Psyclone AIOS: A Dynamically Reconfigurable Orchestration Substrate That Is Also Latency-Competitive Out of the Box

CMLabs Technical Report — v2.2, July 2026

Abstract

Psyclone AIOS is a compiled, in-memory orchestration substrate for real-time AI systems whose defining property is not raw speed but **dynamic reconfigurability**: context-addressed global routing, multi-trigger single-queue module gating, ephemeral compute, and live rewiring of module subscriptions — by the module itself, and (as substrate capabilities) by an LLM supervisor or a human operator. This paper does two things. First, it introduces Psyclone's architecture and explains why these capabilities constitute a different capability class from static-topology messaging systems. Second, it reports a rigorous single-node latency benchmark against NATS 2.14.3 and ROS 2 (lyrical / Fast-DDS) — 10 independent trials per condition, N = 100,000 messages per trial, equal N across systems, 95% confidence intervals, CPU pinning, and consistent one-way external-timestamp semantics across all harnesses. The headline results are reported without spin: **out of the box, with zero tuning, Psyclone beats NATS at every payload size and is within ≈4 µs of default ROS 2 at small payloads, with the tightest and most predictable tails of any system tested** (p99.9 ≤ 124 µs at 64 KB, where NATS reaches 761 µs). **After expert manual tuning, ROS 2 wins on raw median**: Fast-DDS shared-memory with data-sharing XML profiles reaches ≈27 µs, and single-process intra-process composition reaches ≈9 µs. We analyse what that tuning costs, what it gives up (the intra-process winner abandons process isolation and multi-host distribution entirely), and why the day-one, zero-tuning comparison is the one that matters for most engineering teams. The comparison also covers the true single-node peer class — Eclipse iceoryx, eCAL, and ZeroMQ `inproc` — measured on the same box with the same methodology, and the full campaign, including all three ROS 2 configurations, is replicated on bare metal (`z1d.metal`, C-states off): the dedicated zero/minimal-copy IPC specialists are faster than Psyclone's default (iceoryx ≈6 µs p50), tuned ROS 2 intra-process is a raw-median winner on metal (≈10 µs, near-payload-invariant), and the bare-metal run confirms that the outlier tail trials on the virtualised box were environmental jitter, not transport behaviour. We further report cross-node (multi-host) RTT/2 latency estimates, measurements of `rmw_iceoryx_cpp` (the fastest RMW measured), and a NATS run configured with `GOMAXPROCS=8` on both boxes. Psyclone itself is open source under LGPL v3, so every system in the comparison, Psyclone included, is independently rebuildable from published source. Psyclone's default is comparably fast rather than the raw-latency leader — iceoryx, rmw_iceoryx, and tuned ROS 2 post lower medians — but those are bare transports, whereas Psyclone is an orchestration substrate: crash-isolated processes, dynamic scaling, in-runtime simulation of new functionality, and a dataflow graph that is a runtime object. The latency microbenchmark is table stakes; the thesis of this paper is that Psyclone pairs competitive default latency and superior tail predictability with a set of runtime-reconfiguration capabilities — measured in Section 6 — that are not offered as first-class, third-party-drivable operations by NATS or ROS 2.

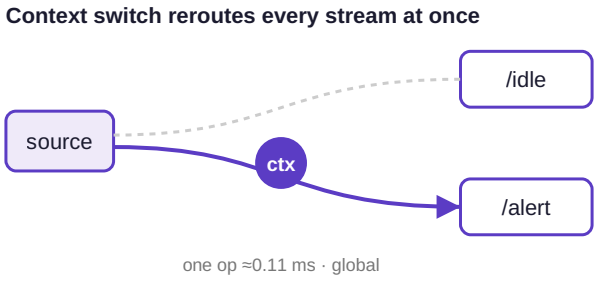
1. Introduction: What Psyclone Is, and Why It Is Different

Interactive AI systems — voice agents, embodied robots, multi-model copilots — are pipelines of specialised modules connected by a messaging fabric. Two questions determine whether a fabric is fit for this purpose:

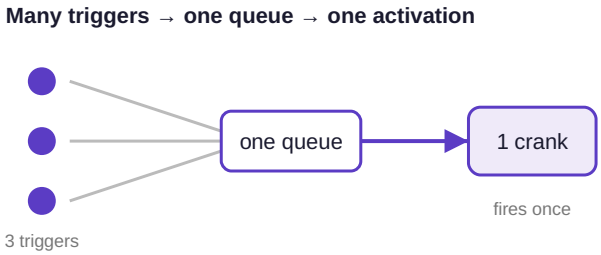
how fast does it move a message, and — increasingly the more important question — *how fast can the shape of the pipeline itself change?* Modern AI workloads are not static graphs. A voice agent that detects a foreign-language caller needs a translator in the loop *now*. A perception stack that spots an anomaly needs to re-route sensor streams from a summariser to a tracker *now*. An LLM-based supervisor that decides the current pipeline is suboptimal needs to rewire it *without a redeploy*.

Psyclone AIOS is built for that second question. It is a compiled (C++) orchestration substrate in which modules communicate through an in-memory, shared-space transport, and in which the *topology* of the dataflow graph is a first-class runtime object. Five capabilities define it; we state them up front because they frame everything that follows, including how to read the benchmark.

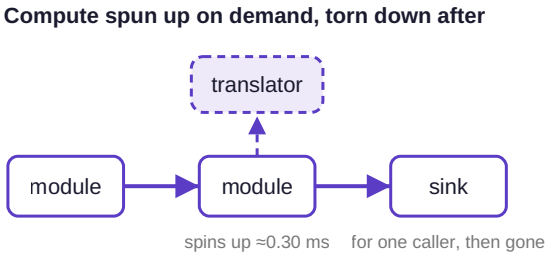
1. Sub-millisecond global data rerouting on hierarchical contexts. Psyclone routing is context-addressed, not topic-wired. Subscriptions are declared against a hierarchy of contexts (e.g. `/mission/idle` vs `/mission/alert/track`), and the *active context* determines which subscriptions match — globally across the graph. Switching context is a single sub-millisecond operation that re-routes every affected stream at once (Figure 5; measured at ≈0.11 ms end-to-end in Section 6.6 on a small single-node graph — multi-node and large-graph scaling is not yet measured). In ROS 2 or NATS the equivalent — changing which modules receive which streams — means launch-file remapping, node restarts, or hand-written application-level dispatch code on every subscriber.



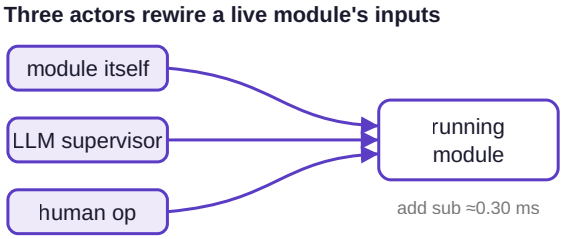
2. Trigger batching. When messages arrive in bursts, Psyclone can coalesce multiple pending deliveries into a single module activation instead of paying a wake-up, scheduling pass, and dispatch per message (Figure 7, left), and a single module can gate on multiple named triggers through one queue — the mechanism behind the multi-input join demonstrated in Section 6.2/6.6. We are careful here: the Section 4 latency benchmark ran one message in flight and therefore did **not** exercise batching, so batching is *not* the explanation for its tight tails (those come from the shared-memory transport and thin control layer). A dedicated burst/coalescing experiment with activation counting is future work; what is measured today is the multi-trigger single-queue join, not per-message dispatch coalescing under load.



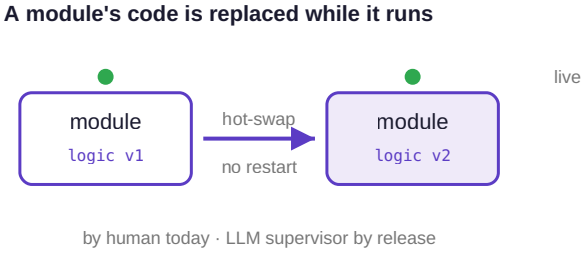
3. Ephemeral nodes and modules. Psyclone modules (and whole nodes) can be spun up and torn down on the fly as part of normal operation, with the new compute attaching to the running graph in-memory (Figure 7, right). ROS 2 nodes carry discovery, lifecycle-state, and DDS-participant weight that makes them poor candidates for per-request instantiation; NATS subjects are static strings with no attached compute lifecycle at all — spinning up a consumer is an application-level concern outside the fabric.



4. Runtime subscription updates — by the module, an LLM supervisor, or a human. A Psyclone module's subscriptions are already dynamic and pre-programmed, but crucially they can be *updated while the system runs*, and by three different classes of actor (Figure 6): the **module itself** (a module that notices it needs a new input stream simply adds the subscription — adaptive pipelines), a **system-level LLM supervisor** (a model that observes the running graph and rewires it — self-optimising, LLM-supervised orchestration), or a **human operator** (hot-patching dataflow in production without redeploying or restarting anything). This live rewiring of the dataflow graph turns pipeline topology from a deployment-time artifact into a runtime variable. Neither NATS nor ROS 2 offers an equivalent: both let a client change *its own* subscriptions at runtime, but neither provides a first-class way for a third-party supervisor or operator to rewire *another* module's inputs, nor a single global context switch that re-routes every affected stream at once.



5. Live reprogramming of the running system — including its code. Beyond rewiring which streams flow where, Psyclone allows the *behaviour* of a running system to be changed while it runs: a module's logic can be replaced, and new functionality introduced into the live graph, without tearing the system down. This is the deepest differentiator, because it turns not just the topology but the *program itself* into a runtime object. It is a working, in-product capability today, performed by a human operator; the same mechanism is being driven by an LLM supervisor in engineering now, and that autonomous path will be available by the time this paper is published — the substrate hook it drives already ships. No broker or DDS middleware offers anything comparable: in NATS and ROS 2, changing a component's behaviour means redeploying or restarting it. In Psyclone, the system's code is as mutable at runtime as its wiring — the property that matters when an AI system must adapt not only *where its data goes* but *what its components do*, as fast as its inputs change.



These five capabilities are a **different capability class** from what a broker or a DDS middleware provides, and they cannot be captured by a latency table. But a reconfigurable fabric that was *slow* would be

uninteresting, so the latency question still has to be answered — and answered honestly. Section 3 onward does that, with a methodology built to survive hostile review: equal-N, multi-trial, CI-reported statistics; both default and expert-tuned configurations of every competitor; identical one-way latency semantics across harnesses; and full disclosure of the environmental factors we could and could not control. The results include configurations in which Psyclone loses.

The honest headline is this: **Psyclone is comparably fast to the best-tuned alternatives while offering a class of capabilities none of them provide.** Out of the box, with zero tuning, it beats NATS at every payload, sits within a few microseconds of default ROS 2 at small payloads, and has far tighter tails than either at 64 KB. Expertly tuned ROS 2 (shared-memory XML profiles; single-process intra-process composition) and the dedicated zero/minimal-copy IPC specialists (iceoryx, eCAL, ZeroMQ `inproc`) post lower raw medians — and we report those numbers plainly, in the tables, including the ≈9 µs intra-process and ≈6 µs iceoryx results (Section 4.1). But those systems are *transports*: they move messages and nothing more. What an engineer should take from the data is that Psyclone pays a small, predictable latency premium over a bare transport and in exchange gets an orchestration substrate with crash-isolated processes, dynamic scaling of live compute, in-runtime simulation of new functionality, and — uniquely among everything measured — a dataflow graph that is a runtime object, rewireable while the system runs. **Psyclone moves messages competitively fast and lets an LLM rewire the dataflow at sub-millisecond speed and reprogram the system's code while running — the capability that matters when the shape of an AI system has to change as fast as its inputs.**

A note on scope and honesty: the capabilities described in this paper are those of the shipping Psyclone 2.1.0 substrate as exercised in our own products and tests — including live reprogramming of a running system's code, a working, in-product capability performed today by a human operator. The same mechanism is being driven by an LLM supervisor in engineering now and will be available by the time this paper is published; the substrate hook it drives already ships. A fully closed-loop *self-programming* system (Psyclone autonomously authoring and rewriting its own module specifications end-to-end) and cross-node module migration remain roadmap and are not claimed as shipping here.

2. Background and Related Systems

The systems we compare span three roles: distributed brokers, full robotics/DDS middleware, and dedicated in-memory transports. Figure 0 places them — and Psyclone — on the two axes this paper cares about: raw single-node speed, and how much of the system is reconfigurable at runtime.

