# A Faster and More Reliable Middleware for Autonomous Driving Systems

Yuankai He
*University of Delaware*

Weisong Shi
*University of Delaware*

***Abstract*—Ensuring safety in high-speed autonomous vehicles depends on rapid control loops and tightly bounded delays from perception to actuation. Many open-source autonomy systems depend on ROS 2 middleware, but when multiple sensor and control nodes share a single computing unit, ROS 2 and its DDS transports introduce significant (de)serialization, data copying, and discovery overheads, eroding the available time budget. We present *Sensor-in-Memory (SIM)*, a shared-memory transport specifically designed for intra-host pipelines in autonomous vehicles. SIM keeps sensor data in their native memory layouts (e.g., `cv::Mat`, PCL), uses lock-free bounded double buffers that overwrite old data to prioritize freshness, and requires only four lines of code to integrate into ROS 2 nodes. Unlike traditional middleware, SIM operates beside ROS 2 and is optimized for applications where data freshness and minimal latency outweigh guaranteed completeness. SIM incorporates sequence numbers, a writer heartbeat, and optional checksums to provide data ordering, liveness, and basic integrity. On NVIDIA Jetson Orin Nano, SIM reduces data transport latency by up to 98% compared to ROS 2 zero-copy DDS such as FastRTPS and Zenoh, lowers mean latency by about 95%, and sharply narrows p95/p99 outlier latency by around 96%. In tests on a production-ready L4 vehicle running Autoware.Universe, SIM increased localization frequency from 7.5 Hz to 9.5 Hz. When applied across all latency-critical modules, SIM cuts average perception-to-decision latency from 521.91 ms to 290.26 ms, resulting in a 13.6 ft (4.14 m) shorter emergency braking distance at 40 miles/hr (64 km/hr) on dry concrete.***

## I. Introduction

Autonomous vehicles (AVs) are safety-critical. As speed increases, the *allowable perception-to-decision latency shrinks* because the vehicle travels farther each millisecond. At 40 mph (∼17.9 m/s), every 100 ms of delay adds ∼1.79 m to reaction distance. On a vehicle running the open-source Autoware.Universe stack, we measure a mean perception-to-decision time of **521.91 ms**, with **369.45 ms** in ROS 2 publish/subscribe paths—showing that the communication substrate can dominate end-to-end delay and tail behavior [1].

Open-source autonomy stacks, such as Autoware.Universe [2] and earlier Apollo releases [3], commonly use ROS 2 [4] with DDS for intra-vehicle messaging. Publish–subscribe improves modularity and reuse, but *(de)serialization*, *redundant copies*, and *dynamic discovery* add latency and jitter, especially with high-rate cameras and LiDAR. On embedded platforms with limited CPU and memory, these DDS-induced costs cap control-loop frequency and compress safety margins.

This work was supported in part by the National Science Foundation under Grants NSF 2140346 and NSF 2426482.

We introduce *Sensor-in-Memory (SIM)*, a domain-specific *shared-memory transport* and API for *intra-host* (single-ECU) AV pipelines. SIM keeps sensor data in *application-native layouts* (e.g., `cv::Mat`, PCL), removes (de)serialization and format conversion, and uses *bounded double buffers* with *overwrite-on-update* to favor *freshness over completeness*. A *lock-free* reader/writer interface yields predictable handoffs. Lightweight safety hooks—*sequence identifiers* (ordering), *writer heartbeats* (liveness), and an *optional checksum* (basic integrity)—reduce the risk of stale or corrupted reads. SIM integrates into existing ROS 2 nodes with ≈4 lines per pub-sub pair, preserves open-source tooling, and coexists with DDS/Ethernet for inter-ECU links.

**Scope:** intra-host pipelines where low latency and freshness are the primary objectives.

We evaluate SIM on NVIDIA Jetson Orin Nano and on a full-scale production-ready L4 vehicle against widely used *open-source, ROS 2-compatible zero-copy transports*: Fast DDS in shared-memory configuration [5], and Zenoh [6]. Across camera and LiDAR pipelines, SIM lowers end-to-end latency—**up to 98% lower maximum**, **≈95% lower mean**—and **reduces p95/p99 tails by ≈96%**. On Autoware.Universe, these gains raise application throughput (*NDT matching* 7.5 Hz → 9.5 Hz) and cut average perception-to-decision latency (521.91 ms → 290.26 ms). The shorter reaction time implies a **13.6 ft (4.14 m)** reduction in emergency-braking distance at **40 mph (64 km/h)** on dry concrete pavement.

**Contributions**

- **Safety-oriented bottleneck analysis** of intra-vehicle ROS 2/DDS paths in the Autoware autonomy stacks under high-rate sensing.
- **Design and implementation** of *SIM*, a native-layout, overwrite-first, bounded shared-memory transport with safety-aware lifecycle primitives (ordering, liveness, integrity).
- **Low-friction ROS 2 integration** (≈4 lines per pub-sub pair) that preserves open-source ecosystem compatibility.
- **Empirical comparison** on embedded and vehicle platforms against *Fast DDS (zero-copy SHM)*and *Zenoh* [5], [6], showing large mean/max reductions and **≈96%** p95/p99 tail reduction, plus application-level gains in Autoware.Universe.

Section II reviews ROS-based AV dataflow and details DDS overhead and jitter. Section III presents SIM's design and API. Section IV reports camera/LiDAR results versus Fast DDS

(zero-copy) and Zenoh on embedded and vehicle platforms. Section V concludes the paper.

