SINTA Jurnal Sistem Informasi dan Teknologi Komputasi
Volume 3 Nomor 3 Juli 2026 [page: 182-189]
DOI: 10.61124/sinta.v3i3.309

ANALISIS SISTEM KOMUNIKASI ANTAR NODE PADA ROBOT MENGGUNAKAN ROBOT OPERATING SYSTEM

Muhammad Rafif[1], Yulindon Khaidir[1]
[1]Program Studi Teknik Telekomunikasi, Politeknik Negeri Padang, Indonesia
*e-mail: muhammadrafif899@gmail.com

Abstrak
Penelitian ini bertujuan untuk menganalisis performa sistem komunikasi antar node pada Robot Operating System (ROS 1) dengan fokus pada mekanisme publish–subscribe. Metode yang digunakan adalah pendekatan eksperimental dengan memvariasikan frekuensi pengiriman pesan (10 Hz, 50 Hz, dan 100 Hz), ukuran message (kecil, sedang, dan besar), serta jumlah subscriber (1 hingga 3 node). Parameter yang dianalisis meliputi latensi komunikasi, throughput data, dan penggunaan CPU. Hasil pengujian menunjukkan bahwa latensi komunikasi berada pada kisaran rendah, yaitu di bawah 1.5 ms, dan menurun hingga 0.77 ms pada frekuensi 100 Hz. Throughput meningkat secara linear terhadap ukuran message, dari sekitar 90 B/s pada data kecil hingga lebih dari 100 KB/s pada data besar. Sementara itu, penggunaan CPU menunjukkan peningkatan seiring bertambahnya jumlah subscriber dan frekuensi pengiriman, namun tetap dalam batas yang stabil pada sistem embedded. Hasil ini menunjukkan bahwa ROS 1 memiliki performa komunikasi yang responsif, stabil, dan skalabel untuk mendukung sistem robotik modular. Penelitian ini memberikan kontribusi dalam bentuk evaluasi sistematis terhadap karakteristik komunikasi ROS 1 yang dapat menjadi referensi dalam pengembangan sistem robotika berbasis middleware.

Kata kunci: komunikasi antar node; latensi; performa sistem; ROS; throughput

Abstract
This study aims to analyze the performance of the inter-node communication system in the Robot Operating System (ROS 1) with a focus on the publish-subscribe mechanism. The method used is an experimental approach by varying the message sending frequency (10 Hz, 50 Hz, and 100 Hz), message size (small, medium, and large), and the number of subscribers (1 to 3 nodes). The parameters described include communication latency, data throughput, and CPU usage. The test results show that the communication latency is in the low range, which is below 1.5 ms, and decreases to 0.77 ms at a frequency of 100 Hz. The throughput increases linearly with the message size, from around 90 B/s for small data to more than 100 KB/s for large data. Meanwhile, CPU usage shows an increase with the increase in the number of subscribers and sending frequency, but remains within a stable limit in the embedded system. These results indicate that ROS 1 has responsive, stable, and scalable communication performance to support modular robotic systems. This research provides a contribution in the form of a systematic evaluation of the communication characteristics of ROS 1 which can be a reference in the development of middleware-based robotics systems.

Keywords: inter-node communication; latency; system performance; ROS; throughput

1. PENDAHULUAN

Robot Operating System (ROS) merupakan framework middleware yang banyak digunakan dalam sistem robotika modern karena menyediakan infrastructure komunikasi yang modular dan terdistribusi antar komponen perangkat lunak pada robot [1][2]. Komunikasi antar node dalam ROS didasarkan pada mekanisme publish–subscribe yang memungkinkan proses publisher dan subscriber bertukar data melalui topic tanpa ketergantungan langsung satu sama lain, sehingga mempermudah pengembangan sistem robotik modular dan paralel [1][2]. Berbagai penelitian telah mengkaji aspek teknis komunikasi dalam ROS dan sistem robot lainnya. Misalnya, studi tentang efisiensi energi dalam komunikasi ROS menunjukkan bahwa implementasi node, frekuensi pesan, dan bahasa pemrograman memengaruhi konsumsi energi dan performa antar node dalam framework ROS [3]. Pendekatan perluasan komunikasi ROS melalui ROS Gateway telah dieksplorasi untuk meningkatkan ketersediaan dan interoperabilitas komunikasi ROS di lingkungan jaringan yang heterogen [4]. Selain itu, studi komunikasi performa ROS dalam konteks IoT menggambarkan bagaimana ROS beroperasi dalam sistem yang terhubung dengan perangkat sensor dan aktuator, yang

