable scalable deployment of robotic software. Integrating ROS 2 with Kubernetes offers a flexible infrastructure, but also introduces a complex, multi-layered communication stack - from DDS middleware to container networks and the physical layer. Each layer adds overhead and variability that impact application-level performance. This paper presents a comprehensive analysis of communication performance across the cloud–edge–robot continuum, focusing on throughput and one-way latency in scalable ROS 2 deployments. We evaluate communication across intra-robot, edge, and cloud segments using wired and wireless connections, including emerging technologies like Wi-Fi 7 and high-speed LAN. Using a Kubernetes-based testbed, we investigate various ROS 2 middlewares, CNI plugins, QoS configurations, and encryption options. Our experiments reveal the impact of network overlays, routing paths, and middleware choices on latency and bandwidth. Despite the inherent complexity, the results confirm the feasibility of deploying ROS 2 in orchestrated, scalable environments. We summarize key insights as practical takeaways, many of which apply beyond Kubernetes, to guide the design of robust cloud/edge robotic systems.**

I. INTRODUCTION

Deploying autonomous robots in real-world environments requires many software modules. Those demand substantial computing power and frequent updates to improve reliability and task performance [1]. Therefore, an infrastructure is needed that flexibly provides computing resources and supports long-term system maintenance. Cloud/edge computing enables offloading of modules such as grasping and perception to enhance performance [2]–[4]. Containerization helps manage dependencies and improve maintainability, while orchestration platforms enable scalable deployment. However, in a general computing infrastructure including cloud, edge, and robots, sufficient communication capability at the application layer must be ensured.

In practice, combining the two frameworks, ROS 2 in robotics and Kubernetes (K8s) from cloud native computing, offers significant potential for a general software infrastructure [5]–[7]. However, integrating both complex frameworks complicates the communication layer. Fig. 1 provides an overview of the communication stack. First, at the physical layer, static manipulators connect to the edge via Ethernet (typically up to 10 Gbps in robotics), while mobile robots rely on wireless links. Within the cloud, network virtualization introduces bandwidth limits and overhead. Second, at the K8s layer, the Container Network Interface (CNI) defines the networking interface. Many plugins are available, such as Flannel, Calico, and Cilium, each optimized for different cloud-native use cases. Depending on the underlying network, networking approaches like Border Gateway Protocol (BGP), Virtual eXtensible LAN (VXLAN), IP-in-IP, or extended Berkeley Packet Filter (eBPF) [8] can be used to realize L2/L3 communication. Third, at the ROS 2 layer, the Data Distribution Service (DDS) is used as the default communication middleware. Various implementations are available, e.g., FastDDS and CycloneDDS. More recently, Zenoh is being integrated as an alternative.

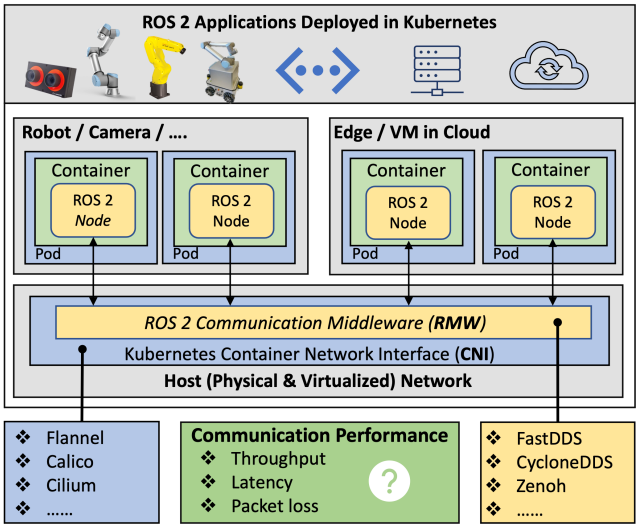

Fig. 1. Application-layer communication performance depends on factors such as physical network topology, virtualization, CNI, DDS implementation, message payload size, publish rate, and settings like QoS and MTU.

Thus, in addition to novel concepts and case studies on ROS 2 and K8s, several key communication-related questions remain: (i) How does the communication perform in terms of bandwidth, latency and jitter to fulfill the requirements from robotic applications with varying payload sizes and publish rates? (ii) How can it be optimized using the distinct opportunities available at different layers? (iii) When deploying a large number of software modules from robot fleets with cloud/edge computing, what communication constraints in the infrastructure must be considered? To address these questions, this study offers a comprehensive analysis of communication performance across all layers. Specifically, we assess the bandwidth and latency overheads when running ROS 2 in K8s, evaluate key performance-influencing factors, and identify important considerations for software scheduling.

[1]Karlsruhe University of Applied Sciences, 76133 Karlsruhe, Germany. {name.surname}@h-ka.de
[2]Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany

First, all the robots and edge/cloud nodes are added as part of the K8s cluster. The overall physical topology of the system is shown in Fig. 2. Communication traffic varies significantly across scenarios, as the location of bottlenecks also changes. For example, static manipulators connected via LAN benefit from high bandwidth, but the software stack's efficiency becomes the limiting factor. In contrast, for mobile manipulators using wireless communication, both latency and bandwidth fluctuate depending on distance and the number of connected clients. Furthermore, the communication path between two ROS 2 software modules depends on the deployment strategy, potentially involving different layers of the stack. Sec. III therefore outlines four typical communication flows in K8s deployments: intra-container, intra-pod, intra-host, and cross-host. Evaluating communication performance across these cases is essential for making informed deployment decisions.