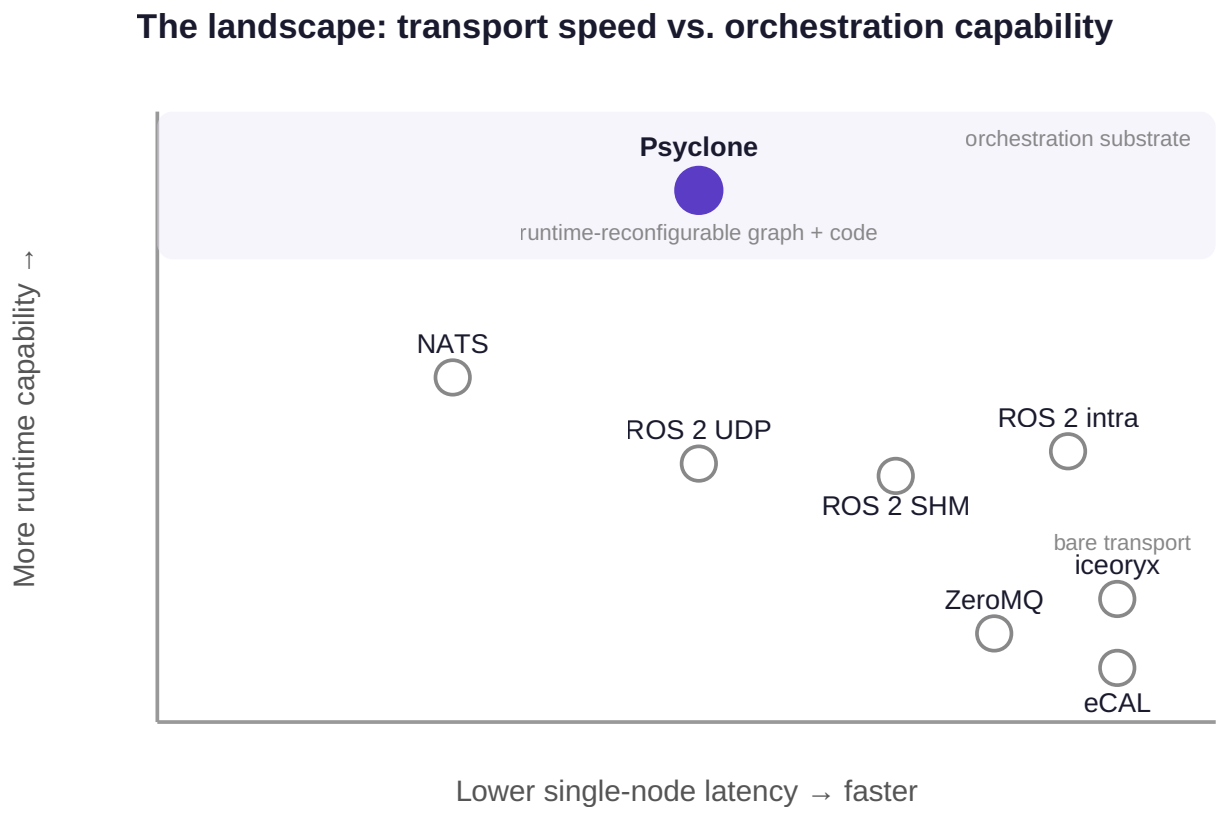


Figure 0. Indicative landscape. The dedicated transports (iceoryx, eCAL, ZeroMQ) win on raw latency but supply only a message path; NATS trades latency for broker capabilities; tuned ROS 2 configurations sit in between. Psyclone is the only system whose dataflow graph — and module code — is a runtime object. Latency positions are from Table 1; the capability axis is qualitative (§1/§6).

NATS is a mature network broker optimised for horizontal fan-out, multi-tenant subject routing, and very high saturating throughput (millions of msg/s). Every loopback message crosses the kernel socket stack twice (publisher→broker, broker→subscriber). We include NATS as the *distributed-messaging archetype*: beating it on single-node latency is expected — a broker's value is elsewhere — and we say so explicitly rather than presenting the gap as a discovery.

ROS 2 (lyrical release, `rmw_fastrtps_cpp` / Fast-DDS 9.4.8 in this study) is the de facto robotics middleware: typed topics, QoS policies, discovery, and a large ecosystem. Critically for this paper, ROS 2 spans several distinct latency regimes depending on configuration: default UDP loopback, Fast-DDS shared-memory with data-sharing (zero-copy) enabled via XML profiles, and single-process intra-process composition (`use_intra_process_comms`, unique_ptr hand-off). We measure all three, because the tuned regimes are the fair peer class for an in-memory fabric.

Eclipse iceoryx is a true zero-copy shared-memory transport built for automotive/robotics: publishers write into loaned shared-memory chunks that subscribers read in place, so payload size barely affects latency. It is the fastest cross-process transport we measured. We exercise it two ways: through its **native untyped API** (loaned chunks; measures the transport itself), and as `rmw_iceoryx_cpp`, iceoryx wired under ROS 2 as an RMW (§4.7). It supplies a transport only — no orchestration, scheduling, or lifecycle layer.

eCAL (Eclipse Continuous Enhancement Communication Abstraction Layer) is a publish/subscribe middleware whose local transport is shared memory. We run it 6.1.1 in its default configuration (SHM

transport, two processes). It lands in the same single-digit-microsecond class as the other IPC specialists at small payloads, with a one-copy penalty at large payloads.

ZeroMQ is a widely deployed messaging library; its `inproc` transport is the relevant peer here — a lock-free in-process channel between two threads (`ZMQ_PAIR`) with no process isolation, so it is a peer to ROS 2 intra-process rather than to the cross-process systems. We include it as the lightweight-library baseline.

These last three — iceoryx, eCAL, ZeroMQ `inproc` — are the **true single-node peer class** for an in-memory fabric, and all are measured (Sections 4.1, 4.6, 4.7). They set the raw-latency floor precisely because they are transports and nothing more: no broker, no orchestration layer, no runtime-reconfigurable graph. That is the comparison Psyclone must be read against — it pays a small, predictable latency premium over a bare transport in exchange for the substrate capabilities of Section 1. Graph-based agent frameworks (LangGraph-style runtimes, streaming voice toolkits) orchestrate LLM calls over Python objects, HTTP, or gRPC and occupy a latency regime orders of magnitude above everything measured here; they are not included.

3. Methodology

This section documents the controls adopted for this benchmark. The design goals were: equal statistical treatment, identical latency semantics across harnesses, and disclosure of every environmental factor we could and could not control.

3.1 Hardware and environment

All measurements ran on a single AWS EC2 `z1d.2xlarge` (Intel Xeon Platinum 8151 @ 3.4 GHz sustained-boost, 8 vCPU, SMT on, 62 GiB RAM), us-east-1c, Ubuntu with kernel 7.0.0-1006-aws. **CPU pinning** was applied with `taskset`: publisher/master on core 2, subscriber on core 3, `nats-server` on core 4; the Psyclone single process was pinned to cores 2–3; the ROS 2 intra-process binary pinned its publisher thread to core 2 and executor thread to core 3. **Frequency-governor control is not available** on this virtualised instance (no `/sys/devices/system/cpu/*/cpufreq`); we disclose this rather than claim it. Residual hypervisor jitter and shared-tenancy effects are a real threat to microsecond-scale measurement; they are bounded empirically by the cross-trial confidence intervals (Section 3.4) and discussed in Section 7.

3.2 Systems and harness semantics

All three harnesses measure the same quantity: **one-way, per-hop, same-host latency, timestamped at the sender immediately before send and differenced at the receiver on arrival, on a common clock**. This replaces v1's heterogeneous semantics (framework-internal age vs `nats bench` request latency vs a Python `rclpy` script).

- **ROS 2 (`rclcpp`, C++, `-O2` Release).** The publisher stamps `CLOCK_MONOTONIC` nanoseconds into a fixed-size POD message field immediately before `publish()`; the subscriber computes `now − stamp` in its callback. Same clock, cross-process (except intra), QoS reliable / keep-last-100. Three configurations: **`udp`** (`FASTDDS_BUILTIN_TRANSPORTS=UDPv4`, forced loopback UDP — the pessimistic baseline; note that stock Fast-DDS on lyrical already enables builtin SHM); **`shm`** (Fast-DDS SHM transport + data-sharing AUTOMATIC via `FASTRTPS_DEFAULT_PROFILES_FILE` XML with `RMW_FASTRTPS_USE_QOS_FROM_XML=1`, LoanedMessage API where available) — **manually tuned**; **`intra`** (single process, `use_intra_process_comms=true`, `unique_ptr` publish, zero-copy pointer hand-off) — **manually tuned**, and no longer a distributed system.
- **NATS (Go, `nats.go` v1.52).** Publisher stamps `time.Now().UnixNano()` into the first 8 payload bytes, paced with sleep-until-deadline; subscriber computes `now − stamp`. True one-way through

`nats-server` v2.14.3 on loopback TCP. This replaces v1's `nats bench` figures, whose round-trip/service-latency semantics were ambiguous.

- **Psyclone 2.1.0** (built from source; LGPL-v3 source package `Psyclone_LGPL_2.1.0.260712`, `psyclone2` submodule commit c8c469e, `git describe` BeforeGenAI-56-gc8c469e, clean working tree). The `PsyTest` PingMaster ring (master→slave2→slave3→master) with exactly one message in flight; each arrival records the framework message-age (created-to-received) of the last hop, i.e. one-way per-hop latency, at the facility's **1 µs resolution** (disclosed; ROS 2/NATS harnesses resolve to 10 ns). Spec: `psy_ping.xml`, trimmed from the stock `psytest_gen.xml`. Default install; **no tuning of any kind**.

3.3 Workload

Payloads of **64 B, 1 KB, and 64 KB**, spanning control signals to audio-frame/embedding-sized transfers. ROS 2 and NATS were driven at a **paced offered load** far below saturation — 5,000 msg/s (64 B, 1 KB) and 2,000 msg/s (64 KB) — so that measured latency reflects per-hop cost, not self-inflicted queueing (this corrects v1, where NATS's multi-millisecond 64 KB tails were partly a queueing artifact of 5k msg/s × 64 KB ≈ 327 MB/s through a loopback broker). Psyclone's ping-pong self-paces at its ≈8.5–9.5k msg/s round-trip cadence with one message in flight.

3.4 Statistics

10 independent trials per system/configuration/payload, each with **fresh processes** and **N = 100,000 messages** — equal N for every system, correcting v1's 10^5-vs-2×10^4 asymmetry. Every reported percentile is the **mean over 10 trials with a 95% t-interval confidence interval**. At N = 100,000 per trial, p99.9 rests on ≈100 samples per trial (≈1,000 across trials), giving usable tail estimates.

3.5 Addenda for this revision (NATS fix; cross-node estimator)

NATS harness fix (GOMAXPROCS). The NATS figures added in Section 4.6 (both virtualised and bare-metal) were re-measured with the `natslat` harness run at **`GOMAXPROCS=8`**. On the 48-vCPU `z1d.metal`, the Go runtime's default `GOMAXPROCS=48` combined with `taskset` pinning to a single core thrashed the Go scheduler and produced millisecond-class self-inflicted queueing — this is what invalidated the original metal NATS run. The re-run fixes this on **both** boxes so the virt/metal pair is like-for-like.

Cross-node estimator (Section 4.8). Cross-node runs used two hosts on the same subnet (private IPs). Cross-machine `CLOCK_MONOTONIC` stamps are not comparable and no PTP was deployed, so cross-node latency is reported as **RTT/2 measured on a single clock** (echo/reflector round trip halved) — a *different estimator* from the stamped one-way same-host numbers of Table 1, and not comparable 1:1 with them. ROS 2 discovery used `ROS_STATIC_PEERS` (AWS VPC provides no multicast). The ROS 2 UDP 64 KB cross-node cells used a reduced N = 10,000 per trial (not 100,000) because reliable-UDP fragmentation gives ≈10 ms round trips.

4. Results

4.1 The full picture

Table 1 reports one-way per-hop latency for all systems and configurations. Configurations marked **tuned** required expert, non-default setup: hand-written Fast-DDS XML profiles with data-sharing, or restructuring the application into a single process with intra-process composition.

Table 1. One-way per-hop latency (µs), mean of 10 trials ± 95% CI, N = 100,000 msgs/trial.

System (configuration)	Payload	p50	p90	p99	p99.9
Psyclone (default install)	64 B	38.1 ± 0.2	39.5 ± 0.5	50.6 ± 0.8	70.0 ± 3.7
Psyclone (default install)	1 KB	39.0 ± 0.0	40.4 ± 0.5	51.3 ± 0.3	69.0 ± 1.8
Psyclone (default install)	64 KB	85.1 ± 0.6	93.6 ± 0.7	101.7 ± 0.9	124.0 ± 2.0
NATS (default)	64 B	48.5 ± 1.9	56.8 ± 2.4	67.5 ± 2.9	131.7 ± 40.7
NATS (default)	1 KB	53.9 ± 1.4	63.5 ± 1.4	75.8 ± 1.5	185.9 ± 2.2
NATS (default)	64 KB	155.7 ± 6.0	258.0 ± 10.8	483.1 ± 14.9	761.4 ± 38.3
ROS 2 Fast-DDS UDP (default)	64 B	33.9 ± 0.5	39.2 ± 1.9	43.8 ± 2.1	50.4 ± 3.6
ROS 2 Fast-DDS UDP (default)	1 KB	34.3 ± 0.5	39.1 ± 1.8	44.3 ± 2.6	60.9 ± 24.5
ROS 2 Fast-DDS UDP (default)	64 KB	66.6 ± 0.9	71.8 ± 1.7	78.6 ± 3.3	87.7 ± 5.5
ROS 2 Fast-DDS SHM (**tuned**, XML data-sharing)	64 B	27.4 ± 0.2	28.8 ± 0.3	32.0 ± 1.6	43.8 ± 20.6
ROS 2 Fast-DDS SHM (**tuned**, XML data-sharing)	1 KB	27.7 ± 0.2	30.5 ± 2.3	32.9 ± 2.7	39.7 ± 8.5
ROS 2 Fast-DDS SHM (**tuned**, XML data-sharing)	64 KB	35.2 ± 0.9	38.8 ± 0.8	41.7 ± 0.8	64.2 ± 1.2
ROS 2 intra-process (**tuned**, single-process composition)	64 B	8.8 ± 0.1	11.6 ± 1.3	12.9 ± 1.8	16.4 ± 5.6
ROS 2 intra-process (**tuned**, single-process composition)	1 KB	8.6 ± 0.1	10.8 ± 0.3	11.7 ± 0.3	13.4 ± 1.9
ROS 2 intra-process (**tuned**, single-process composition)	64 KB	10.9 ± 0.6	12.6 ± 1.5	13.8 ± 2.0	16.1 ± 4.0
iceoryx v2.95.5 (native untyped API, loaned chunks, zero-copy)	64 B	6.2 ± 0.0	8.1 ± 0.1	9.2 ± 0.2	9.8 ± 0.4
iceoryx v2.95.5 (native untyped API, loaned chunks, zero-copy)	1 KB	6.2 ± 0.0	8.1 ± 0.0	9.2 ± 0.5	57.0 ± 107.2 †
iceoryx v2.95.5 (native untyped API, loaned chunks, zero-copy)	64 KB	12.3 ± 0.2	14.4 ± 0.2	15.4 ± 0.3	16.3 ± 0.6
eCAL 6.1.1 (default, SHM transport)	64 B	7.0 ± 0.1	9.2 ± 0.2	10.0 ± 0.2	11.0 ± 0.2
eCAL 6.1.1 (default, SHM transport)	1 KB	7.2 ± 0.2	10.6 ± 1.6	11.7 ± 2.1	13.2 ± 3.0
eCAL 6.1.1 (default, SHM transport)	64 KB	25.9 ± 1.2	28.7 ± 1.3	31.2 ± 0.9	39.1 ± 2.4
ZeroMQ 4.3.5 `inproc` (PAIR, 2 threads, 1 process)	64 B	8.1 ± 0.1	10.7 ± 1.0	11.8 ± 1.2	29.9 ± 34.1 †
ZeroMQ 4.3.5 `inproc` (PAIR, 2 threads, 1 process)	1 KB	8.4 ± 0.0	10.7 ± 0.3	11.7 ± 0.4	13.9 ± 3.0
ZeroMQ 4.3.5 `inproc` (PAIR, 2 threads, 1 process)	64 KB	26.8 ± 0.1	30.9 ± 0.3	34.0 ± 0.6	38.3 ± 2.1