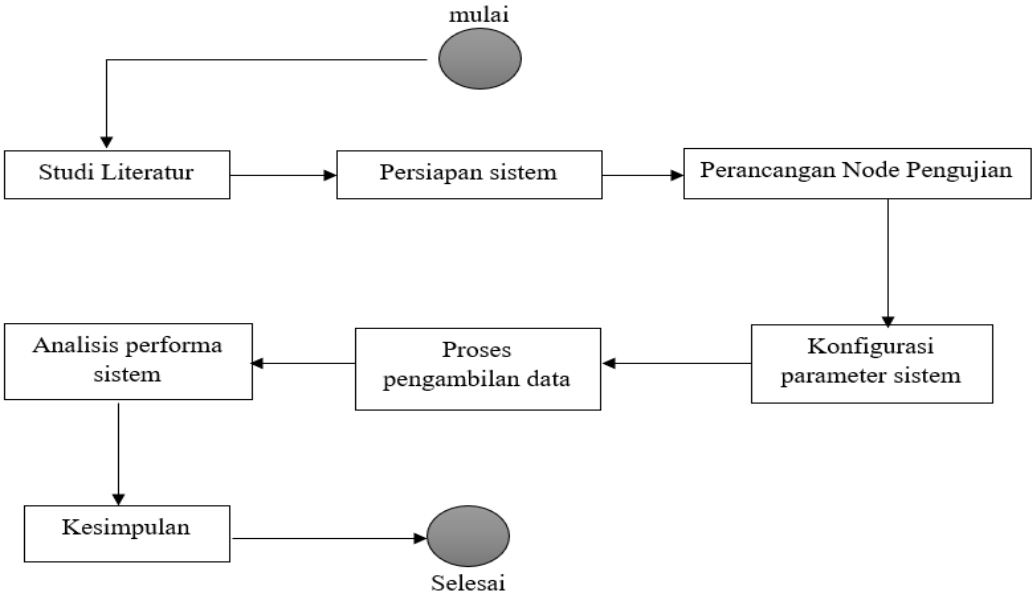
merupakan bagian penting dalam robot jaringan modern [5]. Ulasan komprehensif tentang ROS juga menegaskan bahwa arsitektur ROS mempromosikan modularitas dan fleksibilitas dalam mengintegrasikan berbagai subsistem robotik melalui komunikasi antar proses yang terstandarisasi [1][6]. Dalam studi adopsi ROS, perbedaan fitur komunikasi dan arsitektur antar versi ROS1 dan ROS2 menjadi perhatian yang signifikan dalam komunitas peneliti robot, khususnya bagaimana pengembang melihat kelemahan dan kekuatan masing-masing versi terhadap kebutuhan sistem robot yang kompleks [7][8]. Lebih jauh, kajian sistem software engineering pada ROS menunjukkan bahwa komunikasi antar node adalah bagian penting dalam rekayasa perangkat lunak robot yang efektif dan andal [8].

Namun, sebagian besar penelitian tersebut fokus pada pengembangan atau optimasi komunikasi di ROS2 atau dalam konteks perluasan jaringan, sedangkan studi yang secara sistematis mengevaluasi mekanisme komunikasi antar node pada ROS versi pertama (ROS1) dari perspektif analisis sistem komunikasi itu sendiri, termasuk dampaknya terhadap performa pertukaran data dalam sistem robotik modular, masih relatif terbatas. Penelitian ini bertujuan mengisi kekosongan tersebut dengan melakukan analisis komprehensif terhadap sistem komunikasi antar node pada ROS1, khususnya mekanisme publish–subscribe, struktur topic, dan karakteristik pertukaran pesan dalam skenario robotik modular. Meskipun ROS 1 menyediakan infrastruktur komunikasi yang kuat, ketiadaan arsitektur real-time bawaan seperti pada Data Distribution Service (DDS) di ROS 2 membuat evaluasi latensi dan jitter pada tingkat kernel menjadi sangat krusial [9][10].

2. METODE

2.1 Langkah-langkah Penelitian

Berdasarkan Gambar 1, langkah penelitian dirancang secara sistematis. Flowchart tersebut menggambarkan bahwa penelitian ini dimulai dari tahap studi literatur dilanjutkan dengan persiapan sistem, kemudian perancangan Node pengujian lalu konfigurasi parameter sistem, selanjutnya proses pengambilan data, dan diakhiri dengan analisis performa sistem.