In Sec. V, we begin our experiments with a single publisher/subscriber (pub/sub) pair. First, we quantify the reduction in maximum throughput for different payload sizes, highlighting the overhead introduced by DDS, the CNI layer, and the overlay network using VXLAN. We also measure one-way latency to demonstrate how latency varies with different payload sizes and publishing rates. Regarding the CNI layer, our experiments show that performance improves when using eBPF instead of traditional iptables-based networking. At the ROS 2 layer, we evaluate how different Quality of Service (QoS) configurations affect latency across various payload sizes and publishing rates.

In Sec. VI, we incorporate the physical communication layer. For static manipulators connected via LAN, we assess performance with multiple publishers and subscribers, including the impact of optional encryption at the cluster level. For mobile robots, we set up a state-of-art Wi-Fi 7 infrastructure and analyze performance at varying distances from the access point (AP), as well as under different client-sharing scenarios involving a single AP. In each experiment, we discuss the observed phenomenon and explain the effects of the interaction between layers. Useful findings are summarized as takeaways being also valid for non-K8s deployments.

The primary goal of this work is to provide a comprehensive, quantitative analysis to support the development of a robust communication infrastructure for ROS 2 and K8s using modern networking technologies. Bridging multiple research domains, it offers roboticists insights into infrastructure potential for AI-based systems, networking experts clear requirements from a robotics perspective, and practitioners practical guidance for designing deployment pipelines and communication architectures.

II. RELATED WORK

Tracing back to the early stages of ROS 2 development in 2016, Maruyama et al. [9] conducted a proof-of-concept study on applying DDS to ROS 2, exploring its potential and limitations. The study in [10] evaluates ROS 2 performance in a time-synchronized distributed network for real-time control, focusing on latency and jitter. Similarly, [11] analyzes the real-time capabilities of ROS 2 and OROCOS on both vanilla and PREEMPT-RT Linux kernels, showing that system load significantly impacts real-time performance.

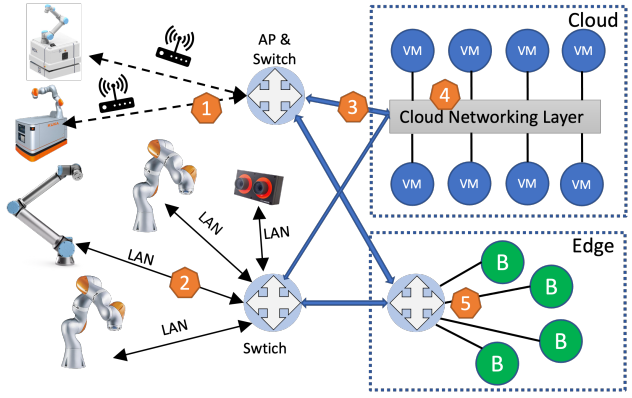

Fig. 2. Physical topology when deploying the robotic software in a cloud/edge/onboard computing continuum. Due to the variety of application requirements and hardware specifications, it is essential to assess different communication scenarios: connection to the edge/cloud ([1,2,3]), communication within the cloud ([4]), and within the edge ([5]).

The authors in [12] examine end-to-end latency in ROS 2 with DDS and provide a detailed breakdown of overhead introduced by the communication middleware. In [13], DDS implementations from various vendors are benchmarked for real-time behavior, and the study explores performance tuning strategies such as dedicated networking and CPU core isolation. For autonomous driving, [14] evaluates ROS 2 with respect to timeliness and error rates, while [15] conducts extensive experiments on a single platform to assess ROS 2 performance and offers design guidance for middleware.

In the ROS 2 community, ROS 2 netperf [16] is commonly used to measure point-to-point latency. While most of these studies focus on strict real-time requirements (e.g., sub-millisecond control loops) and point-to-point performance, the present work shifts attention to application-layer communication performance in K8s-based real-world deployments.

In the context of cloud/edge robotics, the work in [17] compares the performance of MQTT, Zenoh, and DDS in ROS 2 across various network setups for mobile robot applications. In [18], the authors demonstrate the feasibility of running DDS (Connext by RTI) applications within K8s and assess the performance impact of different CNI plugins, including Flannel, WeaveNet, and Kube-Router. However, their analysis primarily focuses on small payloads of up to 16 KB.

Compared to these works, we consider the entire robotic software stack, deployment strategies from a robotics perspective, and infrastructure specifications. While other benchmarking studies typically focus on a single layer, our analysis spans all relevant layers and evaluates their interaction effects. Through extensive experiments involving realistic payloads, QoS settings, publication rates, encryption, and varying numbers of publisher-subscriber pairs, we demonstrate how each layer contributes to overall system performance.

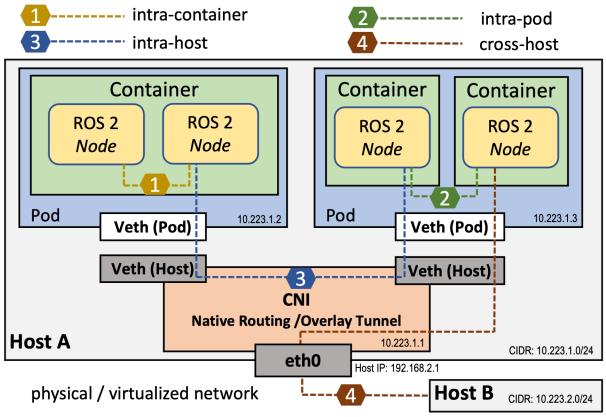

Fig. 3. Four traffic flows when deploying ROS 2 nodes in K8s. Depending on the deployment strategy, various overheads are introduced.