## II. Data Flow and the Role of DDS in an Autonomous Vehicle

### A. ROS 2/DDS Intra-Host Data Path

In open-source AV stacks, ROS 2 with DDS provides publish–subscribe messaging and QoS control for perception, localization, planning, and control [4]. As sketched in Fig. 1 (top), sensor outputs are converted to ROS message types, serialized by DDS, transported, deserialized, and rematerialized into algorithm-native structures. Each step—type conversion, (de)serialization, queueing, and buffer handoff—adds CPU work and delay; with high-rate cameras/LiDAR and embedded SoCs, the accumulated cost eats into the perception-to-decision budget and stresses latency predictability.

### B. What Measurements Show

Empirical studies already show that message size, executor choice, and QoS settings leads to increases in mean and tail latency in ROS 2/DDS pipelines [7]–[12]. A recurring observation is that (de)serialization and redundant copying dominate at high throughput, and that p95/p99 behavior is as consequential as the mean for safety-critical loops.

### C. Tuning Within ROS 2/DDS (and Its Limits)

Response-time analysis and runtime tuning, such as profiling, executor variants, priority/affinity scheduling, and parameter auto-tuning, can reduce jitter [13]–[15]. However, these methods retain DDS abstractions; conversions and copies on the sensor-to-application largely remain, and tail latency often stays sensitive to background load [10], [11].

### D. Zero-Copy and Shared-Memory Communication

A complementary line of work shows that bypassing copies lowers latency and CPU load. Wu et al. and Bell et al. quantify the benefits of shared memory and zero-copy paths in latency-sensitive deployments [16], [17]. Systems such as *TZC* and *ROS-SF* further reduce copying and OS crossings [18], [19]. Several ROS 2 zero-copy paths (e.g., Fast DDS data-sharing with *loaned messages*) reduce copies *within the middleware boundary*, but they apply primarily to *plain/fixed-size* types and require adopting loan/return APIs; applications commonly convert ROS messages to algorithm-native layouts (e.g., OpenCV/PCL) before processing, which reintroduces conversion cost. DDS/RTPS exposes ordering and liveliness via QoS, yet per-frame *application-level* guards (explicit sequence tags, writer heartbeats, checksums) are typically left to the application [4]. These gaps motivate an AV-specific, lifecycle-aware, native-layout design.

### E. Alternatives Beyond ROS 2

Apollo *CyberRT* provides custom scheduling and IPC for high-throughput pipelines, and *AUTOSAR Adaptive* offers a service-oriented architecture with certification pathways [20], [21]. These target production ecosystems and are proprietary or tightly bound to specific stacks, limiting generalizability and independent evaluation in open-source contexts. For comparability and reproducibility, we center our study on ROS 2–based pipelines.

### F. Requirements for an AV Intra-Host Transport

Evidence above implies concrete requirements:

*a) R1: Native layouts, no (de)serialization on the sensor-to-application path.:* Remove conversion costs by publishing/consuming algorithm-native representations [16], [17].

*b) R2: Freshness-first, bounded sharing.:* Deterministically drop stale frames to keep reaction distance small under load.

*c) R3: Predictable handoff with minimal CPU work.:* O(1) copies and a lock-free common case [10], [11].

*d) R4: Safety-aware lifecycle.:* Ordering, liveness, and basic integrity for multi-sensor, multi-threaded pipelines [22], [23].

*e) R5: ROS 2 compatibility.:* Minimal code changes and coexistence with DDS for inter-ECU links [4].

### G. Positioning of SIM

As shown in Fig. 1 (bottom), SIM places a shared-memory buffer on the sensor-application data path, bypassing DDS while preserving ROS 2 integration. Data producers write application-native layouts; bounded double-buffers with overwrite-on-update enforce freshness (R2) and O(1) handoffs (R3); and sequence identifiers, writer heartbeats, and an optional checksum provide ordering, liveness, and basic integrity (R4). Integration occurs within existing ROS 2 nodes with $\approx$4 lines of code, leaving inter-ECU DDS paths unchanged (R5). The goal is not only lower averages but tighter p95/p99 latency in the perception-to-decision loop, which is directly tied to lost stopping margin.

## III. System Design

### A. Role and Scope

SIM shortens the *sensor-to-application* path in open-source AV stacks by replacing the ROS 2/DDS message/(de)serialization/copy chain with a preallocated shared-memory data plane. The design is strictly *intra-host*: it carries high-rate frames between local producers and consumers (e.g., LiDAR$\rightarrow$localization, camera$\rightarrow$obstacle detection), while inter-ECU links remain on ROS 2/DDS or Ethernet. Consistent with §II, SIM preserves native algorithm layouts, enforces freshness-first and bounded sharing, provides constant-time publication and consumption without locks, exposes per-frame ordering and liveness, and remains compatible with ROS 2 tooling and deployments.

### B. Architecture

As shown in Fig 2, each real-time stream owns a *named shared-memory region* that is created once and reused across processes. Regions are sized at initialization from sensor configuration so they can hold the *maximum* frame the sensor can produce. For cameras, resolution and format are stable,

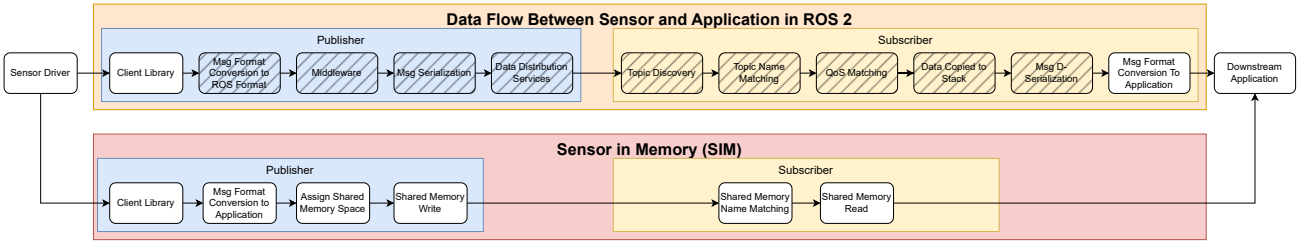


Fig. 1: Top: Data Flow and the Role of DDS in an Autonomous Vehicle
Bottom: SIM Overview