† In each case one of 10 trials carried a ≈500 µs-class p99.9 spike while the other nine sat at their usual level; the bare-metal replication (Section 4.6) shows both spikes vanish on metal, confirming hypervisor/shared-tenancy jitter rather than transport behaviour.

Note on the peer-class rows (iceoryx, eCAL, ZeroMQ `inproc`; measured 2026-07-21 on the same instance, kernel, and methodology as the rest of the table): these are zero/minimal-copy IPC specialists — a transport and nothing above it: no orchestration layer, no broker, no module runtime or dispatch layer — and ZeroMQ `inproc` additionally gives up process isolation (two threads in one process, like ROS 2 intra-process). They are expected to beat an orchestration substrate on raw per-hop latency, and they do: all three sit well below

Psyclone's default install, with iceoryx the fastest cross-process transport measured on this box. Figures 1–4 predate these rows and plot the original five configurations.

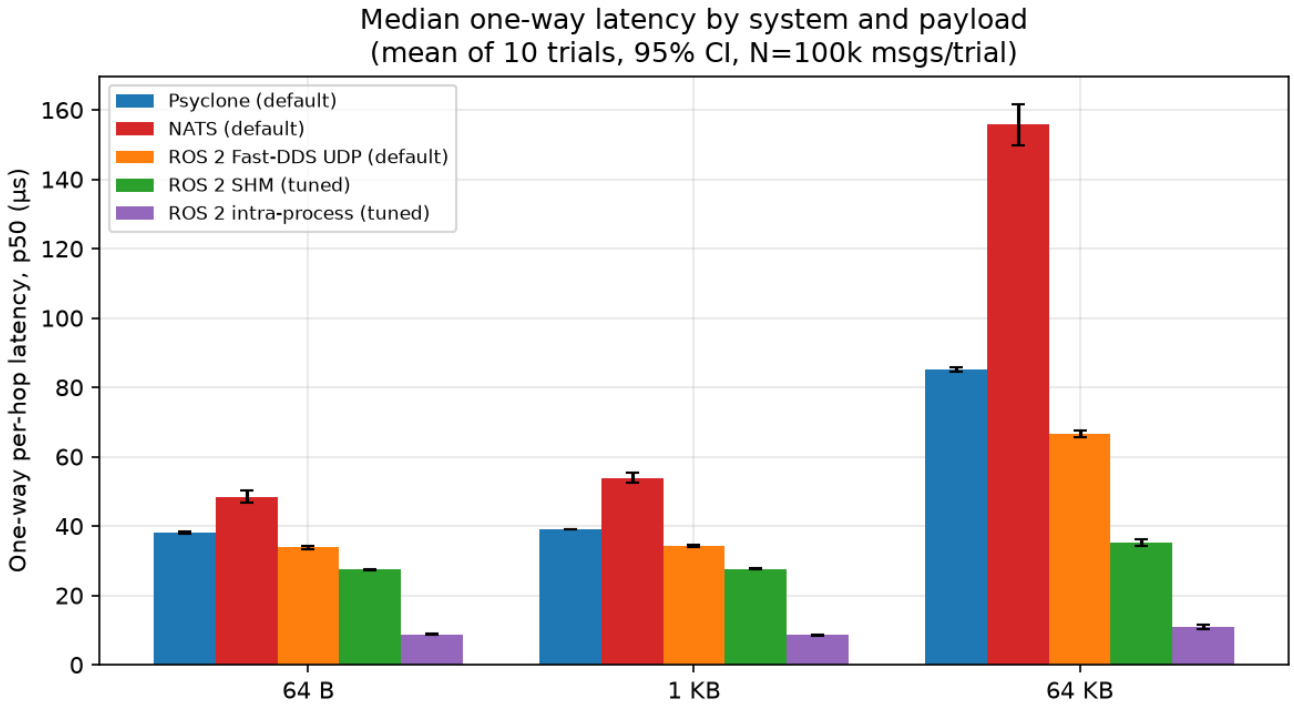


Figure 1. Median (p50) one-way per-hop latency by system and payload; error bars are 95% CIs over 10 trials. Tuned ROS 2 configurations win on raw median; Psyclone leads the zero-tuning field against NATS and is close to default ROS 2.

Stated plainly: **Psyclone's default install is not the fastest thing in the table.** The dedicated IPC specialists lead outright — iceoryx at 6.2 µs p50 (small payloads) and 12.3 µs at 64 KB, eCAL and ZeroMQ `inproc` in the 7–8 µs class — and **tuned ROS 2 wins the median-latency contest among the full-middleware systems**: Fast-DDS SHM with data-sharing runs at ≈27–35 µs across the payload sweep, and intra-process composition runs at ≈9–11 µs. Psyclone's default install runs at 38–85 µs. Anyone quoting this paper should quote those sentences along with the ones that follow.

4.2 Out of the box vs manually tuned — the comparison that matters on day one

The five columns of Figure 2 divide into two groups. **Default installs** — what an engineer gets after following each project's quick-start, with zero tuning: Psyclone, NATS, and ROS 2 over its forced-UDP baseline. **Tuned configurations** — what an engineer gets after becoming (or hiring) a DDS expert: hand-authored Fast-DDS XML profiles enabling SHM + data-sharing with loaned messages, or restructuring the entire application into a single process to exploit intra-process composition.

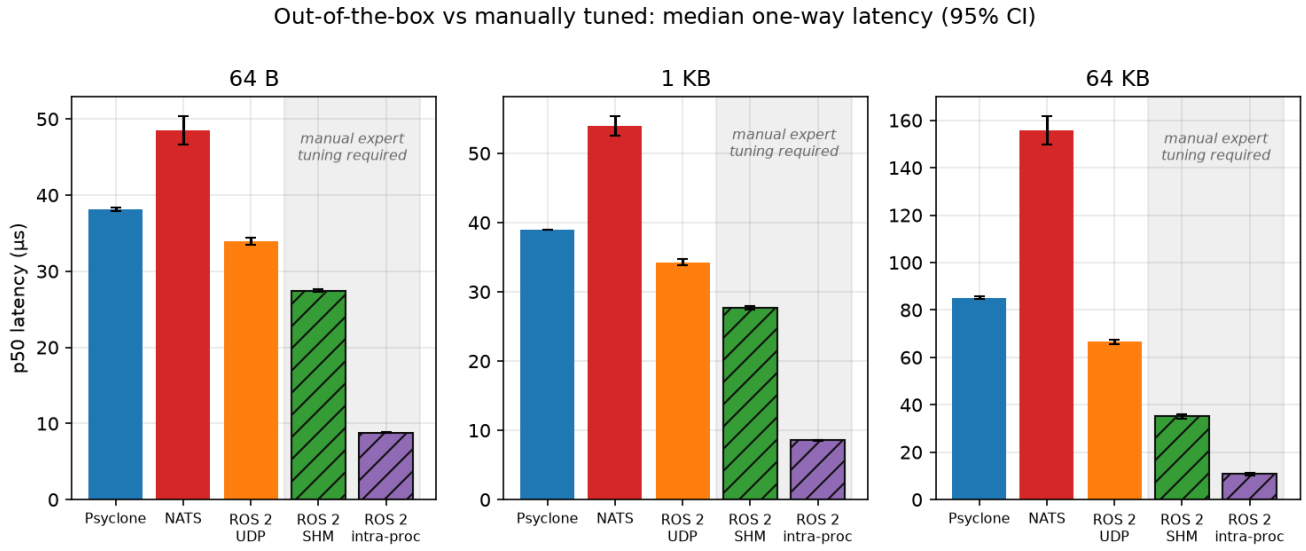


Figure 2. Median latency, default installs vs manually tuned configurations (hatched, shaded region), per payload. The two configurations that beat Psyclone both sit in the "manual expert tuning required" region.

In the zero-tuning group, the picture is consistent across payloads:

- **Psyclone beats NATS at every payload size** — by 10 µs at 64 B (38.1 vs 48.5), 15 µs at 1 KB, and 71 µs at 64 KB on the median, with far larger gaps in the tails.
- **Psyclone is within ≈4 µs of default ROS 2** at 64 B and 1 KB (38.1 vs 33.9; 39.0 vs 34.3) — a gap comparable to a few cache misses and smaller than the cross-system differences in tail behaviour. At 64 KB default ROS 2 leads Psyclone by ≈19 µs on the median (66.6 vs 85.1), while Psyclone's p99.9 remains 6× tighter than NATS's (124 vs 761 µs).

The two configurations that decisively beat Psyclone required work that a default user does not do, and each gives something up. The **SHM/data-sharing** configuration requires writing and injecting Fast-DDS XML profiles (`FASTRTPS_DEFAULT_PROFILES_FILE` , `RMW_FASTRTPS_USE_QOS_FROM_XML`), understanding DDS QoS compatibility with data-sharing, and using the LoanedMessage API in application code. The **intra-process** configuration is the starkest case: it wins by *collapsing the distributed system into one process*. At ≈9 µs it is genuinely fast — but it has abandoned process isolation, independent module deployment, crash containment, and multi-host distribution. It wins by moving into the same restricted single-memory-space regime that Psyclone occupies natively — while giving up the runtime reconfigurability Psyclone provides there (Section 6).

The relevant engineering question is rarely **"what is the fastest configuration an expert can construct" but "what latency does the team get on day one, and what does it cost to keep it as the system grows."** On that question the data are unambiguous: among the orchestration-capable systems, Psyclone's default install is the best all-round zero-tuning performer measured, and the only one whose graph can be re-wired at runtime. (The IPC specialists of Section 4.1 are also fast with no tuning — eCAL runs its SHM transport by default — but they supply a transport, not an orchestration layer, and are compared on that understanding.)

4.3 Tail latency and predictability

For interactive systems, the tail — not the median — governs user experience and real-time deadline compliance. Here Psyclone's results are strongest, and they are not an artifact of tuning: they come from the default install.

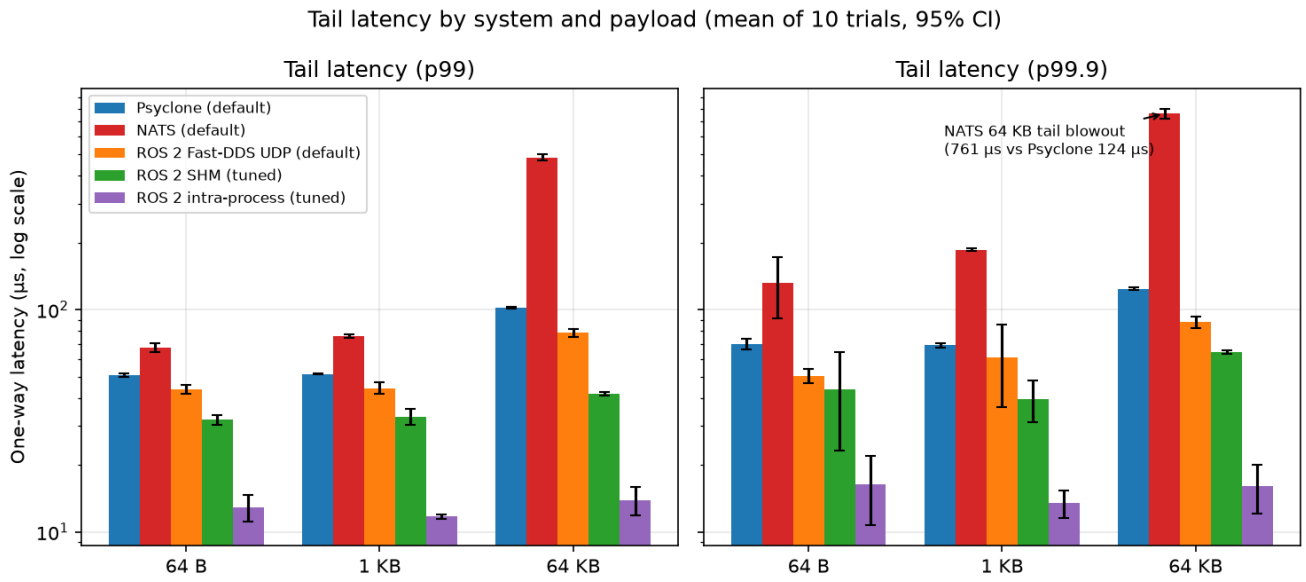


Figure 3. Tail latency (p99, p99.9) by system and payload, log scale, with 95% CIs. Psyclone's p99.9 stays within ≈1.8× of its own median at every payload; NATS's 64 KB p99.9 reaches 761 µs.