III. ROS 2 in Kubernetes

A. ROS 2 and Communication Middleware (RMW)

ROS 2 [19] adopts the DDS as its communication middleware for distributed robotic applications. It exposes standard communication interfaces through topics (publish/subscribe), services, and actions. Under the hood, the ROS 2 Middleware (RMW) layer serializes ROS 2 messages into Real-time Publish-Subscribe Protocol (RTPS) for DDS. DDS typically uses UDP as its transport, which is lightweight but lacks congestion control, retransmission, and flow control. Reliability is instead ensured through DDS mechanisms such as fragment resends, ACK/NACK exchanges, and heartbeats [20]. With QoS policies (e.g., reliability, history depth), ROS 2 allows to tune communication to application needs.

Multiple DDS implementations are available, each with different optimization goals and strengths (e.g., Fast DDS, CycloneDDS). Since serialization, message handling, and queue management differ across implementations, performance also varies. By default, DDS relies on multicast discovery, which is often unsupported by K8s CNIs due to high traffic overhead and degraded performance, and is commonly disabled in Wi-Fi networks.

B. Kubernetes and Container Network Interface (CNI)

In K8s, the basic deployment unit is a pod, which may contain one or more containers. For scalability and fault isolation, pods typically run a single container so that a crash affects only that instance. Each pod has its own network namespace (IP and routes) and communicates with other pods through the CNI. The CNI includes an IP Address Management (IPAM) plugin to assign IP addresses and configure routes on each node.

When the underlying network does not natively support workload IPs, traffic is encapsulated using an overlay such as Virtual Extensible LAN (VXLAN). It encapsulates OSI Layer 2 Ethernet frames into Layer 4 UDP datagrams. While this ensures connectivity, it introduces packet overhead and additional CPU cost for encapsulation and lookup. Many CNIs, such as Calico and Cilium, support bypassing overlays through direct routing, whereas simpler solutions like the

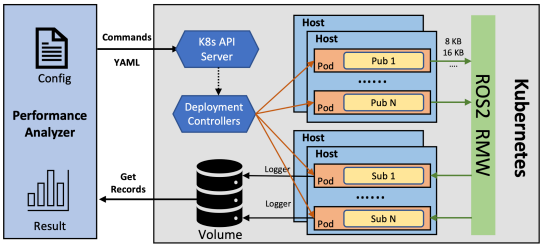

Fig. 4. Overview of our K8s-based approach for analyzing the infrastructure communication performance at the application layer.

easy-to-use Flannel typically rely on L3 overlays. In this work, we analyze the overhead and bottlenecks introduced by different CNI plugins, both qualitatively and quantitatively, with particular focus on their interaction with the datapath and the host network stack.

C. Deployment and Traffic Flow

Depending on the specification of software modules, ROS 2 nodes can be packaged into multiple containers with different levels of microservice granularity. Four deployment strategies are possible, each resulting in a different communication path, as illustrated in Fig 3.

- **Intra-container**: Communication stays inside one container using inter-process communication (IPC), loopback, or shared memory. This achieves the fastest performance.
- **Intra-pod**: Containers share a network namespace, traffic goes via UDP/TCP over the pod's virtual interface. Shared memory is possible by mounting `/dev/shm` but rarely used due to security concerns.
- **Intra-host**: Pods communicate through the CNI bridge. Overhead is small since most CNIs avoid encapsulation, yielding high throughput and slightly higher latency.
- **Cross-host**: Traffic between hosts depends on the CNI and the physical network. With direct routing encapsulation is bypassed; otherwise, overlay network introduces overhead.

From intra-container to cross-host, communication overhead increases along with microservice granularity, improving software isolation and maintainability but reducing performance. Evaluating the actual communication performance over the underlying physical network is thus essential to provide reliable capacity metrics for making informed deployment decisions.

IV. Experiment Approach

This section describes the metrics, hardware, cluster setup, and implemented ROS 2 package for measurement, along with the approach for automated testing, as shown in Fig. 4 It should be noted that this work does not focus on benchmarking individual DDS implementations or CNI plugins, but rather on a comprehensive analysis of the complex communication stack to uncover bottlenecks in deployments.

A. Metrics

Two metrics are used to represent the actual performance. First, bandwidth is measured at the pure ROS 2 message payload, excluding DDS and IP headers. This effective

maximum throughput reflects the infrastructure capability to transmit application data. Second, we measure one-way latency and jitter instead of round-trip time (RTT) due to asymmetric traffic between robots and the edge, as well as the inherent asymmetry of wireless links. This reduces disturbances when analyzing interaction effects. To avoid clock-related inaccuracies, we synchronize hosts using the Precision Time Protocol (PTP) [21] over the LAN interface.

B. Hardware and Kubernetes Cluster Setup

As robot client hardware the ASUS NUC 15 Pro is used, with Intel Core i3, Intel BE200 wireless card, and a 2.5 Gbps LAN. The edge server is equipped with an Intel i5 and 10 Gbps LAN. In the baseline evaluation switch port speed is reduced to 1 Gbps. The software stack runs on Ubuntu 24.04 with kernel 6.14.0-27-generic. The CPU is set to performance mode. The socket send and receive buffers are increased to 256 MB. For the Kubernetes cluster, we use the lightweight K3s (v1.33.4), since communication performance is determined by the CNI and remains consistent across distributions. As CNI, Calico (v3.30.3) is used, with VXLAN (MTU reduced to 1450 B) as the base configuration.

C. ROS2 Performance Measurement

To automatically run the experiment at scale, we propose a Kubernetes-based approach. First, we implement two ROS 2 nodes: one parses the configuration parameters (payload size, publish rate, etc.) and publishes messages, while the other receives the messages and records the network latency. The latency record is written to a mounted volume after measurement. To isolate the latency caused by the communication infrastructure from that introduced by serialization, we use a writable buffer and update only the message header, thereby avoiding additional allocations and memory copy operations per message. We then implement an analyzer tool that communicates with the cluster API, deploy and delete pods, and retrieves monitored data from the shared volume. Each measurement is run for 20 seconds. The ROS 2 package is implemented in ROS 2 Jazzy. We use CycloneDDS (v0.10) and disable the Multicasting.