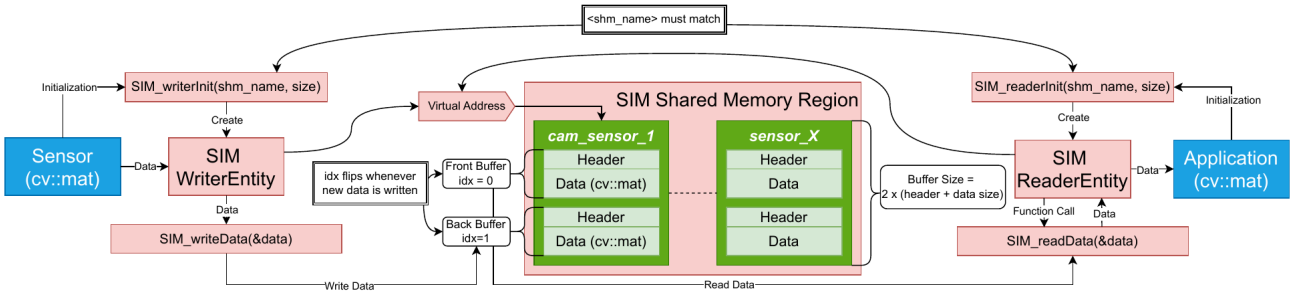


Fig. 2: SIM maps shared memory for each sensor, using unique identifiers, such as /camera_front, with dynamically sized buffers based on sensor specifications. Each sensor has dedicated regions without disrupting others. Library APIs like `SIM_writerInit(shm_name, size)`, `SIM_writeData(data)`, `SIM_readerInit(shm_name, size)`, `SIM_readData(data)`, and `SIM_destroy(shm_name)` are created facilitate memory management and data flow. In order for the application to read the correct sensor data, `SIM ReaderEntity` spawned by `SIM_readerInit(shm_name, size)` must have the exact `shm_name` as the `SIM WriterEntity` spawned by the `SIM_writerInit(shm_name, size)` from the sensor side. Instead of serializing sensor data into a ROS 2 format, data is stored in application-ready format (i.e. most computer vision applications use cv::mat, SIM stores camera images in `cv::mat`). The writer only flips the `idx` when it has finished writing to ensure no data is torn.

yielding a fixed byte size per frame. For LiDAR, deployments select a `MAX_POINTS` upper bound and allocate capacity accordingly. The region contains a small header and two payload buffers (double buffering). In our LiDAR instance, the header stores `idx` as an atomic integer, per-buffer `frame` numbers and `timeStamp` values, and a *published length* that records the size of the currently published frame. The payload holds `PointXYZ data[2][MAX_POINTS]`. Image streams store width, height, channels, stride, and depth so the payload is directly viewable as a `cv::Mat`. All memory is preallocated and mapped once with POSIX shared memory; steady-state operations do not allocate or make system calls. Payloads are laid out exactly as algorithms expect—PCL arrays for LiDAR; `cv::Mat` for images—so there is no message conversion or (de)serialization on the sensor-to-application path.

### C. Publish/Consume Protocol

A single writer publishes by filling the non-published buffer and then flipping `idx` atomically. Concretely, the writer selects the back buffer, writes the new frame into `data[idx=1]`, sets the *published length* to the number of points (or bytes) actually written for this frame, updates `frame[idx=1]` and `timeStamp[idx=1]`, and finally makes the back buffer the front buffer with a store that uses flips `idx` to 0. Readers begin by loading the front buffer with acquire ordering, check whether the `frame` number differs from the one they last consumed, and if so read the *published length* and copy exactly that many elements from the selected buffer into their workspace before updating their local `last_frame`. Because the writer commits only after finishing the back buffer and the length field, and readers always acquire the published index before reading, a reader either copies the new complete frame or the previously published one—never a torn frame and never beyond the written bounds. If producers outpace consumers, intermediate frames are intentionally skipped. That choice favors *freshness over completeness*, which is the safer policy for control at speed.

### D. Capacity Provisioning and Variable Effective Size

SIM provisions memory to the *maximum* the sensor can produce and supports frames that are *smaller than capacity* without padding. The header's *published length* records the actual number of points (for LiDAR) or bytes/pixels (for images) written for the currently published frame. Writers set this length before publishing the new `front_idx` with release semantics; readers observe both the index and the matching length under acquire semantics and therefore never read outside the address range. This policy removes per-

frame size races while preserving constant-time publication and consumption.

### E. Correctness and Timing Properties

We assume a single writer per stream, a bounded capacity fixed at initialization, and crash-stop processes. Capacity is respected because the published length is always $\leq$ the region's capacity. Publication order is well defined because only the buffer being published ever has its `frame` incremented. Visibility is guaranteed by writing the payload, its metadata, and the published length before the published index becomes visible to readers. From these invariants it follows that readers never observe torn frames or read past the end of the written data; readers do not wait for one another or for the writer; and memory usage and queuing are bounded to two buffers. The steady-state handoff time for a frame of size $B$ bytes is well approximated by $T_{\text{handoff}} = \frac{B}{BW_w} + T_{\text{publish}} + \frac{B}{BW_r}$, where $BW_w$ and $BW_r$ are sustained `memcpy` bandwidths of the writer and reader cores and $T_{\text{publish}}$ comprises a small number of atomic operations. There is no term for discovery, dynamic allocation, or (de)serialization, which explains the lower means and tighter p95/p99 observed in §IV.

