The Journal of Open Source Software

Links and Nodes: Middleware for distributed real-time robotic systems

Florian Schmidt [1*], **Johannes Nix**[1*], **Maximilian Mühlbauer** [1], **Jan Cremer**[1], **Maxime Chalon**[1], **Timo Bachmann** [1], **and Antonin Raffin** [1]

1 DLR German Aerospace Center, Institute of Robotics and Mechatronics, Germany [ROR] * These authors contributed equally.

DOI: 10.21105/joss.10777

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @nitishsanghi
- @brpat07

Submitted: 19 June 2026
Published: 19 August 2026

Summary

Links and Nodes (LN) is open-source middleware for building distributed real-time robotic systems. Middleware handles communication and coordination between programs, so developers can use common interaction patterns instead of rebuilding them in each component. LN combines process management with fast publish/subscribe messaging and request/response services, so complex multi-process systems can be reproduced from one text file. Developed internally at DLR from August 2010, released as open source in 2015, and hosted in its current public GitLab repository since 2020, LN supports robotics applications requiring deterministic behavior, low-latency communication, and coordinated process control across hosts and programming languages.

Statement of need

Modern robotic systems are typically composed of many independent processes running on several computers. They must exchange data in real time, manage startup and dependency ordering, and remain reproducible across teams and deployments. Several constraints motivate an integrated middleware system:

- **Process isolation**: Safety and reliability drive separate processes rather than threads, requiring robust inter-process communication and lifecycle control.
- **Language heterogeneity**: Robotics software often spans C, C++, Python, and MATLAB/Simulink, making language-agnostic APIs essential.
- **Configuration scale**: Large systems need one version-controlled source of truth for processes, communication endpoints, and dependencies.
- **Network transparency**: Components should communicate the same way on one computer or across a network.
- **Real-time constraints**: Priority management and low-latency transport are critical for control and perception workloads.

Existing tools cover subsets of these needs. ROS (Quigley et al., 2009) and ROS 2 (Macenski et al., 2022) provide strong messaging ecosystems but manage processes differently and often bring extensive dependencies. ZeroMQ (Hintjens, 2013) offers flexible transport patterns but leaves discovery and process control to the application. DDS (*Data Distribution Service*, 2006) provides a standardized, data-centric publish/subscribe model with rich QoS options, but can be complex to configure for embedded real-time systems. LN targets this intersection by combining process supervision, communication topology, and runtime introspection in one reproducible configuration.

State of the field

Robotics middleware spans from message libraries to full system frameworks. ROS 1 was already influential when LN began, but its original assumptions centered on a single robot with workstation-class resources, good connectivity, and no general real-time requirements beyond special-purpose handling (Gerkey, 2014; Quigley et al., 2009). ROS 2 was initiated in June 2014 specifically to address use cases such as real-time systems, non-ideal networks, embedded platforms, multi-robot deployments, and more structured lifecycle management (Gerkey, 2014; Macenski et al., 2022). This history explains why LN was built, but today's relevant comparison is what remains distinctive.

DDS was the strongest established option for real-time distributed communication. Its decentralized discovery, UDP-based transport model, and rich QoS controls made it suitable for both high-level distributed integration and embedded real-time applications (*Data Distribution Service*, 2006; Woodall, 2014a). That technical credibility was one reason why ROS 2 later adopted DDS. However, the same ROS 2 design analysis also emphasized that DDS's flexibility comes with substantial API and configuration complexity, and that users often need vendor-specific knowledge once they move beyond the common subset (Woodall, 2014a). For a group needing dependable, repeatable architecture, that complexity was costly.

ZeroMQ occupied the opposite end of the design space. It offered a fast, brokerless messaging library with broad language support (Hintjens, 2013), making it attractive as a low-level building block, but not as complete robotics middleware. OSRF's later prototype work on a ZeroMQ-based ROS stack found that discovery, higher-level communication semantics, and transport-serialization glue still had to be maintained separately, and judged the prototype unsuitable for soft real-time scenarios because it relied on transports such as TCP or PGM (Woodall, 2014b). CORBA (*Common Object Request Broker Architecture*, 2004) provided mature remote-object support and language bindings, but its RPC-oriented model and associated complexity were poorly matched to high-rate robotics data flows.

LN was therefore built not because of a lack of alternatives, but because neither message libraries nor full DDS/ROS ecosystems matched needs that recurred in DLR robotics systems. Its distinctive contribution today is the integration of low-latency communication, process management, reproducible deployment configuration, and debugging support in one middleware system. Rather than requiring users to combine a transport library, launch scripts, and separate supervision tools, LN treats runtime topology and communication topology as one system. This is visible in the single configuration model, daemon-managed process supervision, shared-memory data paths for local topics, and in-memory ring-buffer logging.

Software design

LN is organized around the core library (`libln`), the runtime daemon (`ln_daemon`), and the manager application (`ln_manager`). Auxiliary tooling includes the message-definition generator (`ln_generate`) and the message-definition libraries used by the topic and service APIs.