V. ROS 2 Node to Node in K8s

This section analyzes the impact of different technologies and configurations in each layer using a single pub/sub pair. To investigate these effects, we vary payload sizes and publish rates, while considering traffic from real-world applications. The PHY-layer speed is limited at 1 Gbps, which represents the typical bandwidth of industrial PCs that are used as onboard devices in robots. Table I lists the payload and publish rate of common sensor data, outputs from function modules, and control messages. This provides a straightforward and quantitative overview of the communication resources required. In the following, we first evaluate throughput reduction and latency overhead due to overheads. We then examine the impact of the CNI under various traffic load. Finally, we investigate the ROS 2 RMW layer and the influence of QoS settings. In each case, we analyze and discuss the interaction effects across layers.

TABLE I

Example Messages: payload and publish rate [Hz].

Type	Payload	Rate	Note
IMU	0.3 KB	1000	with 3× covariances
RGB-compressed	1 MB	30	compressed
Depth raw	0.6 MB	30	640×480×16U
3D lidar (dense)	2.6 MB	20	multi-layer, ∼130k pts
Object detection	2 KB	30	boxes with IDs/scores
Object point cloud	512 KB	30	cropped 3D point cloud
Semantic Seg.	0.4 MB	10	per-pixel class labels
Grasp candidates	4 KB	10	top-K grasp candidates
Waypoints (Nav.)	5 KB	20	path with 100 timesteps
TCP target pose	0.2 KB	100	for visual servoing
Joint Trajectory	30 KB	10	12 DoF with 100 timesteps
VLA action plan	5 KB	0.1	skills with parameters

Note: payload values are approximate, as they depend on hardware configuration and algorithms setup.

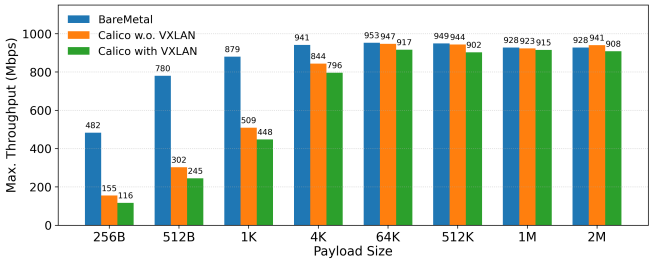

Fig. 5. Comparison of maximum throughput for different payload sizes with overhead. QoS: Best Effort.

A. Overhead of the Containerization and K8s CNI

Various protocol headers are appended for a given payload. First, the ROS 2 RMW adds DDS/RTPS headers, ranging from 56 B to 116 B depending on QoS. UDP (8 B) and IPv4 (20 B) headers are then added. When using an overlay network in K8s, such as VXLAN, 50 B is introduced. Using WireGuard encryption adds an additional 32 B. To prevent inefficient packet fragmentation, the fragment sizes for both DDS and CNI have to ensure the outgoing packet is below the network card MTU (usually 1500 B).

To analyze the overhead effect, experiments with Best Effort QoS are conducted to measure actual bandwidth in terms of effective payload. The publish rate for measuring maximum throughput is dynamically selected to ensure that the subscriber receive the messages with minimal packet loss. We observed that excessively high publish rates can cause the subscriber to receive no messages due to a networking collapse within the cluster.

As shown in Fig. 5, the bandwidth reduction is small for large payloads because the overhead is negligible compared to the payload size. For small payloads, the overhead dominates in increasing the packet size. For example, VXLAN expands payload of 256 B to 406–466 B. Nevertheless, due to the small payload size, the publish rate can reach 50 kHz for 256 B, which is sufficient from an application perspective.

Finding 1: *Bandwidth reduction in ROS 2 with Kubernetes remains acceptable for all common payloads.*

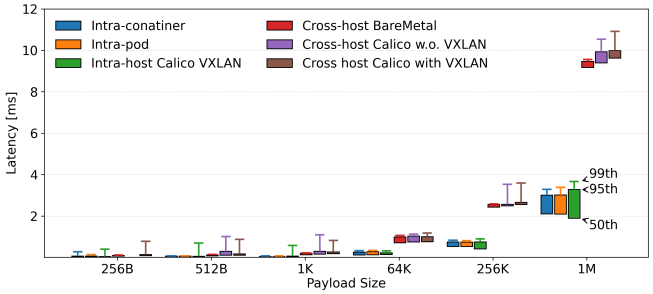

Fig. 6. Network latency overhead vs. payload size. The publish rate is selected to achieve 75% of the maximum throughput measured in Fig. 5.

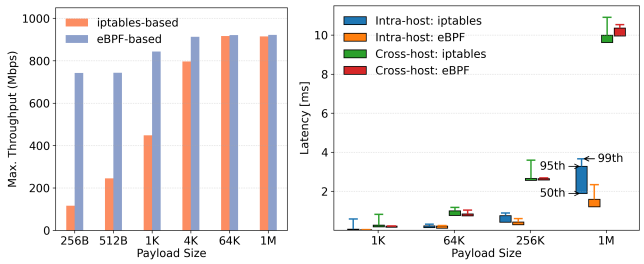

Fig. 7. Comparison of different routing modes in CNI: iptables-based vs. eBPF-based. QoS: Best Effort; VXLAN enabled.

