arXiv:2601.21011v1 [cs.RO] 28 Jan 2026

Meta-ROS：面向自适应可扩展机器人系统的下一代中间件架构

ANSHUL RANJAN, PES University，印度ANOOSH DAMODAR, NEHA CHOUGULE, and DHRUVA S NAYAK,PES University，印度ANANTHARAMAN P.N, PES University, 印度SHYLAJA S S, PES University，印度

机器人学领域面临着诸多挑战，包括现有中间件框架（如 ROS2）的复杂性和互操作性问题，这些问题可能令新开发者难以上手。为了解决这些问题，我们提出了 Meta-ROS，这是一种新型中间件解决方案，旨在通过简化集成、提升性能并确保跨平台兼容性，来简化机器人开发。Meta-ROS 利用 Zenoh 和 ZeroMQ 等现代通信协议，支持不同硬件平台间高效、低延迟的通信，同时支持音频、图像和视频等多种数据类型。我们通过全面测试评估了 Meta-ROS 的性能，并将其与 ROS1 和 ROS2 等现有中间件框架进行了比较。结果表明，Meta-ROS 的性能优于 ROS2，吞吐量最高提升 30%，消息延迟显著降低，资源使用也得到优化。此外，其强大的硬件支持和以开发者为中心的设计，有助于实现无缝集成并提升易用性，使 Meta-ROS 成为现代实时机器人人工智能应用的理想解决方案。

CCS 概念：• **计算机系统组织** → **实时操作系统**；**机器人控制**；专用系统。

补充关键词和短语：机器人学、机器人操作系统、云机器人、中间件、多智能体系统、实时系统、通信协议、Zenoh、机器人传感器

ACM 引用格式：
Anshul Ranjan、Anoosh Damodar、Neha Chougule、Dhruva S Nayak、Anantharaman P.N 和 Shylaja S S。2026年。Meta-ROS：自适应可扩展机器人系统的下一代中间件架构。*ACM*自主与自适应系统汇刊37，4，第111号文章（2026年8月），15页。https://doi.org/XXXXXXX.XXXXXXX

1 引言

机器人学和物联网（IoT）的飞速发展，为可靠、高效的机器人系统开发带来了重大挑战 [29]。机器人操作系统（ROS） [27] 通过提供灵活且模块化的软件框架，在塑造机器人行业方面发挥了关键作用，使研究人员和开发者能够构建复杂的机器人应用。然而，随着机器人系统日益复杂，ROS 2 的局限性也愈发明显。ROS 1 最初由 Willow Garage 开发 [27]。由于其采用开源模式，它很快成为机器人开发的基础，

作者联系方式：Anshul Ranjan，anshulpranjan@gmail.com，PES University，印度卡纳塔克邦班加罗尔；Anoosh Damodar，anooshsd12379@gmail.com；Neha Chougule，neharajchougule@gmail.com；Dhruva S Nayak，dhruvanayak1905@gmail.com，PES University，印度卡纳塔克邦班加罗尔；Anantharaman P.N，ananth@jnresearch.com，PES University，印度卡纳塔克邦班加罗尔；Shylaja S S，shylaja. sharath@pes.edu，PES University，印度卡纳塔克邦班加罗尔。

允许为个人或课堂使用而免费制作本作品的全部或部分数字副本或纸质副本，前提是不得为营利或商业利益制作或分发副本，并且副本首页须附有本声明及完整引文。对于本作品中由作者以外人士拥有版权的内容，必须尊重其版权。注明出处后可以摘录。若要以其他方式复制、再版、发布到服务器或分发至列表，则须事先获得特别许可和/或支付费用。请向 per missions@acm.org 申请许可。© 2026年 版权归所有者/作者所有。出版权已授权给 ACM。ACM 1556-4703/2026/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

这使得实用工具、算法和社区贡献得以大规模应用。事实证明，它是一个极佳的选择；然而，由于在安全性 [2], 网络拓扑和系统正常运行时间方面存在严重缺陷，它仍无法用于任何商业用途。因此，ROS 2 应运而生，旨在解决这些问题，并为 ROS [18]引入实时能力 [26], 、增强的安全性 [35], 以及对多机器人通信方案更完善的支持。

ROS 2 使用数据分发服务（Data Distribution Service），这是一项用于通信的已发布开放标准，广泛应用于军事、航空航天和金融等关键基础设施领域 [25]。DDS 提供分布式发现、内置安全性，并支持 UDP 等多种传输层。因此，它可被视为实时应用的理想选择 [32]。然而，尽管进行了这些改进，ROS 2 仍给开发者带来了相当大的挑战 [23]。框架的复杂性、陡峭的学习曲线，以及在不同平台之间实现无缝硬件集成的困难，都给开发可扩展且高效的机器人应用带来了阻碍。开发者面临的挑战之一，是为机器人和物联网应用提供统一的操作系统，而该系统最初并非为这些用途而设计。每个机器人系统中的硬件各不相同，因此软件开发变得更加困难，并且代码复用的潜力较低。包括 ROS 在内的拟议解决方案 2 [18], 由于硬件的多样性以及通信协议的异构性，难以提供用于顺畅交互的互操作性框架。计算能力需求也存在差异。

本文旨在提出开发一种先进的开源机器人中间件，该中间件基于 ROS 2 技术构建，同时解决其现有局限 [15, 20]。所提出的中间件旨在通过引入新颖功能，简化不同机器人平台之间的互操作性和通信难题，从而全面满足现代机器人应用日益增长的需求 [5]。这一设想中的中间件将充当各种机器人系统之间的桥梁，使机器人软件能够更轻松地适配更广泛的物理设备 [1]。它将为开发者提供简化的设计方案和一套丰富的工具，以促进开发流程。滤波器将作为有限资源平台之间的接口，使开发者能够更高效地创建复杂的机器人应用，并提升性能 [31]。

机器人行业瞬息万变、发展迅速，因此需要一种适用范围广、稳健且以开发者为中心的元操作系统。它应通过统一多种硬件平台并简化机器人应用的开发，解决当前存在的不足。复杂的软件架构以及与广泛硬件配置兼容的需求带来了重大挑战，造成技术障碍和不尽如人意的结果。机器人软件开发面临的最大挑战之一，是不同机器人系统的硬件集成程度各不相同。硬件不匹配和通信方式各异，阻碍了有效的数据交换与命令执行，导致机器人操作效率低下。此外，要与各种机器人框架实现全面互操作性并非易事，因为现有解决方案无法确保跨平台的无缝通信。