Three observations:

- **Psyclone's tails are tightly bounded and stable.** p99.9 ≤ 70 µs at small payloads and 124 ± 2.0 µs at 64 KB — a p99.9/p50 ratio of ≈1.5–1.8× across the sweep. Its confidence intervals are among the smallest in the study (e.g. 1 KB p50 CI of ±0.0 µs at the facility's 1 µs resolution), i.e. run-to-run behaviour is highly reproducible.
- **NATS's tails blow out with payload.** At 64 KB, p99 is 483 µs and p99.9 is 761 µs — a ≈5× multiple of its own median — even at a deliberately gentle 2,000 msg/s offered load. At 64 B its p99.9 CI of ±40.7 µs shows substantial run-to-run tail instability.
- **Tuned ROS 2 has excellent tails too** — intra-process p99.9 stays under ≈16 µs — though several of its tail estimates carry wide CIs (e.g. SHM 64 B p99.9 = 43.8 ± 20.6 µs), indicating trial-to-trial tail variance that Psyclone does not exhibit.
- **The IPC specialists have very tight tails as well.** iceoryx's p99.9 sits at 9.8–16.3 µs and eCAL's at 11.0–39.1 µs across the sweep. Two isolated virtualised cells (ZeroMQ 64 B p99.9 = 29.9 ± 34.1 µs; iceoryx 1 KB p99.9 = 57.0 ± 107.2 µs) were each driven by a single jitter-spiked trial out of 10 — and the bare-metal replication (Section 4.6) shows both spikes vanish entirely on metal (12.6 ± 0.3 and 6.8 ± 1.2 µs), confirming those outlier trials were environmental, not transport behaviour.

For a fabric intended to sit under real-time AI, "predictably 50 µs" is frequently worth more than "usually 27 µs, occasionally several times that."

4.4 Payload scaling

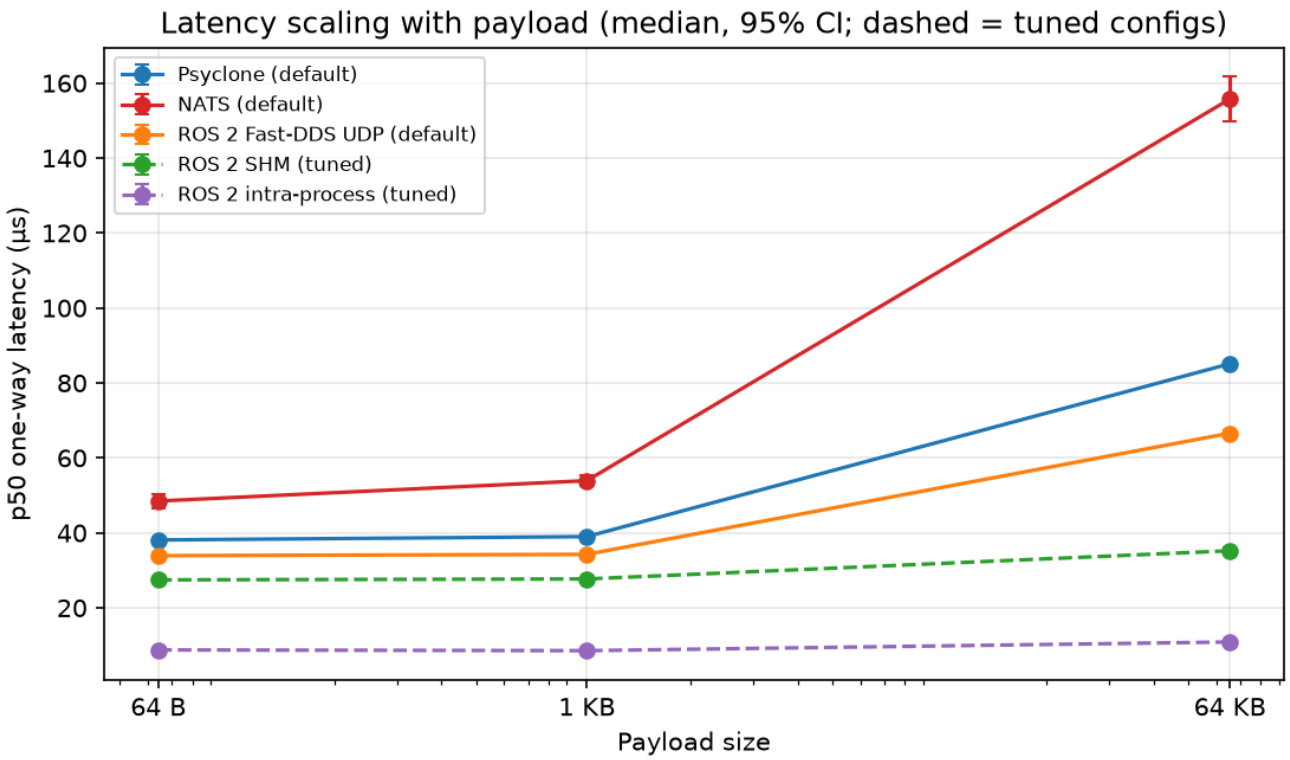


Figure 4. Median latency vs payload size (log-x). Psyclone and default ROS 2 are flat to 1 KB with a modest 64 KB rise; NATS degrades steeply; tuned zero-copy configurations (dashed) are nearly payload-invariant.

Psyclone's median is flat from 64 B to 1 KB (38.1 → 39.0 µs; +2%), confirming that small-payload latency is dominated by fixed per-hop cost. At 64 KB it rises to 85.1 µs — the cost of copying a 64 KB buffer — where the zero-copy tuned configurations stay flat (35.2 and 10.9 µs), which is precisely what zero-copy buys and why it is worth its configuration cost in bulk-transfer-heavy deployments. NATS pays payload costs on every socket traversal and degrades fastest (48.5 → 155.7 µs median; 761 µs p99.9).

4.5 Throughput: an honest note

This campaign measured latency under paced, non-saturating load; it did not measure saturating throughput. Two facts should be stated to preempt misreading. First, Psyclone's ping-pong cadence (≈8.5–9.5k msg/s) is a *single-message-in-flight round-trip artifact*, not a throughput ceiling — but we have not yet measured Psyclone's saturating multi-publisher throughput, and until we do we make no throughput claims for it. Second, NATS's saturating fan-out throughput (millions of msg/s on this class of hardware) is the regime NATS is engineered for and nothing in this paper disputes it: at high-fan-out, multi-host, durability-required workloads, a broker is the right tool. A saturating-throughput measurement for Psyclone is committed future work (Section 7).

4.6 Bare-metal replication

To close the hypervisor-jitter caveat of Section 7, the campaign was replicated on **bare metal**: an AWS `z1d.metal` (48 vCPU = 2 × 12-core Xeon Platinum 8151 @ 3.4 GHz, 4.0 GHz boost, SMT on, 2 NUMA nodes), us-east-1c, Ubuntu 26.04, kernel 7.0.0-1006-aws — **the identical kernel build to the virtualised paper box** — with the same harness sources, builds/tags, methodology, pinning, and statistical treatment. Bare metal exposes the controls the virtualised instance hides: governor set to `performance` on all cores,

turbo on (cores verified at 4.0 GHz under load), and — for the definitive sweep — **C-states pinned to C0** via `/dev/cpu_dma_latency=0`.

A bare-metal-only finding that must be disclosed: idle C-states dominate paced-load latency on real hardware. With default idle states, at 5,000 msg/s (200 µs gaps) the pinned cores drop into deep C-states between messages and every message pays the wake-up: ZeroMQ `inproc` p50 measured 23.9 µs (vs 8.1 virtualised) and eCAL 18.5 µs (vs 7.0). Pinning `/dev/cpu_dma_latency=0` restored p50 to 8.2 µs instantly (spot-checked before/after). The hypervisor on the virtualised box was masking this effect entirely; anyone replicating these numbers on real hardware must control C-states.

Table 2. Virtualised (`z1d.2xlarge`) vs bare metal (`z1d.metal`, C-states off). One-way per-hop latency, µs, mean of 10 trials ± 95% CI, N = 100,000 msgs/trial.

System	Payload	p50 virt → metal	p99 virt → metal	p99.9 virt → metal	p99.9 delta
iceoryx v2.95.5 (native, zero-copy)	64 B	6.2 → **5.7**	9.2 → **6.1**	9.8 → **6.4 ± 0.2**	−35%
iceoryx	1 KB	6.2 → **5.6**	9.2 → **6.5**	57.0 ± 107 † → **6.8 ± 1.2**	**spike gone**
iceoryx	64 KB	12.3 → **7.8**	15.4 → **8.7**	16.3 → **9.1 ± 1.4**	−44%
eCAL 6.1.1 (default SHM)	64 B	7.0 → **7.3**	10.0 → **8.7**	11.0 → **9.2 ± 1.6**	−16%
eCAL	1 KB	7.2 → **7.6**	11.7 → **8.1**	13.2 → **8.6 ± 0.1**	−35%
eCAL	64 KB ‡	25.9 → **18.4**	31.2 → **19.1**	39.1 → **20.3 ± 0.3**	−48%
ZeroMQ 4.3.5 inproc	64 B	8.1 → **8.2**	11.8 → **10.2**	29.9 ± 34 † → **12.6 ± 0.3**	−58%, CI ×100 tighter
ZeroMQ inproc	1 KB	8.4 → **8.5**	11.7 → **10.9**	13.9 → **13.0 ± 0.5**	−7%
ZeroMQ inproc	64 KB	26.8 → **19.2**	34.0 → **20.1**	38.3 → **22.5 ± 0.1**	−41%
Psyclone (default install)	64 B	38.1 → **48.5**	50.6 → **57.6**	70.0 → **76.8 ± 0.8**	+10% (see below)
Psyclone	1 KB	39.0 → **49.0**	51.3 → **58.2**	69.0 → **78.2 ± 1.3**	+13%
Psyclone	64 KB	85.1 → **83.1**	101.7 → **96.2**	124.0 → **115.9 ± 1.7**	−7%

† virtualised: one of 10 trials had a ≈500 µs-class p99.9 spike (hypervisor/shared-tenancy jitter suspected — now confirmed). ‡ metal eCAL 64 KB: latency percentiles valid but computed over the first ≈48 k received msgs/trial (a trimmed subscriber safety-timeout truncated the 50 s-long 64 KB trials); values are extremely stable across all 10 trials (p50 CI ± 0.06 µs). Marked measured, reduced window.

Table 2b. ROS 2 on bare metal. The ROS 2 leg was completed in a later Phase-2 run on a fresh `z1d.metal` (**i-0584eddbbebe775e5**, 2026-07-22 21:59–22:51 UTC; same Ubuntu 26.04 / kernel 7.0.0-1006-aws, same latbench harness as the paper's p0-rerun, ROS 2 lyrical, `rmw_fastrtps_cpp` / Fast-DDS 9.4.8, governor `performance`, turbo on, C-states off, pub core 2 / sub core 3, paced 5,000 msg/s at 64 B/1 KB and 2,000 msg/s at 64 KB). One-way per-hop latency, µs, mean of 10 trials ± 95% t-CI, N = 100,000 msgs/trial.

Config	Payload	p50 virt → metal	p99 virt → metal	p99.9 virt → metal
ROS 2 UDP (default)	64 B	33.9 → **46.5 ± 0.1**	43.8 → **53.0 ± 1.4**	50.4 → **60.7 ± 2.8**
ROS 2 UDP	1 KB	34.3 → **46.9 ± 0.1**	44.3 → **54.8 ± 0.6**	60.9 → **61.3 ± 1.3**
ROS 2 UDP	64 KB §	66.6 → **71.2 ± 1.0**	78.6 → **83.8 ± 2.9**	87.7 → **580.2 ± 1095.9 §**

Config	Payload	p50 virt → metal	p99 virt → metal	p99.9 virt → metal
ROS 2 SHM (tuned)	64 B	27.4 → **39.3 ± 0.2**	32.0 → **41.5 ± 1.7**	43.8 → **43.4 ± 1.8**
ROS 2 SHM (tuned)	1 KB	27.7 → **39.5 ± 0.1**	32.9 → **40.9 ± 1.3**	39.7 → **42.5 ± 1.3**
ROS 2 SHM (tuned)	64 KB	35.2 → **36.6 ± 0.6**	41.7 → **45.4 ± 2.1**	64.2 → **69.0 ± 2.6**
ROS 2 intra-process (tuned)	64 B	8.8 → **10.3 ± 0.1**	12.9 → **12.1 ± 1.5**	16.4 → **12.4 ± 1.7**
ROS 2 intra-process (tuned)	1 KB	8.6 → **10.1 ± 0.0**	11.7 → **11.4 ± 1.0**	13.4 → **11.9 ± 1.5**
ROS 2 intra-process (tuned)	64 KB	10.9 → **10.4 ± 0.1**	13.8 → **11.0 ± 0.1**	16.1 → **11.4 ± 0.1**

§ ROS 2 UDP 64 KB p99.9 on metal: **one of the 10 trials spiked to p99.9 = 4,940 µs while the other nine sat at ≈94–103 µs**; the enormous ±1,095.9 µs CI is that single spike. It is disclosed as measured; the p50/p99 columns for the same cell were unaffected.