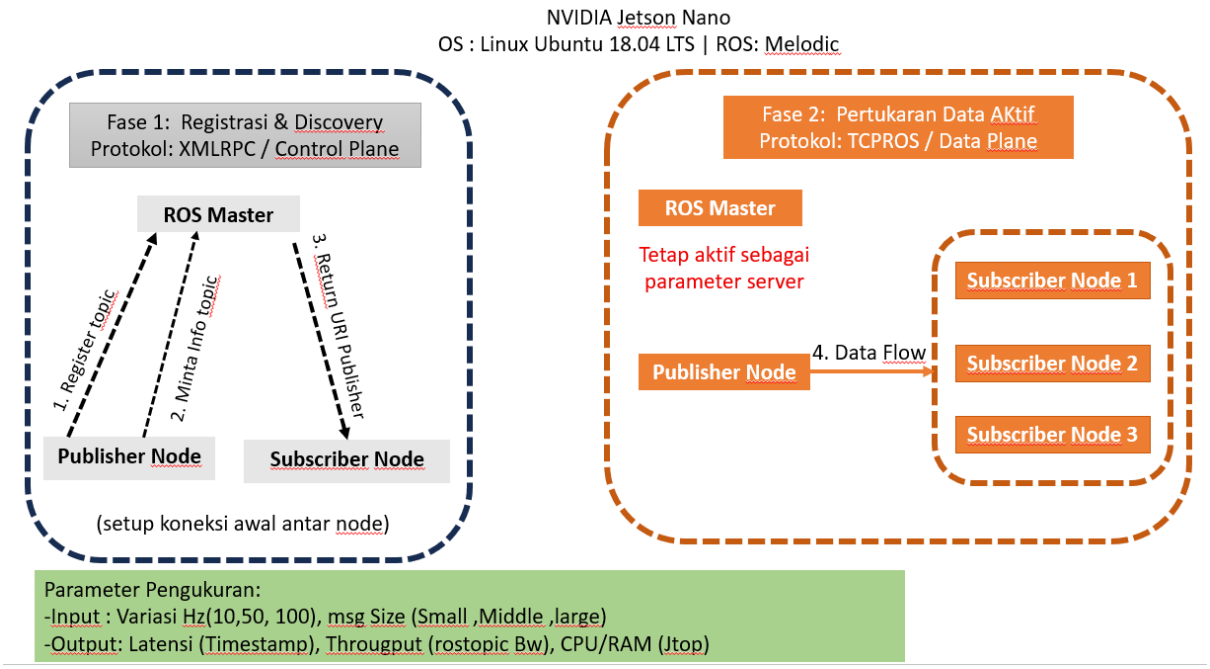
Gambar 1. Flowchart langkah penelitian

2.2 Desain Sistem Pengujian

Penelitian ini menggunakan pendekatan eksperimental untuk menganalisis performa komunikasi antar node pada Robot Operating System (ROS 1). Sistem pengujian dirancang dalam bentuk arsitektur terdistribusi yang terdiri dari beberapa node yang saling berkomunikasi menggunakan mekanisme publish–subscribe. Arsitektur sistem secara keseluruhan terdiri atas satu node sebagai publisher yang bertugas mengirimkan data secara periodik ke suatu topic, satu atau lebih node sebagai subscriber yang menerima data dari topic tersebut, serta ROS Master yang berfungsi sebagai pengatur komunikasi antar node [2][6]. Pengujian dilakukan pada satu perangkat komputasi (single machine) untuk menghindari variabel tambahan

DOI: 10.61124/sinta.v3i3.309

dari jaringan eksternal. Perangkat yang digunakan dapat berupa komputer atau embedded system seperti Jetson Nano.



Gambar 2. Arsitektur Pengujian Sistem Komunikasi ROS 1 pada Jetson Nano

Gambar 2 menunjukkan dua tahapan pengujian, yaitu Fase 1 yang mencakup proses registrasi topik dan penemuan peer di mana ROS Master memberikan URI Publisher kepada subscriber melalui protokol XMLRPC, serta Fase 2 yang merupakan tahap pertukaran data secara langsung (Peer-to-Peer) melalui protokol TCPROS. Pengukuran performa seperti latensi, throughput, dan beban sistem dilakukan sepenuhnya pada Fase 2 dengan memvariasikan frekuensi, ukuran pesan, dan jumlah node subscriber [2][6].

2.3 Skenario Pengujian

Pengujian dilakukan menggunakan beberapa variasi parameter untuk melihat pengaruhnya terhadap performa komunikasi. Skenario pengujian dirancang dengan membagi variasi menjadi tiga aspek utama. Pertama, variasi frekuensi publish ditetapkan sebesar 10 Hz, 50 Hz, dan 100 Hz dengan tujuan menganalisis pengaruh frekuensi terhadap latensi dan beban sistem. Kedua, variasi jumlah subscriber diatur sebanyak 1 subscriber, 2 subscriber, dan 3 subscriber untuk menguji skalabilitas sistem komunikasi ROS 1. Ketiga, variasi ukuran message dikelompokkan menjadi message kecil berupa pesan "hello" (~9 byte), message sedang yang berisi karakter berulang sebanyak 1.000 karakter (~1 KB), dan message besar berupa karakter berulang sebanyak 10.000 karakter (~10 KB) untuk melihat pengaruh dimensi data terhadap throughput dan latency.