为解决这些问题，我们提出的面向机器人学的元操作系统将强调灵活性与易用性，为使用各类硬件平台的开发者提供统一框架。它将力求标准化开发流程，避免兼容性挑战，并促进机器人学新应用的创建。本项目的主要目标如下：

(1) 这是一种轻量级中间件，旨在不依赖特定的操作系统和处理器，因此比 ROS 2 更适用于物联网。
(2) 基于发布-订阅模型开发通信系统。这有助于确保设备之间的交互顺畅进行，避免兼容性问题。

表 1. Meta-ROS 与 ROS1 和 ROS 2 的特性对比

类别	ROS 1	ROS 2	Meta-ROS
网络传输	定制协议 (TCP/UDP)	标准 DDS，抽象化——为其他对象提供.....	基于 Zenoh 的混合实时传输。
网络架构 ture	集中式（roscore）	点对点发现	云集成、混合式 P2P/集中式。
平台支持	Linux	Linux	Linux、Windows、macOS
客户端库	因语言而异	共享 C 库 (rcl)	统一的 Python API，支持多语言。
节点与进程	每个进程一个节点	每个进程多个节-cess	动态节点管-理跨设备。
线程模型	回调队列	可替换执行器	可替换执行器
节点状态管理	None	生命周期节点	自适应状态转-换。
嵌入式系统	实验性（rosserial）	支持（micro-ROS）	优化的 RTOS 集成-功能。
参数访问	基于 XML-RPC 的协议	服务调用	服务调用
参数类型	推断的类型	已声明	强类型 类型化，验证过。

(3) 支持环境中的各种外围设备，包括摄像头、扬声器、麦克风和激光雷达传感器 [37]。

我们的解决方案旨在解决当前机器人框架的关键局限，帮助开发者和研究人员充分发挥新兴机器人技术的潜力，从而推动该领域的创新与进步。

2 相关工作

过去五十年间，机器人软件经历了巨大变革，最初的研究涉及基础自动化工作 [24]。斯坦福购物车等机器人以及能够以自动化模式穿越复杂环境的机器人，代表了这一变革中的创新。早期研究主要致力于寻找确定性算法，以实现分层结构控制系统的导航和反应行为；这些系统构成了现代机器人学架构的基石。

最早的机器人集成软件包之一是移动机器人编程工具包（MRPT） [17]。该软件将传感器、建图算法和控制逻辑集成到自主系统中。MRPT 为机器人实现同步定位与建图（SLAM）作出了重要贡献。虽然 CORBA 和 DDS 等中间件技术颇受欢迎 [25], ，但它们能够在分布式系统中实时共享数据，并对底层通信协议进行抽象。现代机器人框架，例如专注于模块化和可移植性的 Robot NavigationFramework。RNF 是最早采用基于插件架构设计的框架之一，因此可以轻松地在路径规划和避障之间切换。除了有助于提高代码复用率、促进研究团队之间的协作开发外，这类框架也有不足之处：搭建需要丰富的专业知识，导致较小的开发团队或个人无法迅速采用这些应用。

消息传递系统在机器人学的发展过程中发挥了重要作用。早期设计基于简单的点对点通信，后来演变为发布/订阅模型，ZeroMQ 和 Kafka 就采用了这种模式 [13]。这类消息传递系统具有较高的吞吐量，并能确保稳健的容错能力，因此适用于许多需要地理位置分散的单元之间高效通信的机器人应用。例如，研究表明，ZeroMQ 在多智能体机器人学中确实颇具实用价值，它展示了如何在无人机之间高效交换信息 [6]。近期的发展趋势则倾向于轻量、对开发者友好的解决方案。例如，Fawkes 是专为小型机器人平台开发的轻量级中间件。Fawkes 支持实时控制，并提供易于理解的图形界面，用于系统诊断，因此常用于教育机器人学。尽管它的易用性相当出色，但并不具备工业级能力，例如高带宽数据处理或与云平台集成。

与这些较早的框架不同，ROS 2 等现代生态系统重新定义了机器人软件的发展格局[18]。ROS 2 针对 ROS 1 的局限进行了改进，将 DDS 用作其核心通信机制，从而无需中心节点即可实现去中心化通信。这种设计增强了容错能力 [12], , 尤其是在存在间歇性连接的环境中。此外，ROS 2 集成了实时功能 [4, 38] 和内置安全措施 [3], , 解决了医疗保健、自动驾驶车辆[11], 和工业自动化等应用中的关键问题。

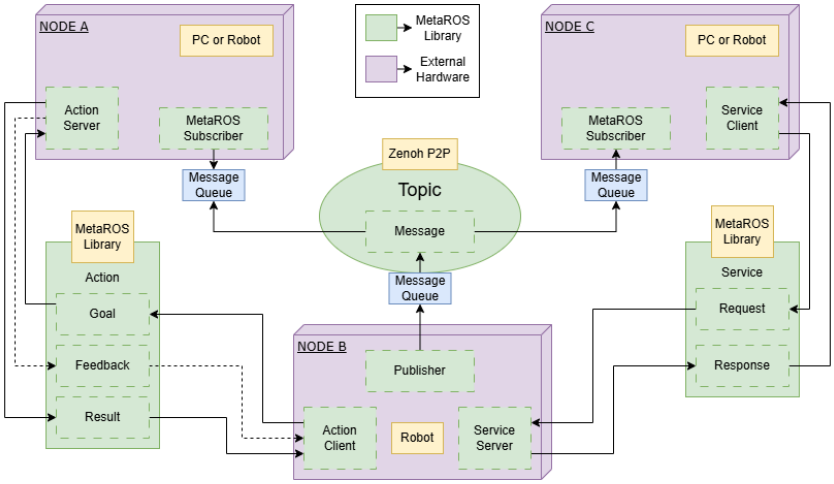

图 1. Meta-ROS 接口及其交互，涵盖主题、服务和动作。

在我们的项目中，我们通过集成云原生技术 [8, 14]，采用了面向新一代机器人系统的方法。借助 AWS Lambda 实现无服务器计算 [39]„ 我们采用事件驱动架构，以提升可扩展性和效率。此外，我们利用 gRPC 协议和 WebSocket 协议，确保机器人智能体与控制中心之间的低延迟通信。微服务编排管理可确保高可用性和无缝部署。与传统机器人框架不同，我们的系统优先考虑互操作性。我们支持多种机器人平台，从空中无人机到水下航行器 [34], , 并通过单一接口实现统一管理 [21]。借助基于 AI 的监控工具，我们的系统能够实时检测异常，并主动防止故障。这种现代化方法让机器人技术开发更加普及，使初创企业和研究人员都能轻松参与 [10, 30]„ 同时应对可扩展性、容错和安全性方面的挑战。云机器人 [9]