### F. Fault Handling and Observability

Shared regions are kernel-backed and *named*, so a crashed process can reattach by name and resume operation without rebooting the stack. Readers detect producer stalls by comparing the most recent `timeStamp` against a per-stream deadline. A minimal diagnostics surface reports drops and the maximum inter-publish interval to aid tuning; during bring-up a per-buffer checksum can be enabled to catch corruption and then disabled in steady operation to save cycles.

### G. Placement and OS Configuration

Headers and payloads are cache-line aligned, and writer/reader threads are pinned to the same NUMA node as the region. Readers map the region read-only; writers map read–write. High-rate streams may lock pages in memory to avoid paging.

### H. ROS 2 Compatibility and Deployment

SIM is a small C++ library that preserves ROS 2 workflows. In latency-critical segments, a developer replaces a publisher/subscriber pair with SIM calls; in practice this change is on the order of four lines per node. Bridge processes expose SIM streams to remote consumers (SIM→DDS) or mirror remote inputs into local shared memory (DDS→SIM), which allows incremental rollout without disturbing inter-ECU communication.

### I. Design Choices and Scope

We use double buffering rather than rings because queues increase delay and tail variance and optimize for completeness rather than freshness. Readers copy out by default because that keeps them independent and wait-free; heavier modules can add a borrowed-pointer variant without changing the publication protocol. SIM assumes a single writer per stream and remains intra-host; multi-producer and cross-host transport continue to use ROS 2/DDS.

## IV. Experiment Setup and Results

### A. Experiment Setup

To evaluate SIM's mean latency and tail improvements, we benchmark on two compute platforms: (1) an NVIDIA Jetson Orin Nano (embedded, small-scale AV research) and (2) a state-of-the-art autonomous vehicle. Platform specifications are in Table I. We report CPU specs only, as the evaluated ROS 2 paths do not use the GPU.

We compare SIM against three ROS 2 middleware backends—Fast DDS (FastRTPS), and Zenoh—selected for their adoption and complementary capabilities (high-performance DDS, shared-memory zero-copy IPC, and a lightweight edge-to-cloud protocol, respectively). All backends use identical QoS (queue depth = 1, best-effort), fixed CPU affinity, the same governor, and the same scheduling policy.

This dual-platform design captures both a resource-constrained edge device (Orin Nano) and a high-bandwidth, multi-sensor AV.

Each setup evaluates two publisher–subscriber topologies: (i) one writer to one reader and (ii) one writer to ten readers, covering low-contention and fan-out scenarios. We report maximum, mean, minimum latency, p95/p99, and standard deviation. Latency measures transport time from immediately before `publish()` to message delivery; camera and LiDAR streams run end-to-end from hardware to application. Sensor models and rates differ per platform due to compute constraints; specifications appear in Table II. All ROS 2 sensor drivers timestamp just before the publish call (not at first-pixel/first-point capture).

### B. Goal and Hypotheses

We test whether the design in §III—native layouts with no (de)serialization (R1), bounded freshness-first sharing (R2), and constant-time atomic publication/consumption (R3)—reduces transport latency and tail percentiles on embedded hardware and lowers system scheduler load.

- **H1 (means & tails):** For LiDAR and camera streams, SIM yields lower *mean*, *p95*, and *p99* transport latency than ROS 2 baselines configured for zero-copy.
- **H2 (robustness under load):** Under 1W→10R, SIM's mean, p95, and p99 latency remains under the baseline.

### C. Methodology

**What we measure.** *Transport latency only*: elapsed time from immediately after the writer publishes a frame to immediately after the reader receives it; sensor capture and downstream compute are excluded. The writer stamps each frame after publish; the reader subtracts that stamp on delivery. Clock source: `CLOCK_REALTIME`; measured timestamp overhead on Orin Nano: 20 $\mu$s.[1]

[1]We verified alternative clocks; `CLOCK_REALTIME` had the lowest overhead and variance on our Orin build.

TABLE I: Experiment Setup

| | CPU Architecture | Max CPU Frequency | CPU Core | RAM | Max RAM Frequency |
|---|---|---|---|---|---|
| Orin Nano | Arm® Cortex A78AE v8.2 64-bit | 2.2 GHz | 8 | LPDDR5 8GB | 3200 MHz |
| Autonomous Vehicle | x86_64 Xeon E5-2667 | 3.2 GHz | 32 | DDR4 128GB | 2400 MHz |

TABLE II: Sensor configurations

| Platform | Sensor (type) | Rate | Payload |
|---|---|---|---|
| Orin Nano | Orbbec Astra Pro (camera)<br>Unitree 4D L1 (LiDAR, 18 ch) | 30 FPS<br>20 Hz | 640×480<br>2,160 pts/frame |
| AV | Basler Pylon (camera)<br>Pandar64 (LiDAR, 64 ch) | 42–120 FPS<br>10 Hz | 1920×1200<br>115,200 pts/frame |

**Data collection.** For each configuration, we collect **5** runs × **1000** frames (discard the first **100** frames). We report min, mean, p95, p99, max, and standard deviation. CPUs are pinned; clocks fixed; identical QoS (depth=1, best-effort) and executor settings across stacks; zero-copy/loaned paths enabled where supported.