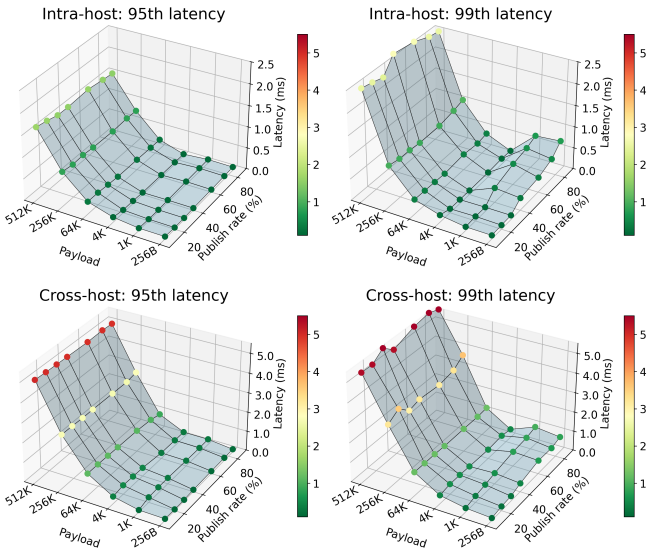

Fig. 8. 95th and 99th latency depends on the payload size and the publish rate. QoS: Reliable; VXLAN enabled.

To evaluate the latency overhead, we consider the deployment strategies described in Sec. III and measure the one-way latency of received messages. Experiments show that latency variance increases under medium to high network load, which is effectively caused by high publish rates. Therefore, we select a different publish rate for each payload size to generate a traffic that corresponding 75% of the maximum throughput obtained in previous measurements, which present a high network traffic load scenario.

As shwon in Fig. 6, for intra-host communication, the CNI maintains 50th-percentile latency at a level comparable to bare-metal or intra-container communication. The 95th-percentile latency grows with payload size due to additional CPU overhead for encoding, decoding, and encapsulation, which increases jitter. In practice, on the same host, CNIs can bypass VXLAN, so this setup mainly illustrates VXLAN's impact. For cross-host communication, latency remains comparable to bare-metal, but the 95th latency is higher for the same reasons as intra-host. For large payloads, where jitter already arises from lower networking layers, the variance through the CNI shows only little further increase.

Finding 2: *Acceptable latency overhead with the overlay network. The latency remains stable but with growing jitter.*

B. Container Network Interface Plugin

The Kubernetes ecosystem offers a variety of CNI plugins. It is essential to evaluate their performance under robotic traffic, as noticeable differences have been observed. In this section, we compare the maximum throughput and latency using Calico with two different routing methods: traditional iptables-based routing, which is similar to bare metal, and eBPF-based routing. The results are presented in Fig. 7.

Using eBPF as the datapath provides significant throughput gains for small payloads, as it bypasses the costly iptables/conntrack rule chain. Packet forwarding and policy enforcement are executed directly in the kernel via eBPF programs, reducing per-packet processing overhead. For larger payloads, throughput becomes limited by the physical layer. Regarding latency, in intra-host communication, eBPF reduces packet processing time by bypassing the iptables/conntrack chains. This is especially beneficial for large payloads, where fragmentation and reassembly add extra delay. For cross-host communication, median latency is comparable with iptables, but eBPF substantially reduces the jitter, especially for small payloads. Because eBPF has more deterministic execution path in the kernel, which avoids scheduling delays and nondeterministic queueing overheads.

Finding 3: *Compared to iptables, using eBPF improves the throughput for small payloads and reduces latency jitter.*

In this study, as the bare-metal baseline uses traditional iptables-based routing, we also use traditional iptables for all comparable experiments to ensure that the findings also apply to environments without Kubernetes.

With iptables-based routing, we found that the latency jitter depends on both payload size and publish rate. An experiment with varying payloads and publish rates is conducted to figure out the interactions between them. As shown in Fig. 8, where the 95th percentile applies to non–real-time modules, and the 99th percentile to time-critical tasks such as visual servoing. For small payloads, a higher publish rate increases the 95th and 99th latencies, For medium and large payloads, however, it only increases jitter, as demonstrated by the 99th latency. Even near the network limit, the latency can still be well maintained with best effort.

Finding 4: *Latency jitter of small payloads is sensitive to publish rate in the cluster, critical for high-frequency control.*

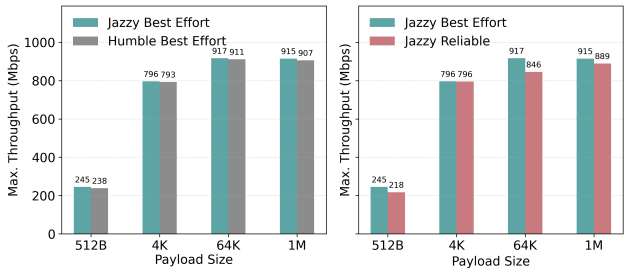

Fig. 9. Max. throughput comparison between ROS 2 versions (Humble vs. Jazzy) and QoS (Best effort vs. Reliable)

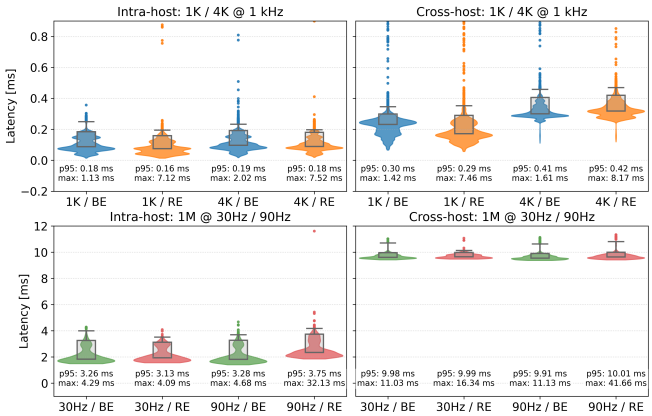

Fig. 10. Latency distribution under different QoS settings (Reliable vs. Best-Effort) in a reliable network, with zero packet loss in all cases.