Five verdict points, reported without adjustment:

1. **Every p99.9 tail in the peer class tightened or held on metal, most by 35–58%.** The two pathological virtualised cells — ZeroMQ 64 B p99.9 = 29.9 ± 34.1 and iceoryx 1 KB p99.9 = 57.0 ± 107.2, each driven by a single jitter-spiked trial — **completely disappeared** on metal (12.6 ± 0.3 and 6.8 ± 1.2). This directly confirms the paper's attribution of those outlier trials to hypervisor/shared-tenancy jitter, not to the transports.
2. **Confidence intervals collapsed**: the worst-case p99.9 CI in the trio went from ±107 µs (virtualised) to ±1.6 µs (metal). Cross-trial variance on the virtualised box was dominated by the environment, not the systems — exactly what Section 7 conjectured.
3. **Medians and rank order are essentially unchanged** (iceoryx < eCAL ≈ zmq inproc ≪ Psyclone default): the virtualised medians were trustworthy. 64 KB medians improve ≈25–40% on metal (memory bandwidth / 4.0 GHz sustained boost).
4. **Psyclone default, honestly:** its 64 KB tails tightened (124 → 115.9 µs p99.9) and all its CIs are tight, but its small-payload medians measured **≈10 µs slower on metal** (48.5 vs 38.1 µs at 64 B). We report this without spin. Plausible contributors: Psyclone's ping ring runs 4+ threads time-slicing on the 2 pinned cores, and the scheduling/SMT topology of the 2-socket 24-core part differs from the 8-vCPU virtual slice; the framework's 1 µs timestamp resolution is also disclosed in Section 3.2. Its *predictability* claim — tight, sub-100 µs small-payload tails with ±≤1.7 µs CIs — is confirmed on metal.
5. **ROS 2 shows the same small-payload metal slowdown (Table 2b).** UDP p50 46.5 vs 33.9 µs virtualised, SHM 39.3 vs 27.4, intra-process 10.3 vs 8.8 — consistent with the small-payload metal slowdown seen for Psyclone in verdict 4: a property of the 2-socket metal part under C-states-off paced load, not of any single system, since it appears across independently implemented harnesses and transports. At 64 KB the picture matches the virtualised ordering: intra-process stays flat (10.4 µs — zero-copy pointer hand-off holds), SHM lands at 36.6 vs 35.2 virtualised, UDP 71.2 vs 66.6. Tuned intra-process tails on metal are the tightest ROS 2 has shown anywhere in this campaign (p99.9 ≤ 12.4 µs across all payloads); the single udp-64 KB spiked trial (§) is the one blemish in the ROS 2 metal data.

Metal coverage status (disclosed). Two earlier attempts at the ROS 2 metal leg failed — the first instance was externally terminated mid-run and a follow-up was likewise terminated 25 minutes after launch — but a later single-box Phase-2 run (**i-0584eddbbebe775e5**, 2026-07-22) completed the ROS 2 leg cleanly; its results are Table 2b above. **NATS on metal was initially measured invalidly**: the Go pub/sub at default `GOMAXPROCS=48` pinned to one core thrashed the runtime scheduler, producing millisecond-class self-inflicted queueing (raw data retained and marked invalid). That gap is **now closed**: a `GOMAXPROCS=8` re-run on both boxes (Section 4.7) produced valid NATS figures, and metal NATS turns out to be essentially identical to

virtualised NATS — the earlier "NATS is slow on metal" artifact was entirely the GOMAXPROCS misconfiguration.

4.7 Fixed NATS re-run and rmw_iceoryx_cpp

Two additions close gaps disclosed in earlier revisions. Same methodology as everywhere else in the paper (mean ± 95% CI over 10 trials, N = 100,000 msgs/trial, µs); campaign statistics in the run's `ci_summary.json`.

NATS, re-measured with the GOMAXPROCS fix (both boxes). The `natslat` harness was re-run at `GOMAXPROCS=8` (Section 3.5) on the virtualised paper box (`z1d.2xlarge` i-09bc1e841ea1617ea) and on bare metal (`z1d.metal` i-0b7c4d295acbf0bed, governor `performance`, turbo on, C-states off).

Table 3. NATS (default, loopback TCP through `nats-server`), GOMAXPROCS=8 harness, virtualised vs bare metal. One-way per-hop latency, µs, mean of 10 trials ± 95% CI, N = 100,000 msgs/trial.

Box	Payload	p50	p99	p99.9
virt (z1d.2xlarge)	64 B	57.14 ± 0.25	75.41 ± 0.74	796.9 ± 368.44
virt	1 KB	62.70 ± 0.13	132.39 ± 50.27	2700.07 ± 80.38
virt	64 KB	169.31 ± 1.55	5380.92 ± 170.29	9944.28 ± 333.41
metal (z1d.metal)	64 B	57.91 ± 0.30	75.74 ± 0.73	1065.69 ± 342.45
metal	1 KB	64.04 ± 0.15	97.04 ± 4.00	2645.09 ± 84.76
metal	64 KB	159.66 ± 3.22	5331.65 ± 267.92	10424.61 ± 458.47

Headline: **metal ≈ virt** (64 B p50 57.9 vs 57.1 µs) — the earlier "NATS slow on metal" result was *entirely* the GOMAXPROCS=48-pinned-to-one-core artifact. Footnote on Table 1: the paper's existing Table 1 NATS virtualised numbers (48.5 / 53.9 / 155.7 µs p50) came from the older `nats.go` one-way harness; the fixed-harness virtualised re-run above (57.1 µs p50 at 64 B) is slightly higher than Table 1's original (48.5) due to harness and GOMAXPROCS differences. Both are disclosed and retained; they are not identical and we do not pretend they are — Table 1's NATS row is left as originally measured, and this table carries the validated like-for-like virt/metal pair.

rmw_iceoryx_cpp (zero-copy RMW for ROS 2). Build succeeded with two disclosed patches, and all caveats follow. `rmw_iceoryx` has no lyrical branch, so we used its jazzy-era commit `1d03b7e`; it required (1) an `ament_target_dependencies` compatibility macro, and (2) **iceoryx v2.0.6** — it does *not* build against iceoryx v2.95.x, so the native-iceoryx peer-class rows (v2.95.5, Table 1) and this RMW row (v2.0.6) use **different iceoryx versions** (which is why it gets its own table rather than a Table 1 row). Built with `-DBUILD_TESTING=OFF`; some newer rmw symbols were unresolved at runtime (rclcpp degrades gracefully; the data path is unaffected). Loaned messages / the zero-copy path were enabled, with RouDi v2.0.6 core4.

Table 4. rmw_iceoryx_cpp (ROS 2 over iceoryx v2.0.6, zero-copy loaned messages), virtualised and bare metal. One-way per-hop latency, µs, mean of 10 trials ± 95% CI, N = 100,000 msgs/trial.

Box	Payload	p50	p99	p99.9
virt (z1d.2xlarge)	64 B	10.47 ± 0.14	13.64 ± 1.51	15.36 ± 2.35
virt	1 KB	10.81 ± 0.20	14.26 ± 1.92	15.98 ± 2.38
virt	64 KB	17.83 ± 0.35	20.68 ± 0.68	28.08 ± 7.73

Box	Payload	p50	p99	p99.9
metal (z1d.metal)	64 B	19.50 ± 0.52	21.44 ± 1.37	21.95 ± 1.44
metal	1 KB	19.80 ± 0.44	23.72 ± 1.51	24.38 ± 1.53
metal	64 KB	25.80 ± 0.32	26.94 ± 0.15	27.42 ± 0.11

rmw_iceoryx is the fastest RMW measured: virtualised 10.5 µs p50 at 64 B — ≈2.6× faster than tuned Fast-DDS SHM's 27.4 µs, close to ROS 2 intra-process (8.8 µs) and native iceoryx (6.2 µs) — while keeping full ROS 2 semantics and process isolation. This closes the "most interesting missing competitor" future-work item flagged in Sections 2 and 7 of earlier revisions. And, keeping the paper's honesty standard: **rmw_iceoryx is faster than Psyclone's default install at every payload size on both boxes.**

4.8 Cross-node (multi-host) latency

Everything above is single-host. This section adds the first cross-node measurements: two hosts on the same subnet (private IPs), virtualised pair = 2× `z1d.2xlarge`, metal pair = 2× `z1d.metal`.

These are RTT/2 single-clock one-way *estimates*, not stamped one-way measurements (cross-machine `CLOCK_MONOTONIC` is not comparable; no PTP was deployed — Section 3.5). That is a *different estimator* from Table 1's stamped same-host one-way numbers, and the two tables must **not** be compared 1:1. ROS 2 discovery used `ROS_STATIC_PEERS` (AWS VPC has no multicast).

Table 5. Cross-node one-way latency *estimate* (RTT/2, single clock), p50, µs, mean of 10 trials ± 95% CI, N = 100,000 msgs/trial (except ROS2-udp 64 KB: N = 10,000 — see below). Not comparable 1:1 with Table 1 (different estimator).

System	Box pair	64 B	1 KB	64 KB
NATS	virt (2× z1d.2xlarge)	132.49 ± 0.06	132.77 ± 0.50	313.07 ± 7.73
ROS 2 UDP	virt	125.52 ± 9.79	131.06 ± 4.67	2760.21 ± 1835.75 ¶
NATS	metal (2× z1d.metal)	125.09 ± 0.08	128.32 ± 0.25	298.88 ± 0.92
ROS 2 UDP	metal	118.98 ± 2.91	126.05 ± 3.94	260.82 ± 3.79
eCAL	—	N/A	N/A	N/A
Psyclone	—	N/A	N/A	N/A
iceoryx / rmw_iceoryx / ZeroMQ `inproc` / ROS 2 intra / ROS 2 SHM	—	N/A	N/A	N/A

¶ **ROS 2 UDP 64 KB pathology, reported plainly.** The 64 KB ROS 2 UDP cells used N = 10,000 per trial (not 100,000) because reliable-UDP fragmentation gives ≈10 ms round trips. On the virtualised pair the 64 KB run had a **catastrophic tail**: p50 2.76 ms with p99.9 ≈ **1.2 s**, caused by reliable-UDP fragmentation reassembly under loss — a real pathology of fragmenting 64 KB reliable messages over UDP on this network, not a harness artifact. The metal pair was far better behaved (260.82 µs p50).

N/A rows, with reasons stated explicitly: **eCAL** cross-node is N/A because its cross-node transport needs UDP multicast and AWS VPC provides none — an environment limit, not an eCAL defect. **iceoryx, rmw_iceoryx, ZeroMQ `inproc`, ROS 2 intra-process, and ROS 2 SHM** are single-host by design. **Psyclone is N/A**

because its external/cross-node attach path is non-functional on Linux in 2.1.0 (already disclosed in Section 6.3); it was not measured, and no number is faked or extrapolated for it.

On what *was* measurable cross-node, NATS and default ROS 2 UDP sit in the same ≈119–133 µs class at small payloads, with NATS far more stable at 64 KB on the virtualised pair.

5. Analysis

Why Psyclone's defaults are fast. Three mechanisms: (i) an in-memory shared-space transport — no kernel socket traversal on the critical path; (ii) no per-message serialization tax (no CDR encode/decode); (iii) a thin dispatch layer whose trigger batching amortises wake-up costs under bursts. The result is a low *and low-variance* fixed per-hop cost — visible in both the flat 64 B→1 KB medians and the small CIs.

Why tuned ROS 2 is faster. Fast-DDS data-sharing and intra-process composition eliminate the remaining copies entirely (loaned buffers; pointer hand-off). Zero-copy beats one-copy; at 64 KB this is decisive (35.2 / 10.9 vs 85.1 µs). Psyclone currently copies payloads on its critical path; a zero-copy large-payload path is an identified optimisation target, and the 64 KB column shows exactly what it is worth.

Why the IPC specialists are fastest of all. iceoryx, eCAL, and ZeroMQ `inproc` are transports and nothing more: zero-copy loaned chunks (iceoryx) or a single copy into shared memory / an in-process message (eCAL, zmq), with no orchestration layer, no broker, and no module dispatch/scheduling machinery on the critical path. iceoryx's zero-copy path is nearly payload-invariant (6.2 → 12.3 µs from 64 B to 64 KB) while retaining process isolation — which is why it essentially matches single-process ROS 2 intra-process. An orchestration substrate carrying a context-routing and dispatch layer per hop should be expected to trail them, and Psyclone does.

Why NATS trails everywhere on-node. Two full socket traversals per hop plus broker scheduling put a floor under its latency and make it payload-sensitive. This is architecture, not implementation deficiency — and it purchases capabilities (multi-host fan-out, durability, subject-based multi-tenancy) that neither Psyclone nor single-node ROS 2 configurations provide.

What the latency deltas do and do not buy. We do not claim that saving transport microseconds "funds System-2 reasoning": over a dozen pipeline hops, the median difference between Psyclone and any measured competitor is well under a millisecond, which does not fund an extra LLM call or retrieval (those cost 10^2–10^4 ms). The defensible claims are about **jitter and rate**: (i) in high-rate inner loops (100 Hz audio frames, sensor fusion, barge-in paths crossing many hops), per-hop tails multiply into deadline misses, and the fabric with a bounded p99.9 (124 µs) behaves qualitatively differently from one that excursions to 761 µs; (ii) predictable per-hop cost makes end-to-end latency *budgetable* — an engineer can allocate a deadline across a 12-hop pipeline with ±CI-level confidence; and (iii) what actually enables adaptive System-1/System-2 orchestration is not reclaimed microseconds but **runtime reconfigurability** — the ability to insert, remove, or re-route deliberative stages while the reflexive loop keeps running (Section 6). Latency parity is what makes that reconfigurability free to use; it is not itself the payoff.