### *D. Results on Jetson Orin Nano*

**Transports and workloads.** We first present a direct performance comparison of *SIM*, *Fast DDS (Zero-Copy)*, and *Zenoh*, where all transports are configured to run under the system's standard scheduling policy for a fair evaluation. We then separately demonstrate the additional optimization potential of SIM by applying a real-time *SCHED_FIFO* scheduling policy, a step that is straightforward with SIM but non-trivial to implement for complex middleware like DDS. Streams on Orin Nano (Table I): indoor LiDAR (18 channels @ 20 Hz; `PointXYZ[]` with **2,160** points/frame) and camera (640×480 @ 30 FPS; `cv::Mat`). Concurrency: 1W→1R and 1W→10R. For all report results on the Orin Nano - Whiskers show *min–max*; filled dot is the *mean*; long tick is *p95*; short dashed tick is *p99*. Transport-only; best-effort QoS; zero-copy enabled; pinned threads; fixed clocks; No scheduling priority elevation. **Mean (95% CI across runs, $n = 5$):**

*1) LiDAR Results (18 ch @ 20 Hz):*

*a) 1W→1R.:* Shown in Fig 3, SIM attains a mean of **0.2260**ms and tails p95=**0.2981**ms, p99=**0.3822**ms, max=**0.8788**ms. Relative to the best baseline using FAST DDS, p95 decreases by **70.02**% and p99 by **64.80**%; mean speedup=**3.02**×. These reductions follow from R1/R3: no ROS message conversion and constant-time handoff on the sensor-to-application path. Unlike ROS, SIM allows for an additional scheduling priority elevation on the kernel level that can reduce the latency to achieve a mean of **0.0737**ms and tails p95=**0.2428**ms, p99=**0.2556**ms, max=**0.2647**ms

*b) 1W→10R.:* Shown in Fig 4, SIM attains a mean of **0.542**ms and tails p95=**0.513**ms, p99=**0.570**ms, max=**9.990**ms. Relative to the best baseline using FAST DDS, p95 decreases by **57.87**% and p99 by **49.72**%; mean speedup=**2.94**×, supporting H2. Bounded overwrite-on-update (R2) avoids backlog under fan-out. While it seems like SIM has a larger maximum communication latency, unlike ROS, SIM allows for an additional scheduling priority elevation on the kernel level that can reduce the latency to achieve a mean of **0.0776**ms and tails p95=**0.2981**ms, p99=**0.3822**ms, max=**0.8788**ms

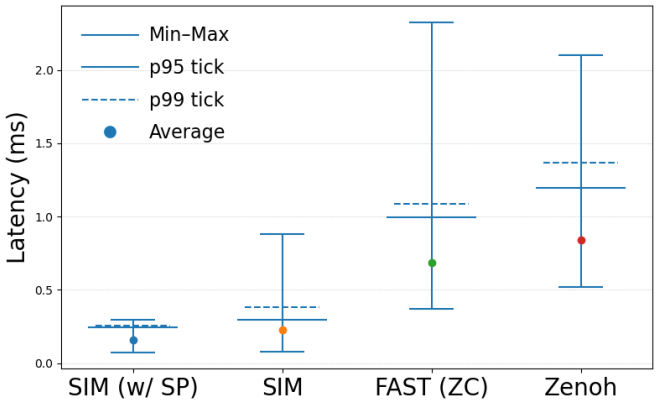


Fig. 3: LiDAR latency (1W→1R, linear-scale), Orin Nano. SIM 0.226 [0.219, 0.234]; Fast DDS 0.685 [0.650, 0.720]; Zenoh 0.839 [0.773, 0.905]. When scheduling priority is set to 99 with *SCHED_FIFO*, SIM's mean latency decreases to 0.158 [0.150, 0.164].

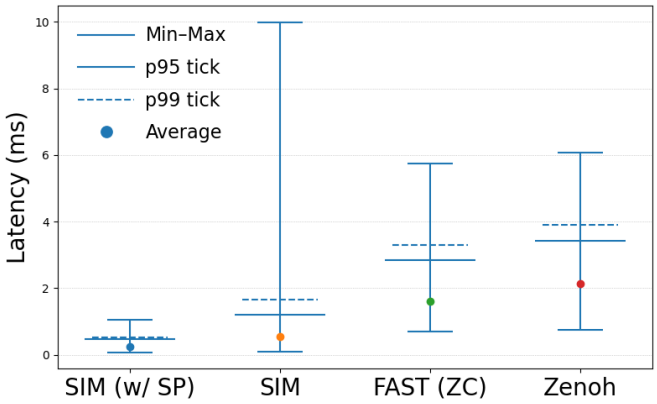


Fig. 4: LiDAR latency (1W→10R, linear-scale), Orin Nano. SIM 0.542 [0.513, 0.570]; Fast DDS 1.593 [1.583, 1.603]; Zenoh 2.136 [2.077, 2.196]. When scheduling priority is set to 99 with *SCHED_FIFO*, SIM's mean latency decreases to 0.231 [0.215, 0.246].

*2) Camera Results (640×480 @ 30 FPS):*

*a) 1W→1R.:* Shown in Fig 5, SIM attains a mean of **0.2537**ms and tails p95=**0.3454**ms, p99=**0.4665**ms, max=**2.4371**ms. Relative to the best baseline using Zenoh, p95 decreases by **91.92**% and p99 by **91.61**%; mean speedup=**14.6**×. These reductions follow from R1/R3: no ROS message conversion and constant-time handoff on the sensor-to-application path. Unlike ROS, SIM allows for an additional