C. ROS 2 Middleware (RMW)

Similar to CNIs, there are multiple DDS implementations, each with different optimization goals. This leads to performance variation across use cases. We compare maximum throughput of two ROS 2 distributions: Humble and Jazzy, in which only the `rmw_cyclonedds` version differs (1.3.4 vs. 2.2.3). Fig. 9 shows improved bandwidth in Jazzy. Regarding QoS, we compare the maximum throughput of Reliable and Best Effort. In the reliable network, since the throughput reduction with Reliable is minor, the measured with Best Effort can be used as the maximum capability for scheduling.

Finding 5: *Run the benchmarks again after upgrading the communication stack to ensure optimal performance.*

Regarding latency, we assess the effect of QoS (Reliable vs. Best Effort) for both intra-host and cross-host communication. Two representative payloads are considered: 1/4 KB at 1 kHz for control messages, and 1 MB at 30/90 Hz for sensor data. The latency distributions are shown in Fig. 10. For both small and large payloads, the additional overhead of Reliable QoS doesn't increase the 95th and 99th percentile latency. Because the LAN network is stable, the probability of retransmission is very low. DDS implementations usually use Batching ACKs and NACKs to reduce additional latency due to acknowledgment. However, this can introduce spike latency over the 99th percentile. In contrast, Best Effort shows consistent performance with comparable jitter, regardless of the payload or publish rates.

Finding 6: *In reliable networks, Best-Effort QoS is preferable, since Reliable QoS may result latency spikes.*

VI. From Robot to the Edge/Cloud

Integrating robots into K8s simplifies offloading. With minor deployment changes, containerized modules can be moved to cloud or edge nodes. However, communication varies by scenario. Static manipulators can use high-speed wired links, while mobile robots rely on wireless connections. Performance differs between a single pub/sub pair and multiple pub/sub pairs. The following experiments consider the physical layer and real-world application traffic to demonstrate feasibility and identify key performance factors.

A. Static Manipulators: Connection via LAN

For static manipulators, the physical-layer bandwidth is usually sufficient. Since many applications, such as visual servoing, require high throughput for streaming multiple sensor data and receiving control messages with low jitter. The middleware has to deliver those multiple large sensor data and high frequent sensor data simultaneously.

In the experiment, the LAN PHY bandwidth is at 2.5 Gbps. A 1 KB payload at 1 kHz represents small, high-rate control messages, while a 2 MB payload at 30 Hz represents large sensor data. Three publishers and three subscribers on dedicated topics emulate robots with multiple sensors. For security, we recommend using Kubernetes CNI encryption, which allows selective encryption of pods or nodes without modifying ROS 2 modules. The overhead of WireGuard is evaluated, with the MTU reduced to 1380 B. The results are shown in Fig. 11. The REF legend represents one pub-sub pair with equivalent data, such as 2 MB at 90 Hz. NP-1/2/3 denote individual pairs. Despite having the same total data, latencies vary slightly across pairs. This is due to different pub/sub paces causing minor congestion. For small payloads, the latency jitter is minor, but for large payloads, the difference becomes significant.

Finding 7: *Parallel pub/sub reduces network efficiency and increases latency and jitter compared to a single pub/sub with the same data volume.*

Encryption in K8s layer increases latency jitter for small payloads. For high-speed controls that require 2 ms loop time, encryption will become critical. With Reliable QoS, the latency overhead is minor for small payloads. However, for large payloads, latency and jitter rise sharply due to the additional CPU cost of encryption combined with unsynchronized, high-rate data from multiple pairs. This generates bursty traffic that overloads buffers and queues. With Best Effort QoS, the latency of delivered messages remain low; however, packet loss increases for large payload with high rates.

Finding 8: *Encryption can lead problems for high volumes of data over network. Reliable QoS increases jitter, while Best effort causes increased packet loss.*

B. Mobile Robots: Connection via WiFi

Mobile robots have more demand on computing power and generate more heavy sensor traffic and require still reliable latency. 5G network can better guarantee latency reliability with licensed spectrum but at high cost. The WiFi

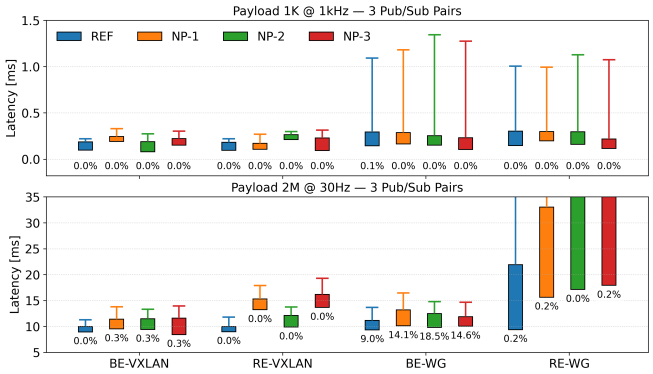

Fig. 11. 3 Pub/Sub pairs via LAN. NP: node pair; REF: Equivalent data amount with one pub-sub pair. WG: WireGuard; BE: Best Effort; RE: Reliable.