6. The Versatility Thesis: Capabilities the Latency Table Cannot Show

The benchmark establishes that Psyclone is latency-competitive without tuning. This section details the capabilities that motivated building it — the properties that make it an *orchestration substrate* rather than a message pipe. Each is a shipping capability of Psyclone 2.1.0; roadmap items are explicitly excluded (Section 6.5).

6.1 Hierarchical-context routing: topology as a runtime variable

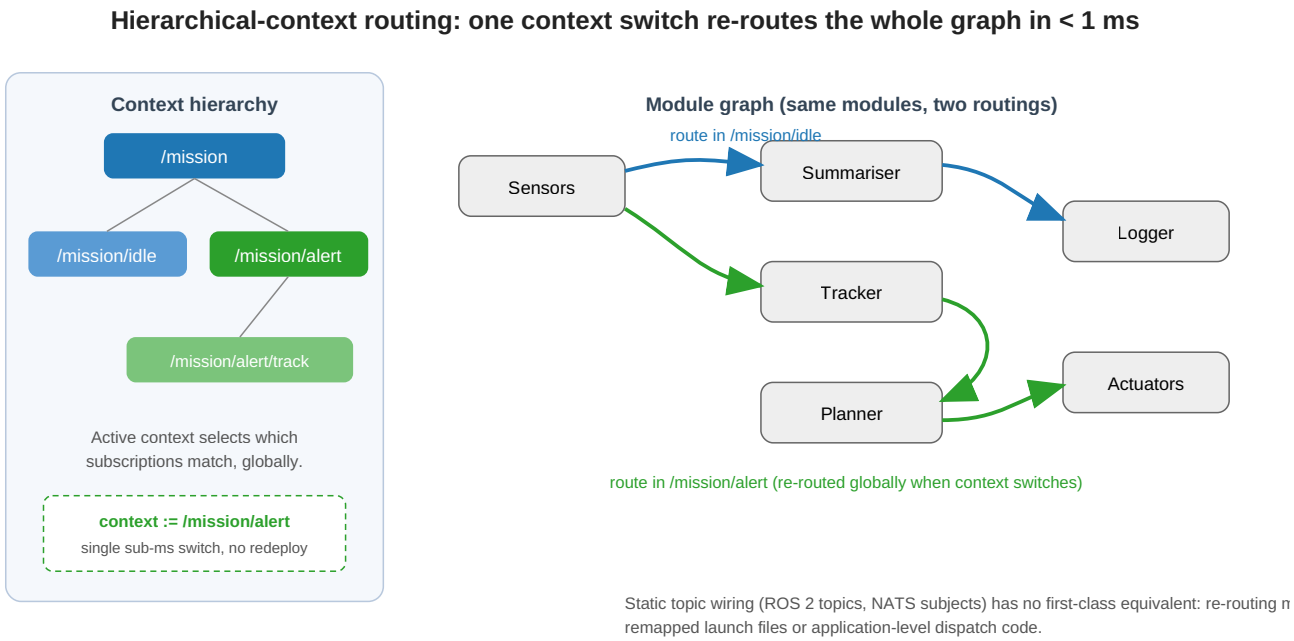


Figure 5. Context-addressed routing. A single sub-millisecond context switch (e.g. `/mission/idle` → `/mission/alert`) re-routes every affected stream across the whole graph. Static topic wiring has no first-class equivalent.

In topic-wired systems, "who receives what" is fixed by strings chosen at design time; changing dataflow in response to changing conditions means either pre-plumbing every mode into every subscriber (dispatch logic smeared across the codebase) or restarting/remapping nodes. Psyclone inverts this: subscriptions are declared against a **hierarchy of contexts**, and the active context determines which subscriptions match, globally. Mode changes — idle→alert, wake→conversation, nominal→degraded — become one sub-millisecond routing operation instead of a redeploy. Because contexts are hierarchical, subscriptions can be as broad (`/mission`) or as narrow (`/mission/alert/track`) as the module needs, and mode structure composes instead of exploding combinatorially. **Measured (Section 6.6): a context switch propagates in 91 ± 3 µs and a message posted immediately behind the switch is delivered on the new route in 112 ± 3 µs (≈0.11 ms), with 3000/3000 messages correctly routed and zero misroutes across 10 trials** — the reroute is effectively atomic with respect to traffic posted right after it. (This was measured on a small single-node graph with two contexts; propagation across a large or multi-node graph is not yet characterised.)

6.2 Trigger batching and the multi-input join

Per-message dispatch pays a fixed wake-up/scheduling cost per delivery. Psyclone lets a single module declare **multiple named triggers that feed one crank through one queue**, so a synchronisation point — a module that should act only when *all* of its inputs are present — lives inside one module with no extra processes or hops. We built exactly this as a running example (not folded into the latency benchmark): a sink subscribing to three streams that fires only when all three of a round are present.

Measured (Section 6.6): it fired exactly 1000 times for 1000 complete triples across 3000 input messages in every one of 10 trials (zero incomplete fires), with third-input→activation latency of 93.5 ± 5.7 µs (p99 133 µs). ROS 2 requires a separate `message_filters` synchroniser with approximate-time heuristics; NATS requires an application-side aggregator service.

Honest scope. This all-inputs gate is expressed in ≈15 lines of module (crank) code over Psyclone's multi-trigger subscription, *not* a declarative framework primitive: Psyclone's PsySpec grammar contains a `<triggergroup>` element and group-policy fields, but in the shipping build these are parsed-but-stubbed — the scheduler still activates the crank once per arriving message, and the join condition is evaluated in the module body. The value Psyclone adds is that the gate needs no synchroniser node and every message carries a framework timestamp; the value it does *not* yet add is a declarative join operator. Separately, this batching mechanism was **not** exercised by the Section 4 latency benchmark (which ran one message in flight), so it is not part of that section's tail-latency result; the tight tails there come from the shared-memory transport and thin control layer.

6.3 Ephemeral nodes and modules: compute that follows the workload

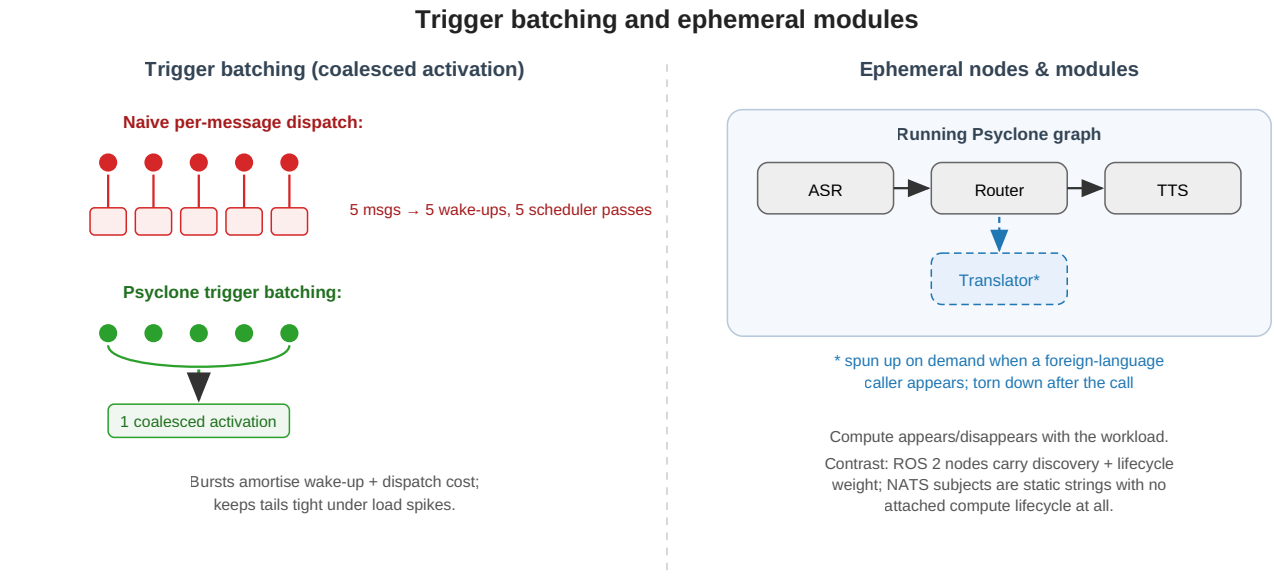


Figure 7. Left: coalesced activation vs naive per-message dispatch. Right: a translator module instantiated on demand for a foreign-language caller and torn down afterwards.

Psyclone modules and nodes can be created and destroyed while the system runs, attaching to the in-memory graph on instantiation. This makes per-request or per-situation compute practical *inside the fabric*: a translator that exists only while a foreign-language caller is on the line; a diagnostic probe attached to a live stream for thirty seconds; a burst of parallel workers for a spike. **Measured (Section 6.6): instantiating a fresh in-process module (a new crank) into the running graph and receiving its first message takes 0.30 ± 0.003 ms — roughly an order of magnitude below a ROS 2 composable component's constructor→first-message (3.2 ms in-container; note this ROS 2 figure includes up to one 200 Hz publisher period, ≈2.5 ms, of sampling quantisation, so treat the multiplier as approximate) and far below a practical `ros2 component load` (≈990 ms) or a standalone node spawn (≈200 ms constructor→first-message, discovery-dominated; ≈322 ms including process fork/exec).** ROS 2's node abstraction — with DDS participant creation, discovery traffic, and lifecycle management — is not designed for this cadence; NATS provides no compute lifecycle at all (subjects are passive strings). **Honest caveat:** Psyclone's *external-process* attach path (a separate OS process joining a running node) is currently non-functional in our Linux build and is not claimed here; the 0.30 ms figure is for an in-process module, and a full Psyclone node cold-boot (221 ms) is only roughly comparable to a ROS 2 standalone node spawn (≈322 ms incl. exec), not a large win.

6.4 Runtime subscription updates: three hands on the patch panel

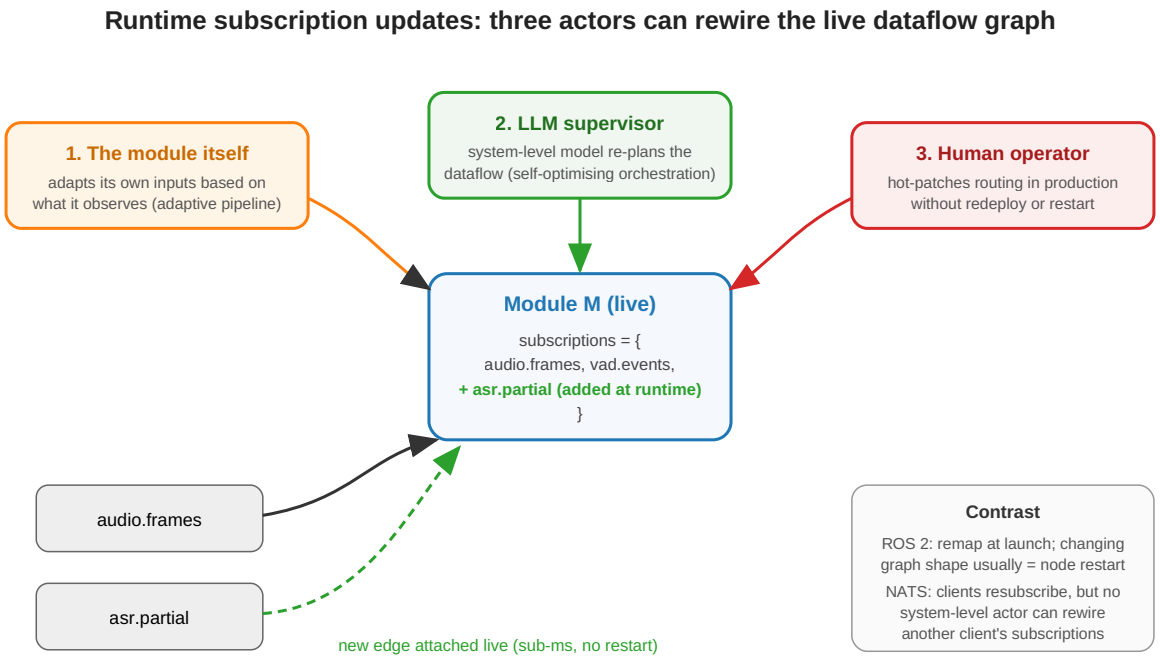


Figure 6. A live module's subscription set being extended at runtime. Three classes of actor can rewire the graph: the module itself, an LLM supervisor, and a human operator.

A Psyclone module's subscriptions are dynamic data, not compiled-in wiring — and they can be **updated at runtime by three classes of actor**:

- **The module itself.** A module that determines it needs another input — an ASR module subscribing to a second microphone stream when diarisation detects a new speaker — adds the subscription via `PsyAPI::addSubscription()` and receives data on the new stream shortly after. **Measured (Section 6.6): issue→first-message-on-the-new-subscription is 300 ± 3 µs end-to-end (the `addSubscription()` call itself is 188 ± 3 µs), with zero failures across 1000 runtime updates.** Pipelines become *adaptive* rather than provisioned-for-worst-case. **Honest semantic:** in the shipping build a runtime update *replaces* the module's trigger set and binds a fresh crank, rather than hot-appending a single trigger to a running crank — an update must restate the triggers the module still wants. It is a real, reliable runtime rewire, not an incremental append.
- **A system LLM supervisor.** Because subscriptions are runtime-mutable system objects, a supervisory model can observe the running graph and rewire it: route a confusing input to a heavier model, attach a verification stage after a low-confidence answer, prune stages from a hot path. This is the substrate mechanism on which LLM-supervised, self-optimising orchestration is built. (The supervisor *policies* we run today are ours and application-specific; an autonomous general-purpose supervisor is roadmap, not a claim of this paper — the substrate capability it would drive is shipping. In the current build the module-initiated `addSubscription()` path is the demonstrated mechanism; a dedicated third-party supervisor API beyond that path and the PsyProbe web control surface is not yet shipped.)
- **A human operator.** An engineer can hot-patch dataflow in production — tap a stream into a debugger, swap a misbehaving module's input source, insert a filter — without redeploying or restarting anything.