- **Core library (libln)**: A C library implementing communication primitives, services, and a client API. It is LGPLv3-licensed to allow dynamic linking into proprietary applications, while the rest of LN is GPLv3.
- **Runtime daemon (ln_daemon)**: A C++ process managing remote process spawning, priority handling, and lifecycle control via a versioned protocol.
- **Manager (ln_manager)**: A Python application with GUI and CLI interfaces, configuration parsing for `.lnc` files, and high-level scripting hooks.

The design grew from practical deployments and feasibility constraints rather than a prescriptive standard, leading to several deliberate trade-offs. LN keeps processes isolated instead of collapsing functionality into one address space. That increases middleware needs but improves

fault containment, supports mixed-language systems, and matches the reality of robotic stacks that combine controllers, planners, perception modules, and operator interfaces. LN also deliberately couples communication and process supervision. This is less general than a pure messaging library, but ensures that one configuration specifies what runs, where, and how components exchange data.

Two orthogonal communication paradigms emerged:

- **Topics (publish/subscribe)**: Optimized for low-latency signal transport. Local traffic uses shared memory, UDP provides network transparency, and TCP is available when reliable topic delivery is required. Publisher-host topic traffic can be logged into an in-memory ring buffer for lightweight introspection without external tooling.
- **Services (request/response)**: Optimized for synchronization and reliable calls. Unix domain sockets are used locally for efficiency, and TCP provides network-transparent service calls.

Process management uses the same configuration model: processes declare dependencies and priorities, while the manager monitors state and output in real time. This keeps control-plane and data-plane concerns in one workflow, reducing gaps between development and deployment. Compared with DDS-style systems, LN intentionally exposes a smaller and more opinionated configuration surface. Rather than asking users to tune many per-endpoint QoS policies, LN offers a constrained set of transport choices: shared memory for local low-latency topics, UDP for network-transparent topic transport, and TCP when reliability is more important than minimum latency. This narrower design reduced configuration burden and made behavior easier to reason about for researchers who needed dependable system integration more than maximum middleware generality.

Another important design choice is the split between a small C communication core and higher-level tooling in Python. The C library keeps the runtime lightweight and embeddable, while the Python manager provides interactive inspection, configuration handling, and scripting without imposing Python on deployed client code. This separation helped LN remain usable across heterogeneous systems, including deployments that combine real-time-adjacent native code with more flexible supervisory tooling. The security model is similarly pragmatic: LN supports manager-daemon authentication, but it does not try to provide a full encrypted transport stack for all traffic. That keeps the runtime small and deployment-oriented, but means LN is intended for trusted or protected network environments.

Linux, especially Debian/Ubuntu, is LN's primary open-source installation, build, and reviewer testing platform; QNX, Windows, and VxWorks are additional deployment targets. SCons and distribution-specific helpers handle builds where required. A small system-library footprint keeps deployments lightweight.

Research impact statement

LN has been used for over 15 years in DLR robotics research, including mobile manipulation, human-robot interaction, and real-time perception pipelines. Its single-configuration approach enables reproducible system setups across teams, and its low-latency transports support control loops that require deterministic behavior. The software evolved alongside real deployments, leading to a focus on debuggability, predictable performance, and stable APIs rather than rapid feature expansion.

LN has enabled research projects that rely on synchronized sensor fusion, modular control architectures, and distributed simulation environments. Representative published uses include humanoid teleoperation and Surface Avatar mission software (Bauer et al., 2025; Beik-Mohammadi et al., 2021), space-robotics control stacks for the SpaceDREAM arm and a universal robotic joint (Bahls et al., 2025; Mühlbauer et al., 2025), educational and experimental rover platforms (Bekkers et al., 2023), humanoid real-time control experiments (Loeffl et al.,

2016), and autonomous rover operation and task-execution studies (Brunner et al., 2018, 2019; Schuster et al., 2017). The platform's process-management capabilities allow complex experiments to be launched and monitored consistently, which reduces integration overhead and supports repeatable experimentation across these domains. The source repository is hosted at https://gitlab.com/links_and_nodes/links_and_nodes. Additional repositories provide Rust bindings for LN and tooling for the configuration language (Tree-sitter grammar and a language server). Developed since August 2010, LN has many tagged releases, an automated test tree covering local and remote publish/subscribe, services, console behavior, networked setups, and clean-build scenarios, plus extensive user and reference documentation. These artifacts support research reuse because LN's contribution is not only the code itself, but also the operational knowledge required to keep a distributed robotics middleware system maintainable over many years.

AI usage disclosure

Software changes before March 2026 were written without AI agents. Since March 2026, Florian Schmidt used OpenAI Codex with GPT-5.5 for code or documentation in `python/links_and_nodes/scope/`, `.gitlab*`, the web UI and manager-started VNC sessions, Python parameter blocks, DSA key generation, sync/async manager clients, network-usage statistics, new documentation, and this manuscript. These contributions were steered, reviewed, edited, and fact-checked by Florian Schmidt; the GitLab CI changes were also reviewed by Antonin Raffin.