以及分布式系统方法 [7] 进一步增强了现代机器人平台的能力，使复杂的多机器人系统 [22]和协作式人机交互 [16, 28]成为可能。

3 Meta ROS

3.1 范围

Meta ROS 支持任何操作系统，且极易安装和使用。该库包含众多用于实现各种功能的函数和类，可供导入和使用。开发 Meta ROS 时，我们的目标是设计一种轻量级中间件，使其能够运行于任何操作系统或处理器，并且比 ROS2 更适用于物联网设备。创建一个采用发布者-订阅者模型的通信系统，将使不同设备能够通过操作系统轻松交互而不会遇到问题。我们还希望降低 ROS2 的安装复杂度，并通过将其作为 Python 库发布到 PyPI 实现了这一目标。另一个重点是开发设备驱动程序，以确保全面支持摄像头、扬声器、麦克风和激光雷达传感器等多种外部外围设备。

3.2 设计

我们认为，通过减少辅助材料并降低依赖项管理的复杂性，实际上我们正在开发一种轻量且易于使用的中间件解决方案，它能够复现并改进 ROS 的功能和代码模式，包括主题、订阅者、消息和回调 [19, 33]。我们的目标是优先采用清晰、基础的解决方案，从而提高开发者的工作效率。

3.2.1 设计原则。 MetaROS 的设计遵循一系列原则和具体要求。以下原则是其核心：

简洁性： 开发者可以轻松采用 ROS 的编码模式，无需额外费心管理 Linux、bash、Docker 和虚拟机等依赖项。

服务质量： 我们的中间件优先考虑服务质量的重要方面，例如速度、安全性、隐私和可用性。

分布式： Meta ROS 将需求拆分为功能上相互独立的组件，例如设备驱动程序、感知系统和控制系统。这些组件拥有各自的执行环境，并通过显式通道进行通信，从而实现去中心化且安全的组合。

异步性： 组件之间的通信以异步方式进行，从而形成基于事件的系统。**模块化：** 遵循 UNIX 的设计理念——让每个程序专注做好一件事——我们在多个层面实现模块化，包括库 API、消息定义和命令行工具。

实时计算： 考虑到机器人应用普遍存在实时计算需求， [36], 该中间件提供 API，用于施加特定于应用的限制；这对于实现安全性和性能目标至关重要。

3.2.2 设计细节。 Meta ROS 旨在根据设计原则和机器人开发者的需求，满足以下要求：

新颖性： Meta ROS 为中间件架构提供了一种新颖的方法，剖析通信模式，并利用 Zenoh[13] 基于发布者-订阅者范式构建独特的通信系统。此外，我们计划增强对多种编程语言的支持，并确保与各种外部设备兼容，从而为当前的机器人操作系统提供更优的替代方案。

性能：Meta ROS 通过优化数据序列化和通信协议来降低延迟并提升整体系统性能 [33]。为确保资源受限设备上的操作系统保持灵敏、轻量，我们采用了高效的资源管理策略，例如缓存算法。

对于发布-订阅、客户端-服务器和动作客户端-动作服务器架构中的非关键节点，我们使用完全公平调度器（CFS）[4]。CFS 使用红黑树，通过调整时间片（1）和虚拟运行时间（2）来管理执行，从而公平地分配 CPU。

$$T_i = T_{\text{sched}} \times \frac{w_i}{W_{\text{total}}} \tag{1}$$

$$V_i = V_i + \frac{w_{\text{base}}}{w_i} \times R_{\text{real}} \tag{2}$$

安全性：为确保数据完整性并防止节点间通信，Meta ROS 包含多种安全措施，例如输入和数据清理、访问控制以及 AES 加密 [2, 35]。安全升级选项可确保操作系统始终更新至最新的安全标准，并防止未经授权的访问。

可靠性：Meta ROS 使用容错策略 [12]，例如消息队列和自动节点恢复，以确保系统即使在发生短暂故障时也能持续运行。

资源利用：我们的目标是最大限度地利用可用资源，例如 CPU 和内存，确保操作系统在各种硬件配置下都能高效运行。我们采用缓存方法等高效的内存管理策略来缩短运行时间。

3.3 通信模式

Meta ROS 的通信模型基于发布者-订阅者范式，与 ROS2 模型相似[18], ，但在此基础上进行了重大改进，并引入了全新的框架。

3.3.1 主题。元操作系统（Meta OS）采用以主题为核心的异步消息传递框架。主题提供发布者-订阅者机制，用于通信。不过，Meta OS 将这一概念扩展为强类型接口架构。在 Meta OS 中，端点在节点这一抽象下构成计算图。

节点是关键的组织单元，使用户能够有效地管理复杂系统并对其进行推理。我们的设计支持多对多通信，为系统自省带来显著优势。开发者无需修改系统，只需创建订阅即可监控任意主题上的消息，从而增强可观测性和调试能力。

3.3.2 服务。虽然通过主题进行异步通信用途广泛，但在某些场景下，请求-响应模式更为合适。Meta OS 通过服务支持这种模式；服务是一种通信模式，旨在将请求与响应直接进行数据关联。这对于验证任务是否完成或是否收到确认尤其有用。