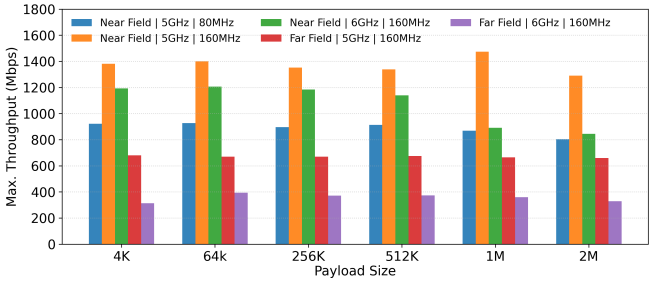

Fig. 12. Maximum throughput in ROS 2 within K8s, in different channels and different distances from the AP. Ceiling-mounted access point. Near Field: 8 m; Far Field: 20 m with a plasterboard wall in the middle.

is more accessible. The WiFi 7, with new features such as 6 GHz, Multi-Link Operation (MLO), and 4096-Quadrature Amplitude Modulation (4K-QAM), offers more performance gains, making it well suited for indoor robotics.

In this experiment, we use the UniFi E7 Access Point (AP) from Ubiquiti. The goal is to demonstrate its usability, assess real-world performance, and identify future challenges. In the experiment, we use the same NUC 15 PCs and don't mount additional antennas for signal enhancement. The AP is dedicated to these clients, and the transmission power is set high to maintain high performance.

First, we measured the maximum throughput in different channels (bandwidth and frequency) at different distances from the access point (AP). The near-field (8 m from the AP) represents good signal strength area, while the far-field (20 m, with a plasterboard wall in between) represents the worst case. As shown in Fig. 12, near-field throughput exceeded 1 Gbps in both 160 MHz channels, with even higher rates in the 5 GHz band due to the less signal damping at low frequency. In the far field, 160 MHz channel at 5 GHz still achieve over 500 Mbps, which is sufficient for transmission high-compressed images. In practice, a second AP would be deployed at this far distance to extend coverage and capacity.

Finding 9: *High bandwidth enables transmission of large sensor data, but effective capacity remains location- and channel-dependent and must be benchmarked.*

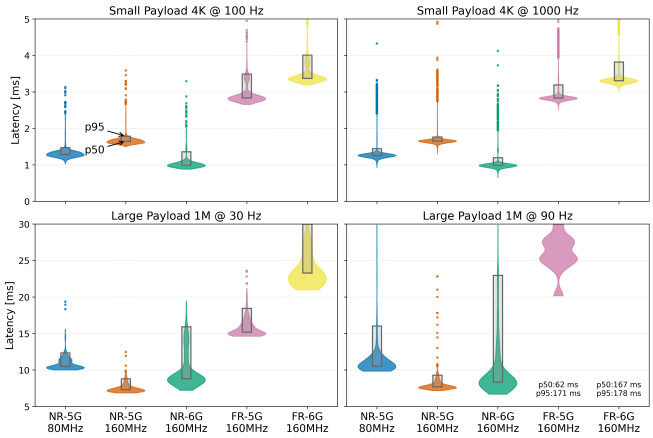

Fig. 13. Latency distribution via different channels in near (NR) and far (FR) field. QoS: Reliable. Signal strength: NR-5GHz: -44dBm, NR-6GHz: -50dBm, FR-5GHz: -66dBm, FR-6GHz: -74dBm

Regarding latency, one-way latency is measured with system clocks synchronized via PTP over LAN. K8s traffic is sent only over Wi-Fi. Fig. 13 shows latency distribution for a 4 KB payload at 100/1000 Hz and a 1 MB payload at 30/90 Hz. Reliable QoS is used, as Best Effort caused high packet loss. For small payloads, near-field latency is stable below 4 ms, while far-field adds 2 ms. The 6 GHz–160 MHz channel shows the best latency due to clean spectrum and only three UDP packets per 4 KB payload. 5 GHz performs better in the far field due to the better signal strength. For large payloads, the trend reverses: 6 GHz–160 MHz suffers high variance from fragmentation and retransmissions, and weak signals prevent reliable transmission. In the far field, the 5 GHz channel shows a packet loss rate of about 4%, whereas large payloads cannot be transmitted at all over 6 GHz–160 MHz channel.

Finding 10: *Unlike in reliable network, Reliable QoS is essential for large payload in lossy network; 6 GHz band can reduce latency but increases jitter for large payload.*

C. Multi-Robot with Shared AP

In practice, mobile robots commonly share an AP that operates on both the 5 and 6 GHz bands simultaneously. To study the effect of channel allocation, we tested three clients place in near field, each publishing 1 MB at 30 Hz as normal load, and at 50 Hz to emulate higher sensor loads (60 Hz caused excessive packet loss). We compare three scenarios: (i) all clients on 5 GHz, (ii) all on 6 GHz, and (iii) two on 5 GHz with one reassigned to 6 GHz. The latency distributions of all clients are shown in Fig. 14. At 30 Hz, all three setups achieve acceptable latency. However, jitter increases due to contention and retransmissions. Since the channel is not saturated, no dramatic spikes occur. However, reallocating client C3 (green) to the 6 GHz band significantly reduces jitter. At 50 Hz, congestion further increases latency jitter, reassignment to another band is essential for maintaining reliability.