2.4 Parameter Pengukuran

Beberapa parameter diukur secara spesifik guna mengevaluasi performa sistem komunikasi ini. Latensi komunikasi diukur berdasarkan perbedaan waktu antara saat message dikirim oleh publisher dan diterima oleh subscriber menggunakan timestamp yang ditambahkan pada pesan. Throughput dan bandwidth aktual dipantau menggunakan tool rostopic bw, sedangkan frekuensi aktual dinilai melalui tool rostopic hz untuk membandingkan stabilitas sistem terhadap frekuensi rencana. Selain itu, tingkat penggunaan CPU dan memori diukur secara langsung menggunakan utilitas jtop.

2.5 Metode Analisis Data

Data yang diperoleh dari hasil pengujian dianalisis menggunakan metode deskriptif dan kuantitatif. Analisis dilakukan dengan mengolah data hasil pengukuran ke dalam bentuk tabel, membuat grafik hubungan antar parameter, membandingkan hasil antar skenario pengujian, serta menginterpretasikan hasil berdasarkan konsep komunikasi terdistribusi pada ROS 1. Hasil analisis kemudian digunakan untuk

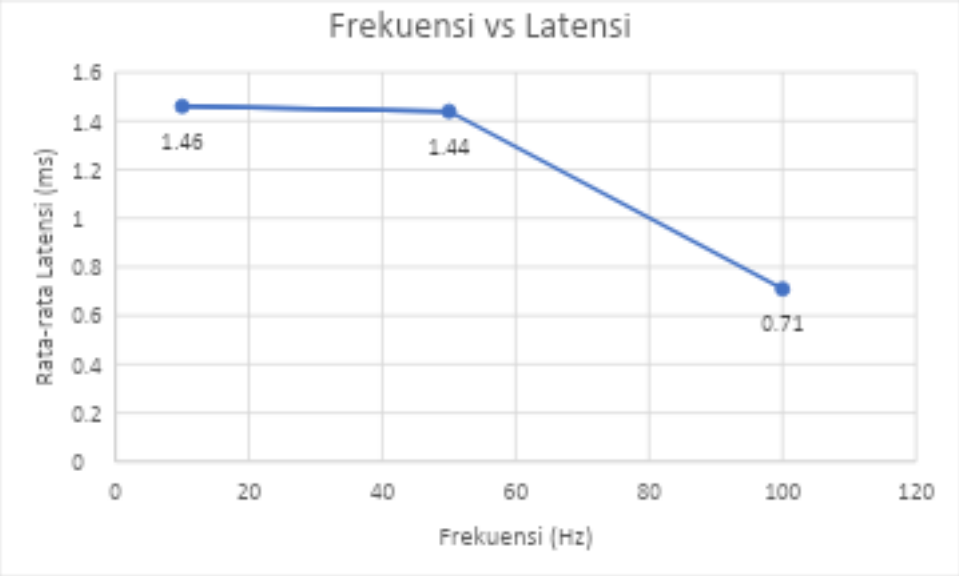
DOI: 10.61124/sinta.v3i3.309

mengevaluasi performa sistem komunikasi dan mengidentifikasi batasan serta karakteristik dari mekanisme publish-subscribe pada ROS 1.

3. HASIL PENELITIAN

3.1 Perbandingan frekuensi dan latensi

Berdasarkan Gambar 3, ditemukan bahwa sistem memiliki latensi yang sangat rendah, yaitu di bawah 1.5 ms untuk seluruh variasi frekuensi. Temuan yang paling signifikan adalah tren penurunan latensi seiring dengan peningkatan frekuensi pengiriman pesan. Pada frekuensi 10 Hz, latensi tercatat sebesar 1.46 ms, namun saat frekuensi ditingkatkan menjadi 100 Hz, nilai latensi justru menurun secara drastis hingga mencapai 0.77 ms [10][11].