与传统的阻塞机制不同，Meta OS 确保服务客户端在调用期间不会阻塞。与主题一样，服务也与节点相关联，从而整合子系统的接口，便于高效组织和诊断。

3.3.3 动作。Meta OS 中一种独特的通信范式是动作。动作提供以目标为导向的异步通信，并支持周期性反馈、取消以及请求与响应配对。这种设计非常适合长时间运行的任务，例如自主操作或复杂计算，同时也足够灵活，可用于各种应用。

与服务类似，动作是非阻塞的，并归属于节点。这种结构使用户能够高效地查看和管理子系统接口。动作模式使开发者能够实现稳健且响应迅速的系统，从而无缝处理长时流程。

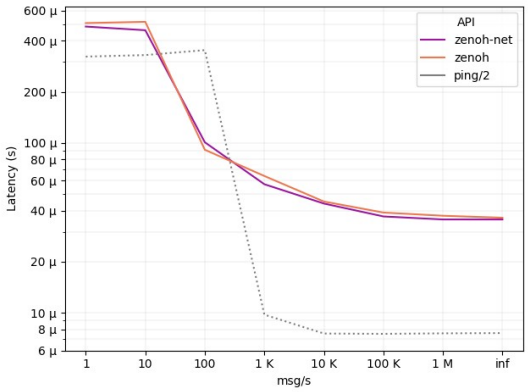

图 2. Meta-ROS 使用 Zenoh 时的性能。

3.3.4 Zenoh。 Meta ROS 利用 Zenoh 框架实现分布式数据处理 [13], , 在网络中提供高效可靠的数据分发机制。Zenoh 以数据为中心的方法通过其基于Rust的实现，实现了高效的数据分发与发现。根据我们的实践分析，以及对比较 Zenoh 与其他通信框架的研究论文的研读，我们得出结论：采用 Zenoh 能够提供最佳性能和最低延迟，且缺点极少。

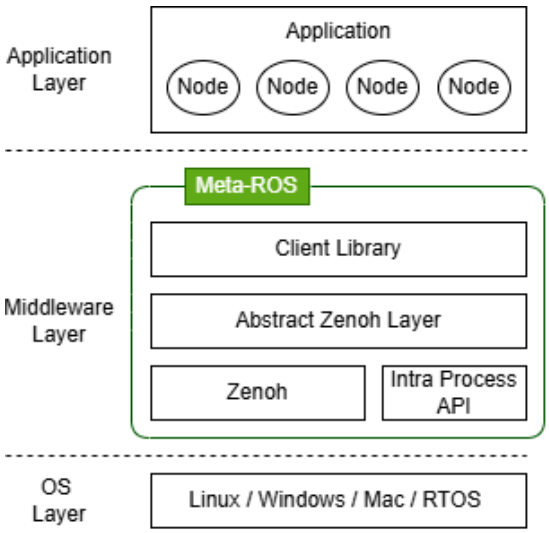

图 3. 系统架构高层概览。

3.4 中间件架构

Meta ROS 中间件以 Python 模块化库的形式构建，遵循面向对象编程原则，以确保可复用性和可维护性。其通信模型基于消息传递

架构，支持发布-订阅和请求-应答两种模式。该模型旨在促进独立组件之间的数据交换，同时最大限度地减少依赖，从而确保灵活性并减少瓶颈。

消息代理。 消息代理充当中央通信枢纽，管理发布者和订阅者之间的交互。该代理基于 Zero MQ 构建，并提供主题，供不同组件发布消息或订阅消息，从而实现事件驱动的通信模型，使各个进程彼此解耦。

发布者类和订阅者类。 发布者类允许不同组件向指定主题发布数据，并支持多种消息类型，例如字符串、整数和浮点数。另一方面，订阅者类会接收其订阅主题中的消息。这些类共同构成高度灵活的通信结构，使组件能够根据需要动态加入或退出数据流。

用于通信控制的支持类。

- **定时器：** 管理消息分发和调度的时间。
- **主题：** 定义用于不同类型数据交换的通信通道。
- **速率：** 控制消息发布到主题的速率，确保时间安排可预测。
- **数据记录器：** 记录组件之间交换的消息，以便进行调试和性能分析。

这些辅助类各自承担特定职责，用于管理消息流、控制通信速率，并支持诊断，从而有效排查问题。

3.5 软件质量

我们采用了全面的测试策略，以确保中间件的可靠性、性能和鲁棒性。该策略结合了功能测试和非功能测试，以保证中间件在各种更新、环境和使用场景下的稳定性。鉴于机器人环境具有不可预测性，我们通过广泛评估，致力于维持较高的容错能力和可靠性。

- **单元测试：** 重点测试发布者、订阅者和消息代理类等指定组件，以验证各个模块的行为和鲁棒性。
- **集成测试：** 确保组件之间实现无缝的数据序列化/反序列化和互操作性。
- **端到端测试：** 确认在各种配置下，从发送节点到接收节点的消息传输均正确无误。
- **性能测试：** 评估正常和压力条件下的吞吐量、延迟和资源使用情况 [15, 20]。
- **可靠性测试：** 在高消息负载下进行稳定性测试，包括针对丢失消息的重试。测试场景包括网络中断和节点故障。
- **兼容性测试：** 对 Meta ROS 在 Linux、Windows 和 macOS 等主流操作系统上进行了广泛测试，以确保无论开发者使用何种操作系统，都能保持稳定的性能。

3.6 可靠性与容错措施

机器人环境本质上难以预测，因此容错是我们的中间件 [12] 的关键要求。Meta ROS 实现了以下可靠性机制：

- **重试机制：** 在发生故障时，通过重试消息传递来确保通信可靠。
- **故障场景测试：** 测试模拟的网络中断和意外节点关闭等场景。
- **错误处理与故障转移：** 在测试过程中，我们评估了错误处理机制，以尽量缩短停机时间。我们还测试了多种故障转移流程，以确保发生故障时仍能持续运行。