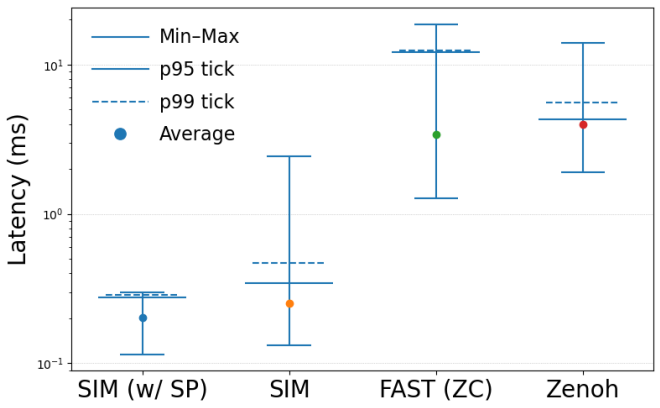


Fig. 5: Camera latency (1W→1R, log-scale), Orin Nano. SIM 0.253 [0.244, 0.261]; Fast DDS 3.40 [2.881, 3.920]; Zenoh 3.984 [3.877, 4.091]. When scheduling priority is set to 99 with *SCHED_FIFO*, SIM's mean latency decreases to 0.204 [0.195, 0.212].

scheduling priority elevation on the kernel level that can reduce the latency to achieve a mean of **0.1152**ms and tails p95=**0.2765**ms, p99=**0.2872**ms, max=**0.2985**ms. When using SIM or Zenoh, all of the published messages were received, FAST DDS loses 2.0% of the messages on average.

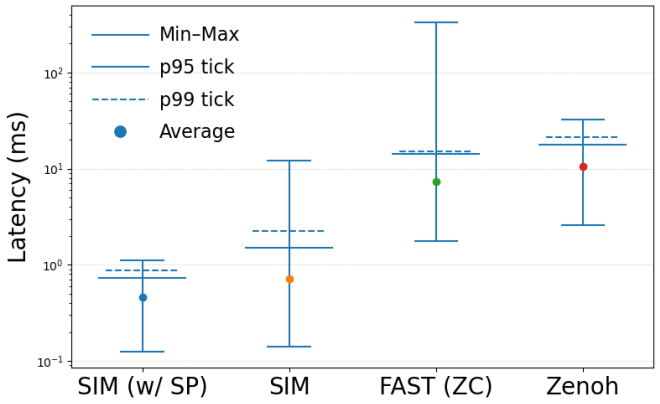


Fig. 6: Camera latency (1W→10R, log-scale), Orin Nano. SIM 0.709 [0.696, 0.739]; Fast DDS 7.226 [6.770, 7.682]; Zenoh 10.484 [10.176, 10.792]. When scheduling priority is set to 99 with *SCHED_FIFO*, SIM's mean latency decreases to 0.457 [0.126, 0.797].

*b) 1W→10R.:* Shown in Fig 6, SIM attains a mean of **0.709**ms and tails p95=**1.496**ms, p99=**2.23**ms, max=**12.205**ms. Relative to the best baseline using FAST DDS, p95 decreases by **89.41**% and p99 by **85.16**%; mean speedup=**10.44**×, supporting H2. Bounded overwrite-on-update (R2) avoids backlog under fan-out. While it seems like SIM has a larger maximum communication latency, unlike ROS, SIM allows for an additional scheduling priority elevation on the kernel level that can reduce the latency to achieve a mean of **0.4566**ms and tails p95=**0.7296**ms, p99=**0.8776**ms, max=**1.1149**ms. When using SIM or Zenoh, all of the published messages were received, FAST DDS loses 2.4% of the messages on average.

*3) System Scheduler Load:* We report *system-wide* scheduler load percentages (idle baseline, 1W→1R, and 1W→10R) measured over **1000** sensor data frames at **100**ms intervals, using identical affinities and fixed clocks across transports. The results are shown in Table III. Higher values indicate more time spent in kernel scheduling activity on the cores involved in the experiment.

TABLE III: System-wide scheduler load (%). Same setup as latency tables; measured at rest and under LiDAR/camera workloads. Max is 600%, with 100% for each processor core.

| Condition | SIM % | Fast DDS (ZC) % | Zenoh % |
|---|---|---|---|
| System Resting (idle) | 6 | 6 | 6 |
| LiDAR 1W→1R | 32 | 23 | **25** |
| LiDAR 1W→10R | 139 | 60 | **58** |
| Camera 1W→1R | 68 | **40** | 56 |
| Camera 1W→10R | 190 | **111** | 114 |

We also report the average CPU usage measured over 1000 frames at a sampling interval of 100ms, using identical affinities and fixed clocks across transports. Higher values indicate higher usage of CPU resources involved in the experiment.The results are shown in Table IV

TABLE IV: System-wide CPU usage (%). Same setup as latency tables; measured at rest and under LiDAR/camera workloads. Max is 600%, with 100% for each processor core.

| Condition | SIM % | Fast DDS (ZC) % | Zenoh % |
|---|---|---|---|
| LiDAR 1W→1R | **5.2** | 6.8 | 6.7 |
| LiDAR 1W→10R | 6.9 | 7.0 | **6.6** |
| Camera 1W→1R | 5.5 | 5.4 | **5.3** |
| Camera 1W→10R | 5.7 | 5.6 | **5.4** |

*Interpretation.* SIM does not introduce significant overhead to the read/write process compared to other ROS 2 DDS, while also lowering mean and p95/p99. At equal QoS and concurrency, SIM achieves lower latency at slightly higher kernel scheduling cost and similiar CPU costs, aligning with R1–R3 from §III. The increase in scheduler load for SIM in high fan-out scenarios (e.g., 190% for Camera 1W→10R) is an expected trade-off of our polling-based reader design. The value, representing cumulative kernel scheduling time on the involved CPU cores, is higher because readers actively poll the shared memory status to minimize receive-path latency. This CPU-for-latency trade-off can be tuned by adjusting the polling frequency. As shown in Table IV, overall CPU usage remains comparable to the baselines, indicating that SIM's approach is efficient despite the higher scheduling activity..