Acknowledgements

We thank contributors to Links and Nodes, particularly those at DLR's Institute of Robotics and Mechatronics who used and improved it in real robotic deployments. We also acknowledge the open-source community behind Linux, GCC, and various Python packages used by this project.

References

Bahls, T., Beyer, A., Sedlmayr, H.-J., Stemmer, A., Burger, R., & Moser, S. (2025). The communication and computation architecture for a universal space robotic joint. *2025 IEEE Aerospace Conference (AERO 2025)*, 1–10. https://doi.org/10.1109/AERO63441.2025.11068565

Bauer, A. S., Köpken, A., Batti, N., Butterfaß, J., Ehlert, T. H., Friedl, W., Gumpert, T., Lay, F. S., Luo, X., Manaparampil, A. N., Mayershofer, L., Raffin, A., Schmidt, F., Seidel, D., Exter, E. den, Luz, R., Schmidt, A., Schmaus, P., Leidner, D., … Lii, N. Y.-S. (2025). System architecture and design considerations for the humanoid robot Rollin' Justin in context of the Surface Avatar mission. *2025 IEEE Aerospace Conference (AERO 2025)*, 1–14. https://doi.org/10.1109/AERO63441.2025.11068600

Beik-Mohammadi, H., Kerzel, M., Pleintinger, B., Hulin, T., Reisich, P., Schmidt, A., Pereira, A., Wermter, S., & Lii, N. Y. (2021). *Model mediated teleoperation with a hand-arm exoskeleton in long time delays using reinforcement learning*. arXiv preprint. https://doi.org/10.48550/arXiv.2107.00359

Bekkers, S., Bettendorf, M. T., Reill, J., Giubilato, R., & Wedler, A. (2023, October). The Lunar Rover Mini: Towards a versatile, open-source mobile robotic platform for educational and experimental purposes. *17th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2023)*. https://elib.dlr.de/202313/

Brunner, S. G., Dömel, A., Lehner, P., Beetz, M., & Stulp, F. (2019). Autonomous parallelization of resource-aware robotic task nodes. *IEEE Robotics and Automation Letters*, *4*(3), 2599–2606. https://doi.org/10.1109/LRA.2019.2894463

Brunner, S. G., Lehner, P., Schuster, M. J., Riedel, S., Belder, R., Wedler, A., Leidner, D., Beetz, M., & Stulp, F. (2018). Design, execution, and postmortem analysis of prolonged autonomous robot operations. *IEEE Robotics and Automation Letters*, *3*(2), 1056–1063. https://doi.org/10.1109/LRA.2018.2794580

Common object request broker architecture: Core specification (Version 3.0.3). (2004). Object Management Group. https://www.omg.org/spec/CORBA/3.0.3/PDF

Data distribution service (Version 1.2). (2006). Object Management Group. https://www.omg.org/spec/DDS/1.2/PDF

Gerkey, B. (2014). *Why ROS 2?* ROS 2 design article. https://design.ros2.org/articles/why_ros2.html

Hintjens, P. (2013). *ZeroMQ: Messaging for many applications*. O'Reilly Media, Inc.

Loeffl, F. C., Werner, A., Lakatos, D., Reinecke, J., Wolf, S., Burger, R., Gumpert, T., Schmidt, F., Ott, C., Grebenstein, M., & Albu-Schäffer, A. O. (2016). The DLR C-Runner: Concept, design and experiments. *2016 IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 758–765. https://doi.org/10.1109/HUMANOIDS.2016.7803359

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, *7*(66), eabm6074. https://doi.org/10.1126/scirobotics.abm6074

Mühlbauer, M. S., Chalon, M., Ulmer, M., & Albu-Schäffer, A. O. (2025). Software for the SpaceDREAM robotic arm. *2025 IEEE Aerospace Conference (AERO 2025)*, 1–10. https://doi.org/10.1109/AERO63441.2025.11068537

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., & others. (2009). ROS: An open-source Robot Operating System. *ICRA Workshop on Open Source Software*, *3*, 5.

Schuster, M. J., Brunner, S. G., Bussmann, K., Büttner, S., Dömel, A., Hellerer, M., Lehner, H., Lehner, P., Porges, O., Reill, J., Riedel, S., Vayugundla, M., Vodermayer, B., Bodenmüller, T., Brand, C., Friedl, W., Grixa, I., Hirschmüller, H., Kaßecker, M., … Wedler, A. (2017). Towards autonomous planetary exploration: The Lightweight Rover Unit (LRU), its success in the SpaceBotCamp Challenge, and beyond. *Journal of Intelligent and Robotic Systems*, *93*(3–4), 461–494. https://doi.org/10.1007/s10846-017-0680-9

Woodall, W. (2014a). *ROS on DDS*. ROS 2 design article. https://design.ros2.org/articles/ros_on_dds.html

Woodall, W. (2014b). *ZeroMQ and friends*. ROS 2 design article. https://design.ros2.org/articles/ros_with_zeromq.html