这一测试策略确保中间件平台稳健可靠，能够在各种复杂的机器人学场景中提供可靠的性能 [18]。

3.7 性能

Meta-ROS 通过广泛测试实现了高度稳定性，满足所有功能性和非功能性要求 [38]。这些测试包括严格的单元测试、集成测试和端到端测试，确保各模块均能稳健运行。Meta-ROS 在各种测试环境中均保持稳定，验证了其在现实世界机器人学应用中的可靠性 [16]。

$$L = T_{\text{receive}} - T_{\text{send}} \tag{3}$$

$$T_{\text{msg}} = \frac{N_{\text{msg}}}{T} \tag{4}$$

Meta-ROS 的性能通过多项关键指标进行评估，包括端到端延迟（L）（3）、吞吐量（T_{msg}）（4）、CPU 利用率（C）（5）和带宽利用率（T_{bit}）（6） [15, 20]。这些指标有助于深入了解系统的效率、响应能力和资源使用情况。评估结果见图 4，展示了 Meta-ROS 在现实世界应用中的性能特征和稳定性。

$$C = \frac{1}{T}\sum_{t=1}^{T} C_t \tag{5}$$

$$T_{\text{bit}} = \frac{B}{T} \tag{6}$$

3.7.1 性能亮点。

- **可靠性：**凭借先进的错误处理机制，即使发生网络故障或节点崩溃，Meta-ROS 仍能维持正常运行 [12]。
- **鲁棒性：**该中间件能够有效处理高消息负载，在压力场景下仍保持平稳运行，性能未见下降 [5]。

与 ROS 等框架相比，Meta-ROS 展现出明显优势 2 [18],，尤其是在性能、易用性和对多样化应用场景的支持方面。

3.7.2 性能指标。

- Meta-ROS 的吞吐量最高比 ROS 高出 30% 2 [18]。
- 高效的序列化和反序列化机制显著降低了消息延迟 [33]。
- Meta-ROS 展现出稳定且高效的带宽利用率 [13]。

4 测试

我们的中间件测试策略旨在确保其可靠性、性能和鲁棒性 [23]。该策略结合了功能性与非功能性测试方法，以确保系统在各种更新、环境和用例下保持稳定。考虑到机器人环境的动态特性，我们在整个测试过程中着重关注容错和可靠性，旨在验证中间件在不可预测条件下的性能 [29]。

功能测试着重确保系统的核心组件（包括发布者、订阅者和消息代理）能够独立运行并在集成环境中正常工作 [27]。单元测试验证了各个模块的行为，而集成测试则确保组件之间能够顺畅地进行数据序列化并实现互操作性 [33]。通过端到端测试，确认消息能够正确传输

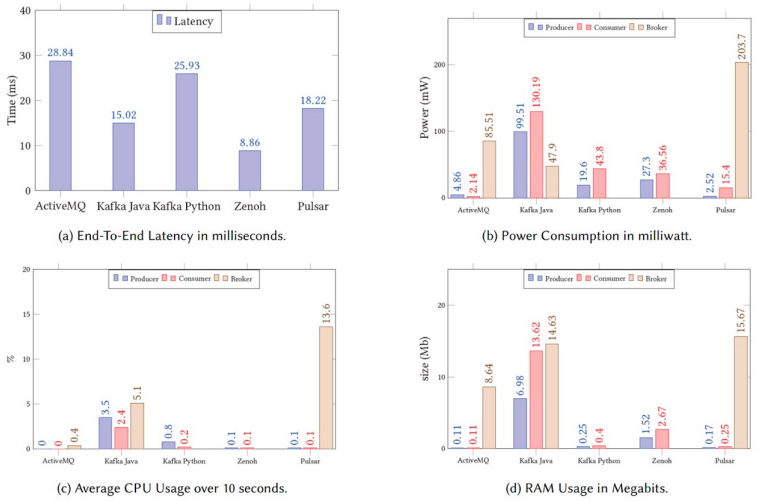

图4. 多项指标的性能比较。

从发送节点传输到接收节点，并适用于各种配置。此项全面的测试方法确保中间件能够在不同条件下可靠地处理消息交换 [19]。

非功能测试主要评估中间件的性能、可靠性和兼容性[38]。性能测试衡量了正常条件和压力条件下的吞吐量、延迟及资源使用情况 [15, 20]。可靠性测试评估了高消息负载下系统的稳定性，包括其从丢失消息、网络中断和节点故障等故障中恢复的能力 [12]。兼容性测试确认了该中间件在不同操作系统（包括 Linux、Windows 和 macOS）上的功能，确保跨平台支持。

单元测试和集成测试侧重于确保核心功能并解决代码中的错误，而 GitHub Actions 则用于自动化回归测试，并持续反馈系统稳定性。仿真器，尤其是 Gazebo，被用于创建虚拟测试环境，以便在进行真实硬件测试之前模拟各种使用场景 [22]。最终，在 PES物联网实验室的机器人上进行的真实环境测试，为中间件在实际场景中的性能和可靠性提供了最终验证 [16, 28]。

测试在多个环境中进行，包括 PC 对 PC 通信，用于模拟网络延迟场景，以验证设备间通信 [5]。Gazebo 中的仿真支持开展受控实验，以评估系统性能；而使用实体机器人进行的现实环境测试，则最直接地体现了中间件的鲁棒性和适应性 [22, 24]。测试流程旨在实现可扩展性，适应各种机器人应用，并能够通过自动化测试支持持续集成 [32]。

尽管测试流程较为全面，但仍存在一定局限性。现实环境测试依赖硬件，偶尔会造成延迟，尤其是在硬件不可用时。此外，尽管仿真器提供了受控的测试环境，但无法完全复现现实世界的动态因素，例如环境噪声和硬件差异性；这些因素可能会在不可预测的场景中影响系统性能。

表2展示了在各节点上测得的不同框架的线程数，包括 ROS1 [27],ROS2（Connext、OpenSplice 和 FastRTPS） [18],以及 Meta-ROS。