*Fault injection.* Similar to FAST and Zenoh, terminating and resuming the writer mid-task does not have any effect on the readers due to the double buffer setup; terminating and resuming a reader mid-task does not have any effect on other readers or the writer. Corrupting the shared memory region will terminate both the writer and the reader process. The OS

will assign a new shared memory region and resume normal read/write functions.

### E. Limitations

Reported latencies measure *transport only*; end-to-end perception-to-decision latencies include sensor capture and algorithm compute. All runs use best-effort QoS and enable zero-copy where available; switching to reliable QoS introduces retransmissions/back-pressure and increases baselines. Clock discipline and kernel build on Orin affect absolute numbers; we fix clocks, pin threads, and equalize affinity across stacks. Sensor formats and rates follow Table I. The comparative ordering follows from §III (native layouts, no (de)serialization, bounded overwrite-on-update, constant-time publication) and should hold for equivalent intra-host settings.

*a) Summary.:* Across LiDAR and camera workloads on Orin Nano, and under both 1W→1R and 1W→10R, SIM lowers mean and p95/p99 relative to Fast DDS (Zero-Copy) and Zenoh and maintains system-wide scheduler load. These results validate H1–H3 and directly reflect the mechanisms in §III.

### F. Results on a Full-Scale Autonomous Vehicle

*a) Goal.:* We evaluate SIM *in situ* on a production-ready autonomous vehicle running Autoware.Universe to answer three questions: (i) what is the end-to-end perception→decision latency and CPU load under the stock ROS 2 pipeline; (ii) how much of that latency is attributable to *communication* within ROS 2; and (iii) what changes when we replace only the *localization* module's intra-host transport with SIM, and what savings are achievable if SIM is applied across all intra-host stages.

**Transports and workloads.** We compare the sensor-to-decision latency of Autoware.Universe under native ROS 2 DDS (FAST (ZC)) to that of using *SIM without elevate scheduling priority* to . Streams on autonomous vehicle (Table I): one outdoor LiDAR (64 channels @ 10 Hz; `PointXYZ[]` with **115,200** points/frame) and seven camera (1920×1200 @ 42-120 FPS; `cv::Mat`).

*b) Method.:* We instrument Autoware.Universe nodes with timestamping at module boundaries and collect transport timestamps at publisher commit and subscriber delivery (sensor capture and downstream actuation are excluded). For each run we record *min/mean/p95/p99/max/std* over **5** runs × **[10 minute runs**. CPU load is sampled as *system CPU%* and *Autoware CPU%* at **100**ms with fixed clocks and pinned threads. QoS is best-effort (depth=1) across all modules.

*c) Baseline end-to-end and ROS 2 communication cost.:* Table V reports the end-to-end (E2E) perception-to-decision latency breakdown under native Autoware.Universe.

*d) CPU load (system and Autoware).:* Table VI summarizes CPU load with the stock stack and after replacing localization's intra-host transport with SIM.

TABLE V: Vehicle: Autoware.Universe E2E perception-to-decision latency (ms). Transport+compute on the vehicle; actuation excluded.

| Metric | Total | Preprocessing | Localization | Perception | Planning |
|---|---|---|---|---|---|
| Min | 320.363 | 43.818 | 18.769 | 188.750 | 69.026 |
| Mean | 521.905 | 90.844 | 30.960 | 280.860 | 119.242 |
| p95 | 788.794 | 127.421 | 66.822 | 396.179 | 198.372 |
| p99 | 1036.798 | 157.957 | 91.011 | 521.262 | 266.568 |
| Max | 1417.972 | 250.467 | 260.475 | 514.595 | 392.435 |
| Std | 123.079 | 25.847 | 17.822 | 38.677 | 40.734 |

TABLE VI: Vehicle: System and CPU load (%). System-wide under fixed clocks and pinned threads.

| Condition | System Load% | CPU Usage% |
|---|---|---|
| System resting (idle) | 4.2 | 200.4 |
| Stock ROS 2 (Autoware) | 57.1 | 1474.1 |
| SIM ROS 2 (Autoware) | 55.2 | 1474.4 |

*e) Replacing localization with SIM.:* We replace the *localization* module's intra-host transport with SIM. The NDT output frequency increases from **7.5 Hz** to **9.5 Hz** (the module's configured maximum is 10 Hz).

When using ROS 2 FAST DDS (ZC), the total latency averages 121.804 ms (8.2 Hz), and the sensor publishes data at 9.5-10 Hz. Over 10-minute runs, the average localization output drops to 7.5Ḣz. When using SIM to transport the data, the total latency decreases to sub-100 ms and the localization module is able to keep up with the sensor output to prevent any backlog.

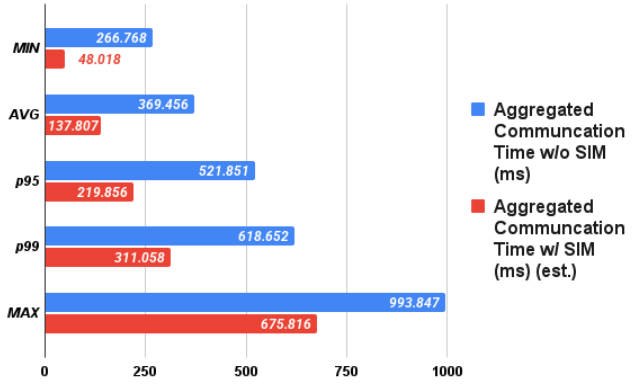


Fig. 7: Side-by-side aggregated communication time with and without SIM.