Gambar 3. Grafik frekuensi vs latensi

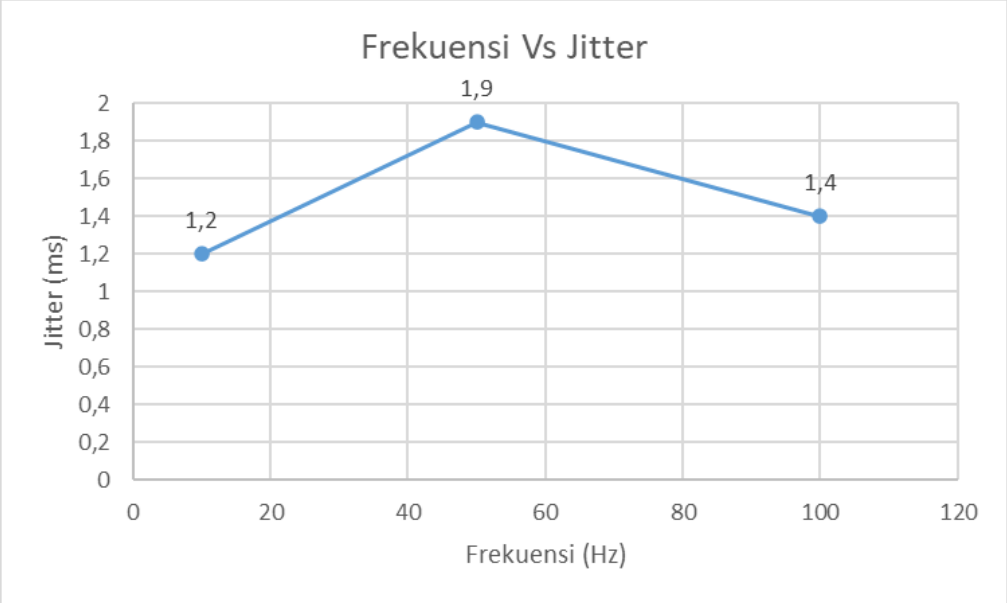
Penurunan latensi pada frekuensi tinggi ini dapat dijelaskan melalui mekanisme process scheduling pada kernel Linux Ubuntu 18.04 yang berjalan di atas platform NVIDIA Jetson Nano. Secara teknis, fenomena ini terjadi akibat aktivitas scheduler di mana pada frekuensi rendah (10 Hz), interval antar pesan (~100 ms) cukup lama sehingga CPU sering kali berada dalam kondisi idle atau low-power state. Hal ini menyebabkan adanya overhead waktu tambahan bagi kernel untuk "membangunkan" proses dan melakukan context switching saat pesan baru tiba. Sebaliknya, pada kondisi steady-state dengan frekuensi 100 Hz, interval antar pesan sangat singkat (~10 ms) yang menjaga callback queue pada ROS tetap aktif dan processor tetap dalam kondisi operasional tinggi (warm state). Hal ini meminimalkan waktu tunggu paket di dalam buffer dan mempercepat eksekusi antrean pesan. Responsivitas di bawah 1 ms pada frekuensi tinggi ini membuktikan bahwa ROS 1 masih sangat andal untuk mendukung aplikasi robotika real-time yang membutuhkan kontrol presisi tinggi.

3.2 Perbandingan Frekuensi dan jitter

Tabel 1. Frekuensi Aktual.

target (Hz)	aktual (hz)	std Dev (s)	Min (s)	Max (s)	Keterangan
10	~ 10.00	0.0012	0.095	0.10	sangat stabil
50	~49.99	0.0019	0.006	0.029	stabil, mulai fluatatif
100	~100.00	0.0014	0.001	0.019	stabil, jitter terkendali

DOI: 10.61124/sinta.v3i3.309



Gambar 4. Grafik frekuensi vs jitter

Tabel 1 menunjukkan bahwa ROS mampu mempertahankan frekuensi komunikasi dengan tingkat akurasi yang sangat tinggi pada berbagai variasi frekuensi. Meskipun terjadi peningkatan jitter pada frekuensi menengah, sistem kembali menunjukkan kestabilan pada frekuensi tinggi akibat tercapainya kondisi steady-state[12]. Hal ini menunjukkan bahwa performa komunikasi ROS cukup robust terhadap peningkatan beban frekuensi

3.3 Througput Komunikasi

Berdasarkan Tabel 2, terlihat bahwa sistem komunikasi ROS 1 menunjukkan konsistensi performa yang sangat linear terhadap beban data yang diberikan. Analisis data kecil pada pengiriman pesan "hello" menghasilkan throughput sebesar 90,6 B/s yang merepresentasikan baseline komunikasi minimal pada mekanisme publish-subscribe. Ketika ukuran pesan ditingkatkan menjadi 1.000 karakter (data sedang), throughput naik menjadi 10.000 B/s. Peningkatan beban sebesar 10 kali lipat dari data sedang ke data besar (10.000 karakter) menghasilkan throughput sebesar 101.000 B/s. Korelasi linear ini membuktikan bahwa protokol TCPROS mampu menangani peningkatan volume data secara proporsional. Rasio peningkatan karakter yang berbanding lurus dengan peningkatan throughput menunjukkan bahwa overhead protokol tetap stabil meskipun payload pesan bertambah. Dengan nilai throughput tertinggi sebesar 101 KB/s pada frekuensi 10 Hz, penggunaan bandwidth ini masih jauh di bawah kapasitas maksimal antarmuka jaringan pada NVIDIA Jetson Nano. Hal ini memberikan ruang (headroom) yang cukup bagi sistem untuk mengintegrasikan data sensor yang lebih kompleks seperti LiDAR atau kamera.