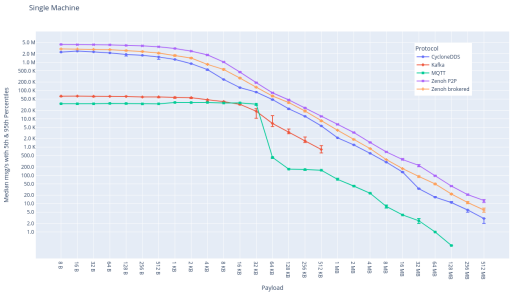

图5. 单机场景下的吞吐量（msg/s）。

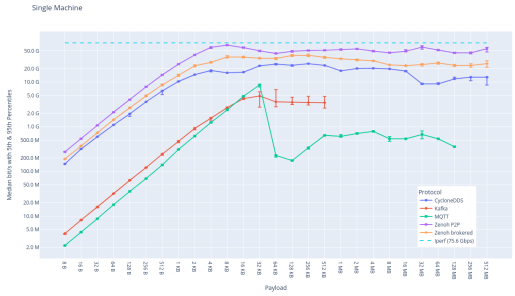

图6. 单机场景下的吞吐量（bit/s）。

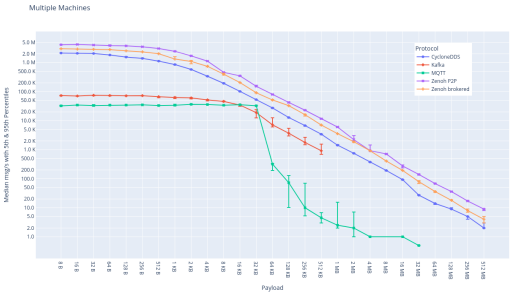

图7. 多机场景下的吞吐量（msg/s）。

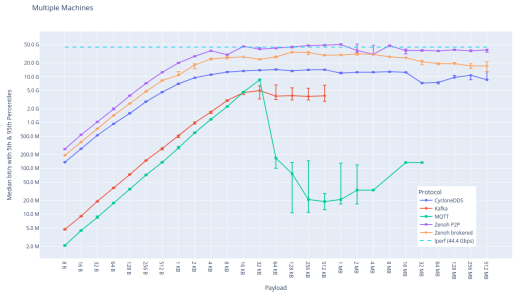

图8. 多机场景下的吞吐量（bit/s）。

表2. ROS1、ROS2 和 Meta-ROS 上的线程数量

框架	Node	主节点
ROS1	5	3
Connext（ROS2）	8	-
OpenSplice（ROS2）	49	-
FastRTPS（ROS2）	5	-
Meta-ROS	12	-

5 结果

Meta-ROS 的实现与测试展现了其在性能、可靠性和多功能性方面的显著进步，使其成为面向现代机器人应用的卓越中间件解决方案 [18, 38]。通过全面的单元测试、集成测试和端到端测试，Meta-ROS 展现出卓越的稳定性，满足了功能性和非功能性要求。它在模拟环境和现实环境中的性能验证了其在各种条件下的鲁棒性，确保其能够可靠地执行关键机器人任务[16, 28]。

在性能评估中，Meta-ROS 超出预期，其吞吐量最高比 ROS 2 高出 30%（图 8） [18, 20]。系统经过优化的序列化和反序列化流程显著降低了消息延迟 [33], ，同时即使在多媒体流等数据负载较大的情况下，也能保持较高的数据传输速率 [37]。此外，即使在带宽受限场景中，Meta-ROS 也展现出高效且稳定的带宽利用率[13], ，从而确保在高速和低延迟网络环境中均能实现最佳性能。

Meta-ROS 还凭借以开发者为中心的设计脱颖而出，提供简化的 API 和全面的文档，使其更易于使用，相比其他机器人中间件也更便于开发者上手 [18, 27]。该框架的跨平台兼容性使其能够在多种操作系统上无缝运行，包括 Linux、macOS 和 Windows，从而拓宽了其在不同开发环境中的适用范围。此外，Meta-ROS 集成了 Zenoh 等先进技术，用于实现高性能数据分发 [13] ，并采用先进的消息队列系统来确保高效的任务调度 [36]。这些特性共同促进了低延迟通信，优化了实时机器人应用的性能 [26, 32]。

在数据处理方面，Meta-ROS展示了其支持多种数据类型的能力——例如文本、图像、音频和视频——这使其特别适用于多媒体密集型机器人应用[37]。与其他框架（如ROS 2 [18],）不同，后者在处理复杂多媒体序列化时面临挑战，而Meta-ROS能够有效管理大规模异构数据集，且不会牺牲性能[33]。这种适应性对于涉及机器人视觉、自然语言处理和实时音视频处理 [16, 37], 的用例尤为有益，在这些场景中，高性能与资源优化至关重要。

我们还评估了ROS1 [27], 、ROS2 [18], 和Meta-ROS中共享库对象（.so文件）的内存大小。这些共享库在执行期间由节点动态加载，并不直接链接到可执行文件，因此对于估算总体内存消耗至关重要。表3汇总了这些库的内存使用情况，并纳入了发布/订阅传输层的数据，从而揭示了各框架的内存占用。

表 3. ROS1、ROS2 和 Meta-ROS 的 .so 文件内存占用

框架	DDS [KB]	抽象层 [KB]	总计 [MB]
ROS1	2,206	-	2.26
ROS2 Connext	11,535	9,645	21.18
ROS2 OpenSplice	3,837	14,117	17.95
ROS2 FastRTPS	1,324	3,953	5.28
Meta-ROS	4,834	6,355	11.19

Meta-ROS 还通过先进的错误处理机制（包括重试和故障转移协议）提升了可靠性，确保消息传递，并在网络故障或节点崩溃期间将系统停机时间降至最低 [12]。在模拟网络中断和节点故障等极端条件下进行的压力测试表明，Meta-ROS 能够保持数据完整性并迅速恢复，从而确保任务关键型应用持续运行 [5]。

此外，Meta-ROS 与云原生技术的集成促进了资源的优化分配和可扩展性，使其非常适合大规模机器人部署 [9, 14]。这种云边协同使边缘设备与云平台能够无缝交互 [8, 10, 34], ，进一步提升了 Meta-ROS 的性能和可扩展性。关键协议消息的传输开销可通过以下公式量化：

$$\text{Wire Overhead} = \frac{S_{\text{total}}}{S_{\text{payload}}} \tag{7}$$

其中 S_{total} 表示消息总大小（首部 + 有效载荷），单位为字节； S_{payload} 表示有效载荷（实际数据）的大小，单位为字节。这项指标有助于评估数据传输效率，具体做法是显示协议首部占消息的比例。传输开销越低，网络带宽的利用效率就越高，每条消息传输的数据也越多。图9直观地比较了不同框架中关键协议消息的传输开销。该比较凸显了 Meta-ROS 在保持高性能的同时最大限度降低传输开销的能力。

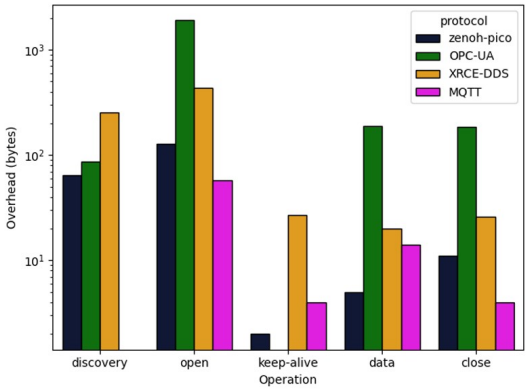

图 9. 跨框架的传输开销比较 rks.