Demonstrated scope, stated honestly: of these three actors, only the **module-initiated** path is measured here (Section 6.6), and its shipping semantic is replace-registration with a fresh crank binding, not incremental append. The supervisor and human-operator paths are the *same substrate mechanism* driven by a different caller; a dedicated third-party supervisor API is not yet shipped beyond the module-initiated call and the

PsyProbe web control surface. We present the LLM-supervisor and operator cases as substrate capabilities, not as separately benchmarked features.

We consider this the deepest differentiator because it changes what kind of artifact a pipeline is. To be precise about the competitors — since a naive "they are static" claim would be wrong: NATS clients *can* subscribe and unsubscribe dynamically, and ROS 2 nodes *can* create and destroy subscriptions at runtime. The distinction is two-fold. First, in NATS and ROS 2 an actor manages only *its own* subscriptions; there is no first-class mechanism for a *third party* — a supervisor or an operator — to rewire another module's inputs without that module's cooperation. Second, those subscription changes are per-endpoint imperative operations, not a *global* context switch that re-routes every affected stream at once. In Psyclone the dataflow graph is a *runtime object* with an API, mutable at microsecond-to-millisecond timescales by the module, a supervisor, or a human alike. Combined with 6.1–6.3, this is what "orchestration substrate" means concretely: the fabric itself is programmable while it runs.

6.5 What we are not claiming

For clarity, and consistent with the honesty standard of Section 4: live reprogramming of a running system's code is a working, in-product capability today, performed by a human operator, and the LLM-driven path over the same substrate hook is in engineering now and will be available by publication (Section 1, capability 5). What remains genuinely roadmap — and is **not** claimed as shipping here — is a fully closed-loop **self-programming** system (Psyclone autonomously authoring and rewriting its own module specifications end-to-end, with no human or fixed supervisor policy in the loop) and **live module migration** across nodes. Nothing in this paper should be cited as evidence that the fully autonomous closed loop or cross-node migration ships today.

6.6 Differentiator measurements

The four capabilities above are not asserted from the specification; each was exercised on the running Psyclone 2.1.0 build with a purpose-built harness, 10 trials, mean ± 95% CI, same rigor as Section 4. Harness sources, specs, raw logs, and analysis are in `benchmark-data/differentiators/`.

Capability	What was timed	Result (mean ± 95% CI)	Correctness
Multi-input join (6.2)	3rd input arrives → module activates	**93.5 ± 5.7 µs** (p99 133 µs)	1000 fires / 1000 triples / 3000 msgs, 0 incomplete, every trial
Context reroute (6.1)	context switch → message delivered on new route	**112.4 ± 3.3 µs** (propagation 91.2 ± 2.6 µs)	3000/3000 correctly routed, 0 misroutes
Runtime subscription update (6.4)	issue → first message on new subscription	**300.1 ± 3.2 µs** (call 188 µs; p99 773 µs)	0 failures / 1000 updates
Ephemeral module (6.3)	instantiate in-process module → first message	**0.30 ± 0.003 ms** (ROS 2 component ctor→msg 3.2 ± 0.9 ms)	—

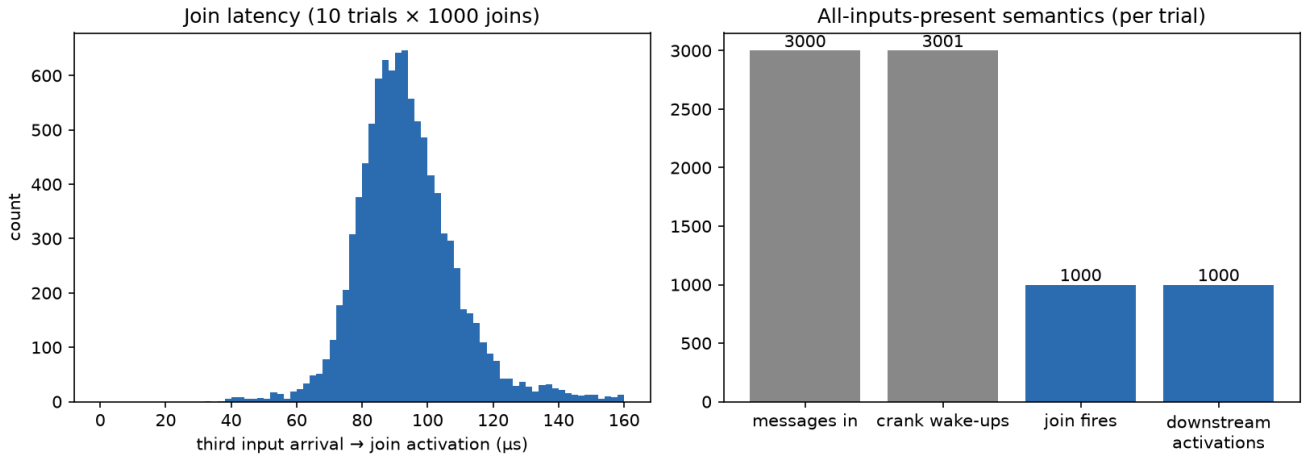


Figure 8. Left: distribution of third-input→activation latency for the all-inputs join (10×1000 joins). Right: activation-count proof — 3000 input messages produce exactly 1000 join fires and 1000 downstream activations, confirming the module acts only on complete triples.

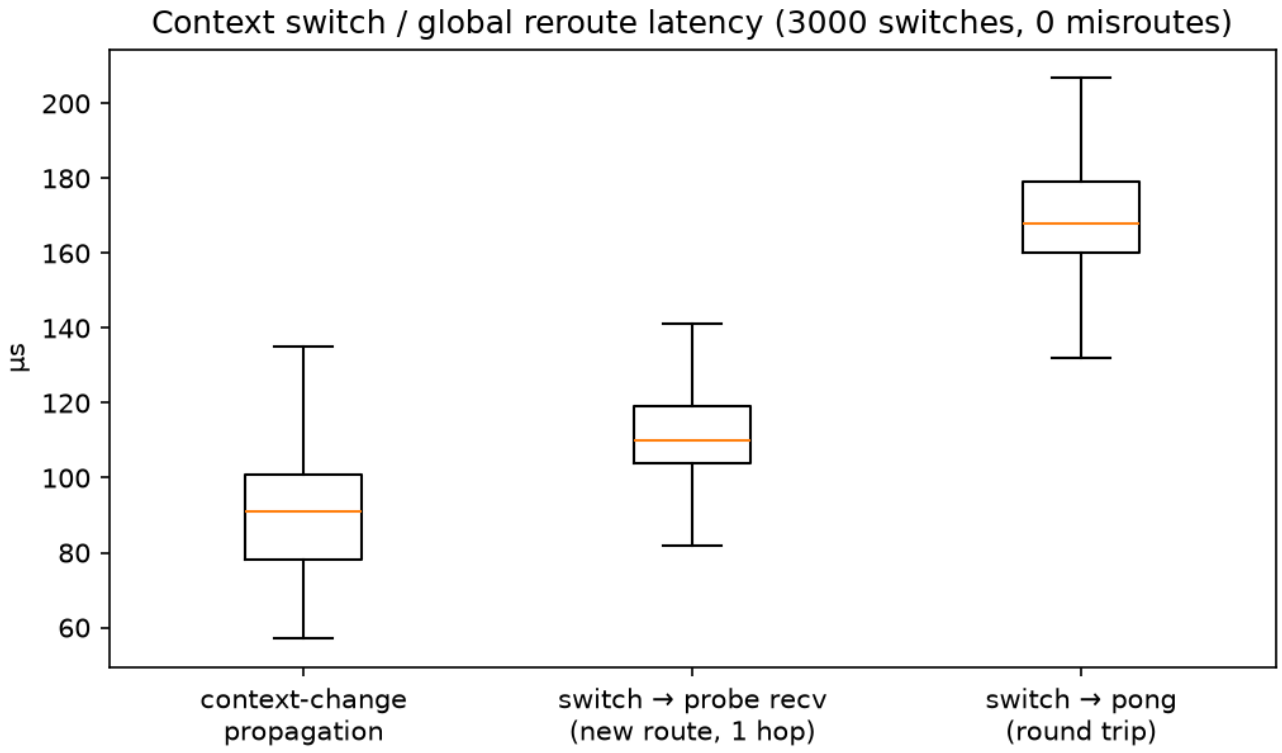


Figure 9. Global context-switch latency across three measured boundaries (propagation, switch→new-route delivery, round trip). A message posted immediately behind the switch is delivered on the new route in ≈0.11 ms with zero misroutes.

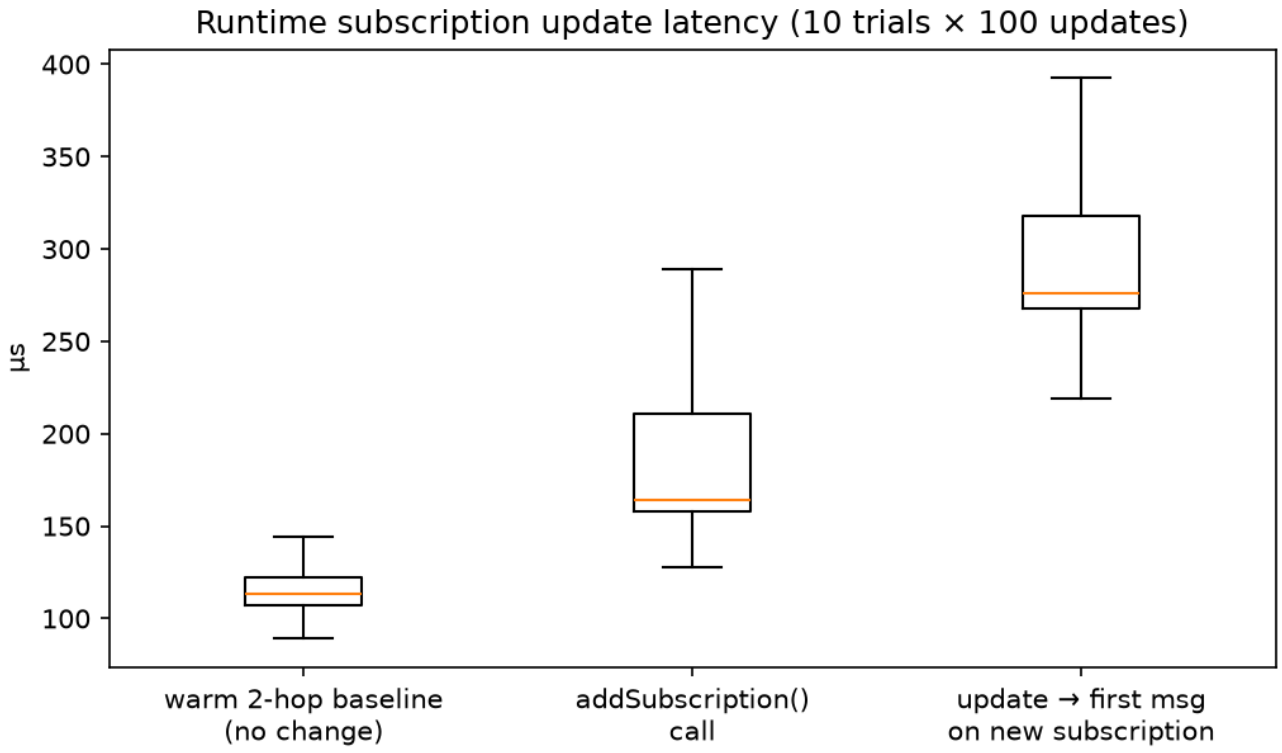


Figure 10. Runtime subscription-update latency: warm 2-hop baseline, the blocking `addSubscription()` call, and end-to-end issue→first-message on the newly added subscription.