TABLE VII: Vehicle: E2E perception-to-decision latency (ms) saved with SIM.

| | Min | Mean | p95 | p99 | Max | Std |
|---|---|---|---|---|---|---|
| Time Saved with SIM (ms) | 218.750 | 231.649 | 219.855 | 307.594 | 318.031 | 53.681 |

*f) End-to-end impact with SIM across intra-host stages.:* As shown in Table VII and Fig 7, applying SIM across

all intra-host communications in *preprocessing, perception, localization, and planning* yields an average E2E reduction of **231.65 ms**. This corresponds to a shorter reaction distance of **13.6 ft (4.1 m)** at **40 mph (64 kmh)** and a shorter reaction distance of **23.8 ft (7.3 m)** at **70 mph (112 kmh)** on dry concrete[2] This reduction enlarges the safety margin available to the controller under emergency braking and high-speed maneuvers.

*g) Limitations (vehicle).:* We report single-run distributions without confidence intervals; results reflect the tested vehicle, sensors, and kernel build. E2E latency includes module compute and intra-host communication but excludes actuation. QoS is best-effort and identical across configurations. Projections for "SIM across intra-host" are derived from measured per-link communication costs, concurrency is accounted for.

## V. Conclusion and Future Works

We introduced SIM, a domain-specific shared-memory framework for AVs that uses a lightweight, lock-free, double-buffered design to cut latency and jitter beyond general-purpose middleware. Evaluations on a Jetson Orin Nano and a full-scale AV show consistent end-to-end latency reductions for high-throughput LiDAR/camera workloads and up to ten readers, with gains up to an order of magnitude in some settings—improving perception-to-control responsiveness at higher speeds. Future work will add worst-case determinism bounds, fault/load and multi-stream/multi-producer stress tests, and a clearer positioning/integration versus DDS/Zenoh and related shared-memory transports (Iceoryx/CycloneDDS/CyberRT). The SIM library and all benchmarking artifacts are available at https://github.com/Croquembouche/DAVOS_SIM

## References


[1] L. Liu, S. Liu, and W. Shi, "4c: A computation, communication, and control co-design framework for cavs," *IEEE Wireless Communications*, vol. 28, no. 4, pp. 42–48, 2021.

[2] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.

[3] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[4] 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. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[5] eProsima, "rmw_fastrtps: Ros 2 middleware interface for fast rtps." [Online]. Available: https://github.com/ros2/rmw_fastrtps

[6] A. Corsaro, L. Cominardi, O. Hecart, G. Baldoni, J. E. P. Avital, J. Loudet, C. Guimares, M. Ilyin, and D. Bannov, "Zenoh: Unifying communication, storage and computation from the cloud to the microcontroller," in *2023 26th Euromicro Conference on Digital System Design (DSD)*. IEEE, 2023, pp. 422–428.

[7] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *EMSOFT '16*. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2016, p. 10. [Online]. Available: https://doi.org/10.1145/2968478.2968502

[8] Z. Jiang, Y. Gong, J. Zhai, Y.-P. Wang, W. Liu, H. Wu, and J. Jin, "Message passing optimization in robot operating system," *International Journal of Parallel Programming*, vol. 48, no. 1, pp. 119–136, 2020.

[9] 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)*, 2021, pp. 1670–1676, iD: 1.

[10] 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)*, 2021, pp. 1–7, iD: 1.

[11] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J. J. Chen, "End-to-end timing analysis in ros2," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 53–65, iD: 1.

[12] J. Kouril, B. Schäufele, I. Radusch, and B. Schnor, "Performance evaluation of a ros2 based automated driving system," *arXiv preprint arXiv:2411.11607*, 2024. [Online]. Available: https://arxiv.org/abs/2411.11607

[13] A. A. Arafat, S. Vaidhun, K. M. Wilson, J. Sun, and Z. Guo, "Response time analysis for dynamic priority scheduling in ros2," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 301–306.

[14] A. A. Arafat, K. Wilson, K. Yang, and Z. Guo, "Dynamic priority scheduling of multithreaded ros 2 executor with shared resources," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 3732–3743, 2024, iD: 1.

[15] S. Macenski, A. Soragna, M. Carroll, and Z. Ge, "Impact of ros 2 node composition in robotic systems," *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 3996–4003, 2023, iD: 1.

[16] 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*, vol. 6, no. 4, pp. 7301–7308, 2021, iD: 1.

[17] O. Bell, C. Gill, and X. Zhang, "Hardware acceleration with zero-copy memory management for heterogeneous computing," in *2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2023, pp. 28–37, iD: 1.

[18] Y.-P. Wang, W. Tan, X.-Q. Hu, D. Manocha, and S.-M. Hu, "Tzc: Efficient inter-process communication for robotics middleware with partial serialization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7805–7812.

[19] Y.-P. Wang, Y. Dong, and G. Tan, "Ros-sf: A transparent and efficient ros middleware using serialization-free message," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, 2022, pp. 82–93.

[20] "Baidu Apollo team (2017), Apollo: Open Source Autonomous Driving, howpublished = https://github.com/apolloauto/apollo, note = Accessed: 2019-02-11."

[21] S. Fürst and M. Bechter, "Autosar for connected and autonomous vehicles: The autosar adaptive platform," in *2016 46th annual IEEE/IFIP international conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2016, pp. 215–217.

[22] S. Liu, X. Jiang, N. Guan, Z. Wang, M. Yu, and W. Yi, "Rtex: An efficient and timing-predictable multithreaded executor for ros 2," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 9, pp. 2578–2591, 2024, iD: 1.

[23] H. Choi, Y. Xiang, and H. Kim, "Picas: New design of priority-driven chain-aware scheduling for ros2," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 251–263, iD: 1.


[2]Reported as measured/estimated for this platform. Reaction-distance scaling uses $d = v \cdot \Delta t$ with speed in m/s and $\Delta t$ in seconds.