与 ROS 等其他框架相比，Meta-ROS 展现出显著优势，包括更高的消息传输速率、更低的延迟以及更优的资源利用率 [15, 20]。这些改进，加上其开发者友好的设计、跨平台支持，以及 Zenoh [13] 和消息队列 [36], 等先进性能特性，使 Meta-ROS 成为机器人学和人工智能集成系统中实时通信的有前景解决方案。总体而言，Meta-ROS 的性能基准测试、容错能力、多功能数据处理能力以及无缝的云边集成 [9, 11]，巩固了其作为尖端中间件解决方案的地位。该系统在可扩展性、可靠性、性能和易于开发等多个维度均表现优异，使其成为下一代机器人系统和人工智能应用 [7]的理想选择。

6 结论

Meta-ROS代表着机器人中间件发展历程中的一次重大飞跃 [18, 27]。它的首次发布不仅实现了提高通信速率、优化带宽利用率和扩大数据类型支持范围等宏伟目标，还为机器人领域的开发者友好型工具树立了新的标杆。只需一行命令即可安装，并能与 ROS 2 代码库无缝集成 [18], ，Meta-ROS 简化了配置和部署机器人框架这一传统上复杂的任务 [17, 25]。成功申请美国专利彰显了其创新方法，也再次证明了它有望重新定义机器人领域的通信标准。Meta-ROS 已在 PyPI 上发布，现已面向全球用户开放，助力世界各地的开发者在各种机器人应用中充分发挥其优势 [29]。

展望未来 ，Meta-ROS 的未来蕴藏着更大的潜力。与云平台的集成将拓展 d
其在不同领域的实用性 分布式系统 [8–11], 使机器人能够利用基于云的资源进行计算 al

任务 [30, 39]。通过 Docker 和 Kubernetes 实现容器化，将使其更具可扩展性和适应性，以便部署在动态的多机器人环境中 [7, 14]。此外，通过将项目开源，我们希望培育一个充满活力的贡献者与合作者生态系统，他们能够增强框架的功能，并将其应用拓展到新的领域 [27]。

总之，Meta-ROS是一个全面而先进的框架，为前沿机器人应用的开发奠定了坚实基础 [1, 6]。凭借强调可扩展性、社区参与和技术进步的路线图，Meta-ROS有望推动创新，并进一步拓展机器人学的边界 [21, 31], ，为未来更智能、更高效的机器人系统铺平道路。

致谢

本研究得到了位于印度班加罗尔（邮编：560085）的PES University计算机科学与工程系的宝贵支持。他们的指导与资源对本工作的顺利完成起到了关键作用。

参考文献

[1] Saeid Dehnavi、Martijn Koedam、Andrew Nelson等. 2021. CompROS：一种基于ROS2的可组合实时嵌入式机器人开发架构. 载于*2021*年*IEEE/RSJ*国际智能机器人与系统会议（*IROS*）论文集. IEEE，布拉格，捷克共和国. [2] B. Dieber、B. Breiling、S. Taurer、S. Kacianka、S. Rass和P. Schartner. 2017. 机器人操作系统的安全性. 《机器人与自主系统》 98（2017年12月），192–203. [3] B. Dieber、B. Breiling、S. Taurer、S. Kacianka、S. Rass和P. Schartner. 2017. 机器人操作系统的安全性. 《机器人与自主系统》 98（2017年12月），192–203. [4] Lukas Dust和Saad Mubeen. 2023. 使用ROS 2的周期系统动态优先级调度. 载于计算机系统工程. *ECBS 2023*. 《计算机科学讲义》，第14390卷. 施普林格，Cham. [5] Lukas Johannes Dust、Emil Persson等. 2022. 使用ROS2的集中式多智能体机器人系统通信处理定量分析. 载于*2022*年*IEEE*第*20*届工业信息学国际会议（*INDIN*）论文集. IEEE，珀斯，澳大利亚. [6] Manele Ait Habouche、Mickaël Kerboeuf、Goulven Guillou和J ean-Philippe Babau. 2024. MoDD：面向无人机系统数据采集的模型驱动框架. 载于*2024*年第*50*届*Euromicro*软件工程与高级应用会议（*SEAA*）论文集. 18–26. [7] Lucas Hernandez教授和Anna Kim博士. 2024. 群体机器人学：集体行为与可扩展性挑战. 《机器人学：科学与政策》 （2024）. [8] G. Hu、P. Tay和Y. Wen. 2012. 云机器人：架构、挑战与应用. （2012）. [9] Jeffrey Ichnowski、Kaiyuan Chen等.