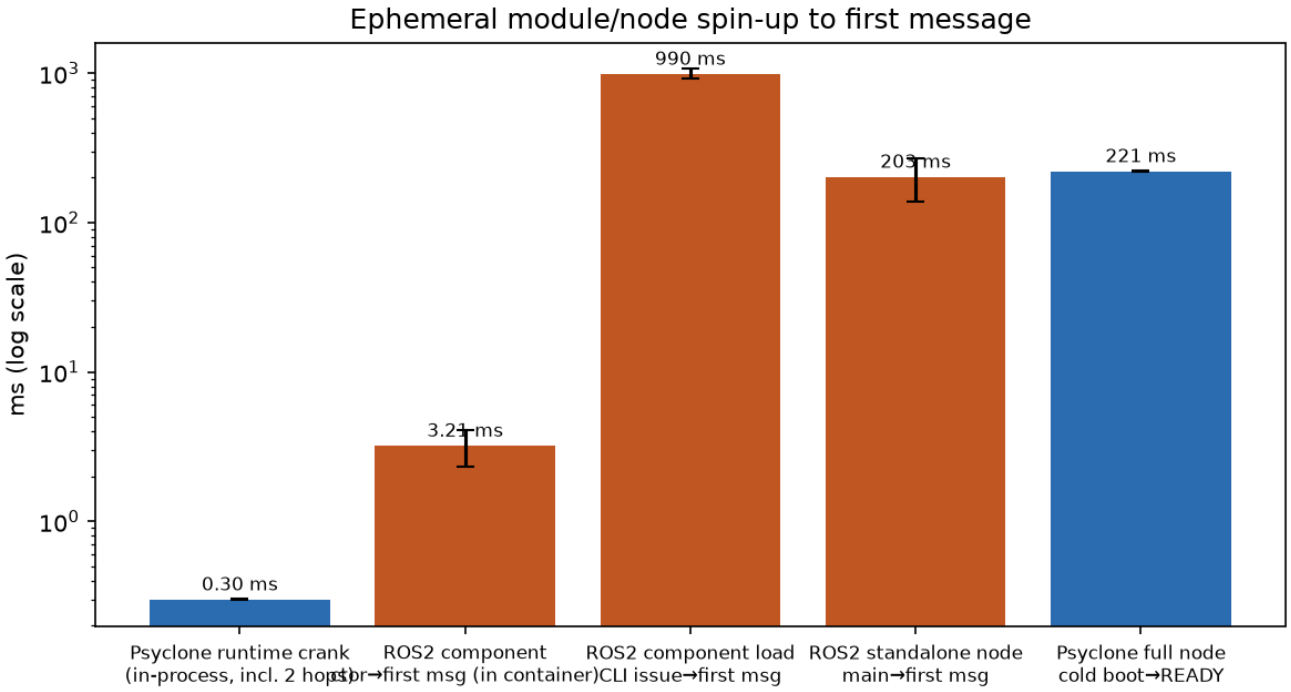


Figure 11. Log-scale spin-up comparison. Adding an in-process module to a live Psyclone graph and receiving its first message (0.30 ms) versus ROS 2 component/node instantiation paths. Psyclone's external-process attach path is currently non-functional on Linux and is not plotted (disclosed in text).

7. Threats to Validity, and Where Psyclone Does Not Win

Where Psyclone trails on raw latency, stated plainly — the numbers are in the tables and stand on their own. (i) Against tuned ROS 2: SHM/data-sharing is ≈28% faster at small payloads and ≈2.4× faster at 64 KB; intra-process composition is ≈4× faster at small payloads (with the caveats of Section 4.2 — intra-process surrenders process isolation and distribution entirely). (ii) Against the dedicated IPC specialists: iceoryx, eCAL, ZeroMQ `inproc`, and `rmw_iceoryx_cpp` all post lower medians than Psyclone's default (iceoryx ≈6 µs, roughly 6× faster at small payloads). These are bare transports — they move bytes between endpoints and provide no orchestration layer, no crash-isolated module lifecycle, no dynamic scaling, no runtime graph reconfiguration, and (for zmq `inproc`) not even process isolation; the latency gap is the cost of the substrate features Psyclone adds on top, and it is small and predictable. (iii) Large-payload copies: Psyclone's 64 KB median (85 µs) trails the zero-copy configurations — addressed on the roadmap (see below). (iv) On bare metal, Psyclone's small-payload medians measured ≈10 µs slower than virtualised (48.5 vs 38.1 µs; Section 4.6) — a systematic property of the 2-socket part under C-states-off paced load, seen across ROS 2 and the peer trio too, not a Psyclone-specific effect. (v) Ecosystem: ROS 2's tooling and community footprint is larger; NATS's cloud operational maturity likewise.

Environmental threats. Frequency governor control was unavailable on the virtualised instance, and shared-tenancy hypervisor jitter was a residual threat bounded only by the 10-trial CIs. The bare-metal `z1d.metal` replication (Section 4.6) closes this for the tail-latency claims: with governor `performance`, turbo on, and C-states off, every peer-class p99.9 tightened or held (35–58%), the two single-trial virtualised outlier cells vanished entirely, and worst-case p99.9 CIs collapsed from ±107 µs to ±1.6 µs — confirming that virtualised cross-trial tail variance was environmental. One disclosure the metal run adds: **on real hardware, idle C-states dominate paced-load latency** (2–3× inflation of small-payload medians with default idle states; Section 4.6); replications must control them. All three ROS 2 metal configurations were measured on `z1d.metal` i-0584eddbbebe775e5 (Section 4.6, Table 2b); the one residual blemish is a single spiked trial in the udp-64 KB cell (p99.9 = 4,940 µs in 1 of 10 trials, other nine ≈94–103 µs — disclosed in the table). For NATS, the `GOMAXPROCS=8` configuration (Sections 3.5, 4.7) shows metal ≈ virtualised.

Instrumentation. Psyclone's message-age facility resolves to 1 µs versus 10 ns for the C++/Go harnesses — quantisation of ±1 µs on values of 38–124 µs (≤3%), disclosed and immaterial to any conclusion drawn. Psyclone timestamps via its own framework facility (the system under test carries its own stopwatch); the ROS 2 and NATS harnesses use external `CLOCK_MONOTONIC` stamps. All are same-host, same-clock-domain.

Scope. Same-host measurements dominate; cross-node behaviour, where network physics dominates and brokers/DDS earn their keep, is now covered by first RTT/2 *estimates* (Section 4.8) for NATS and ROS 2 UDP — with the estimator difference disclosed and Psyclone itself N/A cross-node (external attach non-functional on Linux in 2.1.0). The ping-pong workload measures per-hop latency, not saturation behaviour. The peer class — iceoryx, eCAL, ZeroMQ `inproc` — **is measured** (Sections 4.1, 4.6): it matches or beats tuned Fast-DDS SHM, and sits above Psyclone. `rmw_iceoryx_cpp` (iceoryx as a ROS 2 RMW) **is also measured** (Section 4.7, with its iceoryx-v2.0.6 and patch caveats disclosed). On reproducibility of the Psyclone side: Psyclone AIOS 2.1.0 ships as a downloadable **LGPL-v3 source package** (`Psyclone_LGPL_2.1.0.260712`); everything exercised in this paper lives under `psyclone2/` in that package (measured build pin: commit c8c469e), so the Psyclone rows are as independently reproducible from published source as every baseline. The remaining open measurement gaps are exactly two: **Psyclone saturating throughput** (Section 4.5) and **a zero-copy large-payload path for Psyclone** (Section 5).

8. Conclusion and Reproducibility

Measured with equal-N, multi-trial, CI-reported, pinned-core, one-way-consistent methodology, the latency landscape is: the dedicated zero/minimal-copy IPC specialists are the raw-latency winners (iceoryx ≈6 µs cross-process; eCAL and ZeroMQ `inproc` ≈7–8 µs), tuned ROS 2 configurations lead among full middlewares (≈27 µs SHM, ≈9 µs intra-process, and ≈10.5 µs via `rmw_iceoryx_cpp`, the fastest RMW measured — Section 4.7); **Psyclone's default install is the best zero-tuning performer among the orchestration-capable systems** (beats NATS at every payload; within ≈4 µs of default ROS 2 at small payloads) **and the most predictable of them** (p99.9 ≤ 124 µs everywhere, smallest CIs, ≈1.5–1.8× p99.9/p50 ratio — confirmed on bare metal, Section 4.6). The specialists pay for their speed in scope (transport only, no orchestration layer; zmq `inproc` gives up process isolation); the tuned ROS 2 winners pay in expert configuration or in surrendering process isolation and distribution.

That is the table-stakes result. The thesis of the paper is the combination: a fabric that is latency-competitive out of the box *and* whose dataflow graph — routing, subscriptions, and the compute itself — is a runtime object, mutable in under a millisecond by modules, supervisory models, and human operators, on a substrate that also provides crash-isolated processes, dynamic scaling of live compute, and in-runtime simulation of new functionality. **Psyclone moves messages competitively fast and lets an LLM rewire the dataflow at sub-millisecond speed and reprogram the system's code while running — the capability that matters when the shape of an AI system has to change as fast as its inputs.** For AI systems whose topology must change as fast as their inputs do, that combination is the point.

On the roadmap. Two optimisations would extend the picture and are in progress: a zero-copy large-payload path (which the 64 KB column shows would close the remaining bulk-transfer gap to the zero-copy transports), and a full saturating-throughput characterisation to complement the per-hop latency reported here. Both are engineering work on the shipping substrate, not open research questions.

Reproducibility. Instance: AWS `z1d.2xlarge` i-09bc1e841ea1617ea, us-east-1c, Xeon Platinum 8151 @3.40 GHz, 8 vCPU (SMT on), kernel 7.0.0-1006-aws, Ubuntu. Software: Psyclone AIOS 2.1.0 — open source under **LGPL v3**, built from the downloadable source package `Psyclone_LGPL_2.1.0.260712` (everything exercised in this paper lives under `psyclone2/`; measured build pin: commit c8c469e, `git describe` BeforeGenAI-56-gc8c469e, clean working tree — it is a source package, not a public git repository, so no repository URL is cited), meaning all systems compared, Psyclone included, are buildable from published source; NATS server v2.14.3 with `nats.go` v1.52 harness, ROS 2 lyrical with `rmw_fastrtps_cpp` / Fast-DDS 9.4.8, C++ harnesses compiled `-O2` Release. Protocol: 10 trials × N=100,000 per condition, fresh processes per trial, `taskset` pinning as in §3.1, paced load as in §3.3. Artifacts retained alongside this report in `benchmark-data/p0-rerun/`: harness sources (C++ `rclcpp` pub/sub for all three ROS 2 configs, Go NATS one-way harness, `psy_ping.xml` PsyTest spec), Fast-DDS XML profiles, per-trial raw percentile files, `ci_summary.json` (machine-readable statistics behind every figure and table), the analysis script, and `figures/make_figs.py` (regenerates Figures 1–4 from `ci_summary.json`). The differentiator measurements of Section 6.6 (join, context reroute, runtime subscription update, ephemeral spin-up) are reproducible from `benchmark-data/differentiators/`: harness cranks (`src/DiffBench.cpp`), PySpec files (`specs/diff_*.xml`), the ROS 2 comparison package (`src/ephbench/`), per-trial raw logs (`raw/`), `analyze.py`, and the four data figures. Source-level API references (file:line into the Psyclone tree) for every capability exercised are recorded in `DIFFERENTIATOR-RESULTS.md`, including honest notes where a documented API differs from its implementation (e.g. `<triggergroup>` parsed-but-stubbed; `addSubscription` replace-registration semantics and its success-returns-false bug; broken external-process attach on Linux).

Peer-class and bare-metal artifacts. The peer-class measurements (Section 4.1: ZeroMQ 4.3.5 `inproc`, eCAL 6.1.1, iceoryx v2.95.5; run 2026-07-21 on the same `z1d.2xlarge` instance, kernel, and methodology) are archived in `benchmark-data/peerclass-20260721T215915Z/`: harness sources (`src/*.cpp`, `g++ -O2`, GCC 15.2.0), run scripts, raw per-trial percentile files (`raw/*.txt`), the polluted first ZeroMQ sweep kept for transparency (`raw_zmq_firstrun/`), `ci_summary.json`, `summary.txt`, and run logs. The bare-metal replication (Section 4.6) ran on AWS `z1d.metal` **i-00fb9f875a2e1c26b**, us-east-1c (48 vCPU = 2 × 12-core Xeon Platinum 8151 @ 3.4 GHz, 4.0 GHz boost, SMT on, 2 NUMA nodes), Ubuntu 26.04, kernel 7.0.0-1006-aws (identical kernel build to the paper box), GCC 15.2, governor `performance`, turbo on, C-states off (`/dev/cpu_dma_latency=0`); artifacts in `benchmark-data/metal-20260722T145818Z/`: harness sources, run/provision/build scripts, raw per-trial files (including the invalid NATS runs, marked as such), `ci_summary.json`, `summary.txt`, `campaign.log`, and `env.txt` recording the governor/turbo state. An earlier AL2023-based reference run is retained as `metal-20260722T142757Z/`. The ROS 2 metal leg (Section 4.6, Table 2b) ran 2026-07-22 21:59–22:51 UTC on a second `z1d.metal`, **i-0584eddbbebe775e5** (same Ubuntu 26.04 / kernel 7.0.0-1006-aws, governor `performance`, turbo on, C-states off); artifacts in `benchmark-data/ros2-metal-20260722T225527Z/`: raw per-trial files (`raw/`), `ci_summary.json`, `summary.txt`, harness sources (`latbench-src/`), the Fast-DDS SHM/data-sharing profile (`fastdds_shm.xml`), `env.txt`, and `DONE.flag`.

v2.2 artifacts (fixed NATS, rmw_iceoryx, cross-node). The GOMAXPROCS-fixed NATS re-run (Section 4.7) is archived in `benchmark-data/nats-fixed-20260723T063043Z/` (`virt/` + `metal/`, `natslat_main.go`, `NOTE.md`). The rmw_iceoryx_cpp measurements (Section 4.7) are in `benchmark-data/rmw-iceoryx-20260723T063043Z/` (`virt/` + `metal/`, the patched source tree `src/rmw_iceoryx-patched/`, `NOTE.md`, and build logs). The cross-node measurements (Section 4.8) are in `benchmark-data/crossnode-20260723T063043Z/` (`virt/` + `metal/`, `harness/`, `METHOD.md`, `instances.txt`, `aggregate.py`). Instances used for these runs (all TERMINATED, verified by direct AWS query): metal i-0b7c4d295acbf0bed and i-074b8c6ab47614db7; reflector i-0c7e75fb9d1ac6892; the paper box i-09bc1e841ea1617ea (stopped, pre-existing).