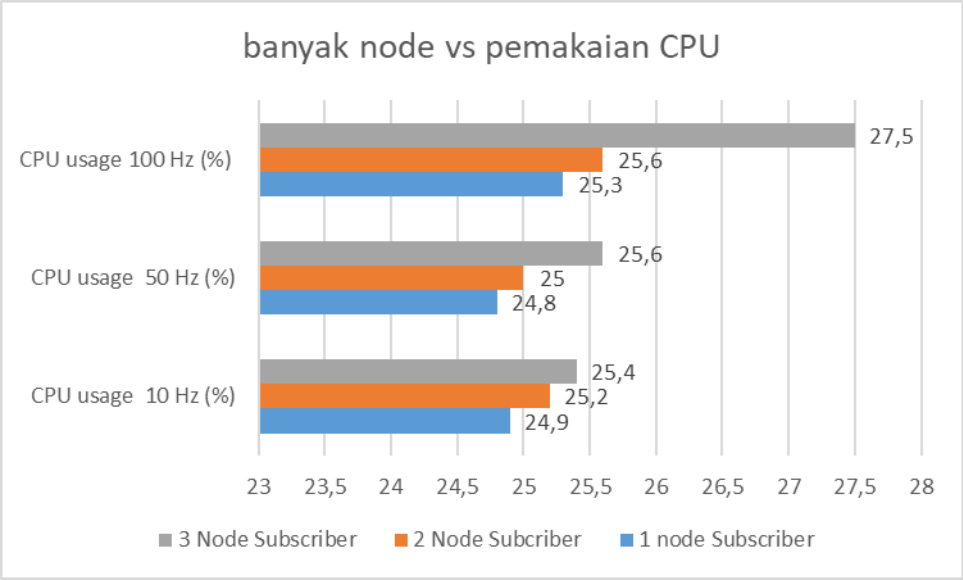
Tabel 2. Throughput Komunikasi pada Frekuensi 10 Hz

Ukuran message	Isi Pesan (Karakter)	Throughput (B/s)
Kecil	“hello”	90,6
Sedang	1.000	10.000
Besar	10.000	101.000

3.4 CPU Usage dan Node

Gambar 5 memberikan gambaran mengenai skalabilitas sistem ROS 1 pada perangkat embedded. Penggunaan CPU berada di angka 24,9% pada kondisi 1 subscriber dengan frekuensi 10 Hz. Ketika jumlah subscriber ditambah menjadi 3 node, beban CPU naik menjadi 27,5%. Kenaikan beban CPU yang hanya sebesar 2,6% meskipun beban kerja meningkat drastis menunjukkan efisiensi mekanisme publish-subscribe ROS 1. Hal ini membuktikan bahwa arsitektur ROS 1 yang bersifat loosely coupled memungkinkan

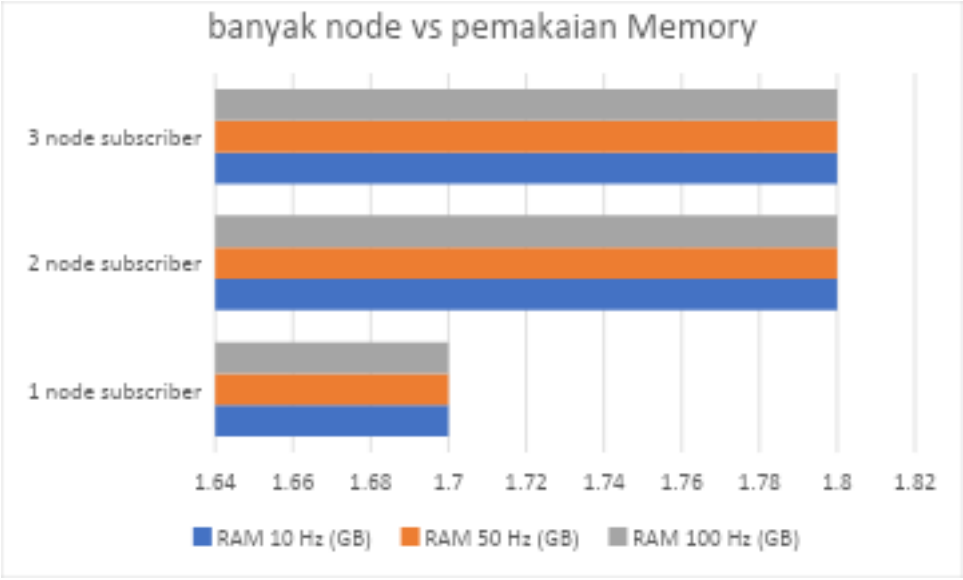
pengembangan sistem robotik modular yang sangat skalabel tanpa mengorbankan performa komputasi secara signifikan.