Finding 11: *Channel assignment is crucial, as additional robots reduce effective bandwidth and increase latency jitter.*

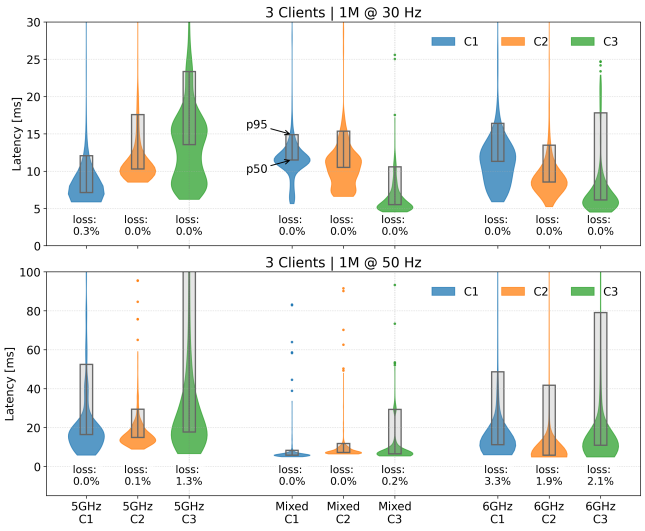

Fig. 14. Latency distribution under different channel allocations in near field. In the mixed case, C1/2 use 5 GHz and C3 uses 6 GHz.

VII. CONCLUSION

Establishing a general software infrastructure that integrates the de facto frameworks ROS 2 and Kubernetes is fundamental for enabling large-scale deployment of AI-based robots in real-world. Such an infrastructure must provide sufficient communication performance to fulfill the requirements of robotic applications. Based on our comprehensive analysis of this complex middleware stack, including both a technical deep dive and extensive experiments, we show that combining ROS 2 and Kubernetes is feasible despite communication overhead. However, communication constraints must be carefully considered during software scheduling and deployment.

By accounting for various physical hardware, we identify the key performance factors across scenarios and explain the interaction effects. The experimental results highlight recent advances while exposing bottlenecks for future work. We summarize 11 key findings from our lessons learned, which remain applicable outside Kubernetes deployments.

We also identified many open challenges. For instance, extremely high publish rates can cause networking issues, and suboptimal channel assignment in Wi-Fi leads to high latency variance. Since these problems span multiple layers and disciplines, this work also contributes a cross-domain perspective on infrastructure design for cloud/edge robotics.

ACKNOWLEDGMENT

Part of this research is being conducted as part of the KI5GRob project, funded by the German Federal Ministry of Research, Technology, and Space (BMFTR), and part of the Interreg Robot Hub Transfer project, funded by the European Union and Carl Zeiss Foundation (CZS) through the Interreg Upper Rhine program.

REFERENCES

[1] A. Dömel, S. Kriegel, M. Kaßecker, M. Brucker, T. Bodenmüller, and M. Suppa, "Toward fully autonomous mobile manipulation for industrial environments," *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, p. 1729881417718588, 2017.

[2] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[3] J. Zhang, F. Keramat, X. Yu, D. M. Hernández, J. P. Queralta, and T. Westerlund, "Distributed robotic systems in the edge-cloud continuum with ros 2: A review on novel architectures and technology readiness," in *2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2022, pp. 1–8.

[4] K. E. Chen, Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, and K. Goldberg, "Fogros: An adaptive framework for automating fog robotics deployment," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 2035–2042.

[5] Y. Zhang, C. Wurll, and B. Hein, "Kuberos: A unified platform for automated and scalable deployment of ros2-based multi-robot applications," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 9097–9103.

[6] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, H. Zhan, D. Xu, R. Ghassemi, J. Kubiatowicz, *et al.*, "Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2," in *Proceedings IEEE International Conference on Robotics and Automation*, 2023.

[7] Y. Zhang, F. Mirus, F. Pasch, K.-U. Scholl, C. Wurll, and B. Hein, "A comprehensive modeling and scheduling approach for allocating distributed multi-robot software to the edge/cloud," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 5799–5806.

[8] L. Rice, *Learning eBPF*. " O'Reilly Media, Inc.", 2023.

[9] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *Proceedings of the 13th international conference on embedded software*, 2016, pp. 1–10.

[10] L. Puck, P. Keller, T. Schnell, C. Plasberg, A. Tanev, G. Heppner, A. Roennau, and R. Dillmann, "Performance evaluation of real-time ros2 robotic control in a time-synchronized distributed network," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 1670–1676.

[11] S. Barut, M. Boneberger, P. Mohammadi, and J. J. Steil, "Benchmarking real-time capabilities of ros 2 and orocos for robotics applications," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 708–714.

[12] T. Kronauer, J. Pohlmann, M. Matthé, T. Smejkal, and G. Fettweis, "Latency analysis of ros2 multi-node systems," in *2021 IEEE international conference on multisensor fusion and integration for intelligent systems (MFI)*. IEEE, 2021, pp. 1–7.

[13] V. Bode, C. Trinitis, M. Schulz, D. Buettner, and T. Preclik, "Dds implementations as real-time middleware–a systematic evaluation," in *2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2023.

[14] J. Kouril, B. Schäufele, I. Radusch, and B. Schnor, "Performance evaluation of a ros2 based automated driving system," in *Proceedings of the 10th International Conference on Vehicle Technology and Intelligent Transport Systems - VEHITS*, 2024.

[15] T. Wu, B. Wu, S. Wang, L. Liu, S. Liu, Y. Bao, and W. Shi, "Oops! it's too late. your autonomous driving system needs a faster middleware," *IEEE Robotics and Automation Letters*, 2021.

[16] ROS 2 Community and Contributors, "Ros 2 netperf," https://github.com/ros2/netperf/tree/main, 2025.

[17] J. Zhang, X. Yu, S. Ha, J. Peña Queralta, and T. Westerlund, "Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ros 2 systems," *Journal of Intelligent & Robotic Systems*, vol. 110, no. 4, p. 162, 2024.

[18] 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 *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2021.

[19] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[20] Object Management Group (OMG), "Data distribution service (dds), version 1.4," Formal specification, OMG, Mar. 2015, oMG document number formal/15-04-10.

[21] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std. IEEE Std 1588-2008, 2008, precision Time Protocol (PTP).