2023 2 2 *2205 09778*. *[ ]* 2023

FogROS：一种使用ROS的云与雾机器人自适应平台. *arXiv:* . 计算机科学.*RO*（4月）. [10] B. Kehoe、S. Patil、P. Abbeel和K. Goldberg. 2015. 云机器人与自动化研究综述. 《*IEEE*自动化科学与工程汇刊》 12, 2（2015年4月），398–409. [11] S. Lee、Y. Choi和J. Jeon. 2024. 通过ROS2与AWS云增强自动驾驶系统：车路协同交互与高性能计算数据处理. 载于*2024*年*IEEE*第*33*届工业电子国际研讨会（*ISIE*）论文集. 1–6. [12] M. Li等. 2016. ALLIANCE-ROS：面向容错协作多机器人系统的ROS软件架构. 载于*PRICAI 2016*：人工智能趋势. 《计算机科学讲义》，第9810卷. 施普林格，Cham. [13] Wen-Yew Liang、Yuyuan Yuan和Hsiang-Jui Lin. 2023. 关于Zenoh、MQTT、Kafka和DDS吞吐量与延迟的性能研究. *arXiv:2303.09419 [*计算机科学.*DC]*（2023年3月）. [14] Y. Liu、Aitor Hernandez Herranz和R. C. Sundin. 2024. *RoboKube*：为机器人学的云原生演进奠定新基础. 技术报告. [15] D. Logavaseekaran和R. Billava. 2023. *ROS2*性能分析. 技术报告. 基于仿真的工程实验室，威斯康星大学麦迪逊分校，麦迪逊，威斯康星州，美国.[16] Yangzesheng Lu、Bingjie Xu、Jingxiang Wang、Qinglei Bu、Mark Leach和Jie Sun. 2024. 一种用于目标检测与抓取的自主家用机器人. 载于*2024*年第*29*届国际自动化与计算会议（*ICAC*）论文集. 1–6. [17] J. Luis. 2010. 使用移动机器人编程工具包开发科学应用. [18] S. Macenski、T. Foote、B. Gerkey、C. Lalancette和W. Woodall. 2022. 机器人操作系统 2：设计、架构与实际应用. 《科学机器人学》 7, 66（2022年5月）.

[19] Steve Macenski、Alberto Soragna、Michael Carroll和Zhenpeng Ge. 2023. ROS 2节点组合对机器人系统的影响. arXiv:2305.*09933 [*计算机科学.RO*]*（2023年5月）. [20] Y. Maruyama、S. Kato和T. Azumi. 2016. ROS2性能探索. 载于2016年嵌入式软件国际会议（EMSOFT）论文集. IEEE，匹兹堡，宾夕法尼亚州，美国. [21] L. Meier、D. Honegger和M. Pollefeys. [无日期]. PX4：面向深度嵌入式平台的基于节点的多线程开源机器人框架. 可获取：https://people.inf.ethz.ch/pomarc/pubs/MeierICRA15.pdf.

[22] 2019 2019 10 48550 1909 09662

[23] 2022 *2*

I. D. Miller等. . 使用多足式机器人进行矿井隧道探索. arXiv（1月）. doi: . /arxiv. . S. Mobaiyen. . 机器人操作系统（ROS）在实时系统中的系统性差距分析. 博士学位论文.

[24] 1983 71, 7 1983 872 884 10 1109 1983 12684 H.

[25] 2003 P. Moravec. . 斯坦福车与CMU漫游车.《IEEE汇刊》（7月），–. doi: . /PROC. . J. Oh、J. Park、G. Jung和S. Kang. . 支持标准家庭网络中间件之间无缝互操作性的基于CORBA的核心中间件架构. 载于2003年IEEE国际消费电子会议（ICCE）论文集. IEEE，洛杉矶，加利福尼亚州，美国. [26] L. Puck等. 2020. 在Linux上使用ROS2实现实时机器人控制的分布式同步配置. 载于2022年IEEE第18届自动化科学与工程国际会议（CASE）论文集. IEEE，1287–1293. [27] M. Quigley、K. Conley、B. P. Gerkey、J. Faust和A. Y. Ng. 2009. ROS：开源机器人操作系统. ResearchGate 3（2009）. [28] Francesco Semeraro、Jon Carberry、James Leadbetter和Angelo Cangelosi. 2024. 好事成三：机器人响应性对多用户人机协作中工作负荷与信任的影响. 载于2024年IEEE/RSJ国际智能机器人与系统会议（IROS）论文集. 2471–2478. [29] M. A. Sheth、Sachin Bhosale先生、Muabid Burondkar先生和助理教授. 2021. 机器人学新纪元研究论文. [30] A. Sorokin、D. Berenson、S. Srinivasa和M. Hebert. [无日期]. 人助机器人、机器人助人：面向新型物体抓取的众包. 访问日期：2025年1月10日. [31] J. Staschulat和R. Lange. [无日期]. rclc执行器：微控制器上ROS应用的领域特定确定性调度机制：进行中的工作. 访问日期：2025年1月10日. [32] Harun Teper、Tobias Betz、Georg Von Der Brüggen、Kuan-Hsun Chen等. 2023. 时序感知的ROS 2架构与系统优化. 载于2023年IEEE第29届嵌入式与实时计算系统及应用国际会议（RTCSA）论文集. IEEE，新潟，日本. [33] Yu-Ping Wang、Yuejiang Dong和Gang Tan. 2022. ROS-SF：一种使用无序列化消息的透明高效ROS中间件. 载于第23届ACM/IFIP国际中间件会议（Middleware '22）论文集. ACM. [34] Z. Wang、S. Yang、X. Xiang、A. Vasilijevic、N. Miskovic和D. Nad. 2020. 面向无人水面艇的基于云的远程控制框架. IFAC-PapersOnLine 53, 2（2020），14564–14569. [35] R. White、G. Caiazza、H. I. Christensen和A. Cortesi. 2018. 面向机器人系统的程序化配置访问控制. arXiv:1810.*08125 [*计算机科学.RO*]*（2018年10月）.[36] Chenhao Wu、Ruoxiang Li、Naijun Zhan和Nan Guan. 2024. ROS中最晚时间消息同步策略的建模与分析.《IEEE集成电路与系统计算机辅助设计汇刊》43, 11（2024），3576–3587. [37] Takumi Yamamoto、Kotaro Hoshiba、Benjamin Yen和Kazuhiro Nakadai. 2024. 基于机器人操作系统的多架无人机声源定位网络实现. 载于2024年亚太信号与信息处理协会年度峰会暨会议（APSIPA ASC）论文集. 1–6. [38] Y. Ye、Z. Nie、X.-J. Liu、F. Xie、Z. Li和Z. Wang. 2023. ROS2实时性能优化与评估.《中国机械工程学报》36, 1（2023年12月）. [39] C. Yun. [无日期]. 用于构建机器人车队管理应用的AWS IoT RoboRunner. 访问日期：2022年2月11日.

收稿日期：2007年2月20日；修订日期：2009年3月12日；接收日期：2009年6月5日