Gambar 5. Grafik CPU Usage vs node subscriber

3.5 Memory Usage

Gambar 6 menunjukkan bahwa penggunaan memori relatif stabil terhadap variasi frekuensi publish (10 Hz, 50 Hz, dan 100 Hz), dengan nilai berkisar antara 1,7 GB hingga 1,8 GB. Hal ini menunjukkan bahwa pada arsitektur ROS 1, penggunaan memori lebih dipengaruhi oleh jumlah node dibandingkan frekuensi komunikasi



Gambar 6. Grafik Memory Usage vs node subscriber

4. DISKUSI

Hasil eksperimen menunjukkan bahwa arsitektur Robot Operating System (ROS 1) yang bersifat loosely coupled memberikan fleksibilitas tinggi tanpa mengorbankan performa secara signifikan pada perangkat embedded [3][4]. Hal ini dibuktikan melalui pengujian skalabilitas, di mana penambahan jumlah subscriber dari satu menjadi tiga node hanya meningkatkan penggunaan CPU sebesar 2,6%. Efisiensi ini mengonfirmasi bahwa mekanisme publish–subscribe pada ROS 1 memungkinkan pembagian fungsi ke dalam node-node independen secara efektif, yang sangat krusial dalam pengembangan robotika modular seperti robot mobile. Selain itu, stabilitas penggunaan memori yang berkisar antara 1,7 GB hingga 1,8 GB pada berbagai variasi frekuensi menunjukkan bahwa beban utama komunikasi ROS 1 terletak pada pemrosesan CPU, bukan pada alokasi memori. Meskipun throughput meningkat secara linear mencapai 101 KB/s saat menangani data besar, ketersediaan sumber daya pada NVIDIA Jetson Nano masih mencukupi karena nilai tersebut jauh di bawah batas maksimal antarmuka jaringan perangkat. Temuan ini memberikan keyakinan bahwa infrastruktur komunikasi ROS 1 tetap andal untuk mendukung integrasi sensor robotik yang lebih kompleks dengan pertukaran data yang intensif jika dibandingkan dengan protokol komunikasi terdistribusi atau middleware industri lainnya [13][14].

Kajian perbandingan dengan penelitian terdahulu menunjukkan bahwa kajian eksperimental yang secara khusus menganalisis performa komunikasi dasar pada ROS 1 menggunakan skenario publisher–subscriber masih memiliki ruang pengembangan yang besar. Berbeda dengan penelitian Albonico dkk. (2025) yang menitikberatkan pada aspek efisiensi energi lintas bahasa pemrograman [3] atau studi pemetaan tren oleh Albonico dkk. (2023) [15], penelitian ini berfokus langsung pada parameter operasional fundamental. Jika Sasaki dkk. (2025) mengevaluasi performa IoT terdistribusi [5] dan Song & Choi (2024) mengkaji perluasan jaringan via gateway [4], data dalam artikel ini melengkapi dengan menyajikan data terukur latensi, throughput, dan jitter langsung pada single-embedded machine. Temuan ini menonjolkan adanya trade-off yang signifikan antara kecepatan respons sistem dan kestabilan komunikasi. Peningkatan frekuensi pengiriman hingga 100 Hz terbukti mampu menurunkan latensi hingga mencapai titik terendah yaitu 0,77 ms. Namun, di sisi lain, frekuensi tinggi ini juga memicu peningkatan nilai jitter atau variasi waktu antar pesan. Fenomena ini mengindikasikan bahwa semakin tinggi frekuensi publish, semakin besar pula beban kerja pada scheduler kernel untuk menjaga konsistensi waktu pengiriman [10][11]. Beban komputasi yang mencapai puncaknya (27,5%) pada skenario frekuensi 100 Hz dengan tiga subscriber menyarankan bahwa pemilihan frekuensi operasional robot harus disesuaikan secara presisi dengan kebutuhan aplikasi agar efisiensi daya dan komputasi tetap terjaga.

5. KESIMPULAN

Berdasarkan hasil pengujian dan analisis yang telah dilakukan terhadap sistem komunikasi ROS 1 pada perangkat NVIDIA Jetson Nano, dapat disimpulkan bahwa sistem komunikasi ROS 1 menunjukkan responsivitas yang sangat tinggi dengan nilai latensi di bawah 1,5 ms, di mana peningkatan frekuensi dari 10 Hz ke 100 Hz justru menurunkan latensi secara drastis dari 1,46 ms hingga menjadi 0,77 ms karena sistem terjaga dalam kondisi steady-state. Selain itu, protokol TCPROS mampu menangani beban data secara linear dan efisien dengan throughput yang meningkat proporsional dari 90,6 B/s pada pesan kecil hingga mencapai 101.000 B/s (101 KB/s) pada pesan besar (10.000 karakter) tanpa menimbulkan bottleneck pada memori. Skalabilitas arsitektur publish-subscribe ROS 1 juga terbukti sangat optimal, ditunjukkan oleh penambahan jumlah subscriber hingga tiga node yang hanya memberikan beban tambahan CPU sebesar 2,6%, sehingga sangat ideal untuk aplikasi robotika modular yang kompleks. Secara keseluruhan, sistem mempertahankan stabilitas operasional yang andal dengan penggunaan memori yang konsisten pada kisaran 1,7 – 1,8 GB, yang menegaskan bahwasa framework ROS 1 tetap sangat relevan dan andal untuk diimplementasikan sebagai middleware sistem otonom maupun robotik modular saat ini.

DAFTAR PUSTAKA

[1] M. Aljamal, S. Patel, and A. Mahmood, "Comprehensive Review of Robotics Operating System-Based Reinforcement Learning in Robotics," *Appl. Sci.*, vol. 15, no. 4, pp. 1–39, 2025, doi: 10.3390/app15041840.

[2] A. N. Morgan Quigley, Brian Gerkey,Ken Conley , Josh Faust, Tully Foote, Jeremy Leibs , Eric Berger, Rob Wheeler, "ROS: an open-source Robot Operating System," *ICRA Work. open source Softw.*, no. Figure 1, pp. 4754–4759, 2009, doi: 10.1109/IECON.2015.7392843.

[3] M. Albonico, M. B. Cannizza, and A. Wortmann, "Energy efficiency in ROS communication: a comparison across programming languages and workloads," *Front. Robot. AI*, vol. 12, no. April, pp. 1–28, 2025, doi: 10.3389/frobt.2025.1548250.

[4] B. Y. Song and H. Choi, "ROS Gateway: Enhancing ROS Availability across Multiple Network Environments," *Sensors*, vol. 24, no. 19, pp. 1–28, 2024, doi: 10.3390/s24196297.

[5] R. Sasaki, A. Takefusa, H. Nakada, and M. Oguchi, "Communication Performance of ROS and ROS 2-Based IoT Systems for Smart Home Applications," *IEICE Trans. Inf. Syst.*, vol. E108.D, no. 8, pp. 895–905, 2025, doi: 10.1587/transinf.2024DAP0005.

[6] Anis Koubaa, Ed., *ROS (Robot Operating System)*, vol. 30, no. 9. Springer, 2016. doi: 10.1007/978-3-319-26054-9.

[7] D. Portugal, R. P. Rocha, and J. P. Castilho, "Inquiring the robot operating system community on the state of adoption of the ROS 2 robotics middleware," *Int. J. Intell. Robot. Appl.*, vol. 9, no. 2, pp. 454–479, 2025, doi: 10.1007/s41315-024-00393-4.

[8] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the Performance of ROS2," pp. 0–9, 2016.

[9] P. C. Scheduling, Z. Wang, S. Liu, D. Ji, and W. Yi, "Improving Real-Time Performance of Micro-ROS with," 2024.

[10] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," *Proc. 13th Int. Conf. Embed. Software, EMSOFT 2016*, pp. 0–9, 2016, doi: 10.1145/2968478.2968502.

[11] C. Bedard, I. Lutkebohle, and M. Dagenais, "ros2_tracing: Multipurpose Low-Overhead Framework for Real-Time Tracing of ROS 2," *IEEE Robot. Autom. Lett.*, vol. 7, no. 3, pp. 6511–6518, 2022, doi: 10.1109/LRA.2022.3174346.

[12] Y. Ye, Z. Nie, X. Liu, F. Xie, Z. Li, and P. Li, "ROS2 Real-time Performance Optimization and Evaluation," *Chinese J. Mech. Eng. (English Ed.*, vol. 36, no. 1, 2023, doi: 10.1186/s10033-023-00976-5.

[13] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "OPC UA versus ROS, DDS, and MQTT: Performance evaluation of industry 4.0 protocols," *Proc. IEEE Int. Conf. Ind. Technol.*, vol. 2019-Febru, pp. 955–962, 2019, doi: 10.1109/ICIT.2019.8755050.

[14] J. Zhang, X. Yu, S. Ha, J. Peña Queralta, and T. Westerlund, "Comparison of Middlewares in Edge-to-Edge and Edge-to-Cloud Communication for Distributed ROS 2 Systems," *J. Intell. Robot. Syst. Theory Appl.*, vol. 110, no. 4, 2024, doi: 10.1007/s10846-024-02187-z.

[15] M. Albonico, M. Đorđević, E. Hamer, and I. Malavolta, "Software engineering research on the Robot Operating System: A systematic mapping study," *J. Syst. Softw.*, vol. 197, 2023, doi: 10.1016/j.jss.2022.111574.