

Abusing DDS Discovery: Denial-of-Service Attacks Against ROS 2

Jiafu Xu¹, Songran Liu^{1*}, Zilong Wang¹, Minghe Yu¹, Yue Tang¹, Yang Wang¹, Weiguang Pang², Wang Yi³

¹School of Computer Science and Engineering, Northeastern University, Shenyang, China

²Qilu University of Technology (Shandong Academy of Sciences), Jinan, China

³Department of Information Technology, Uppsala University, Uppsala, Sweden

xujf1@mails.neu.edu.cn, {liusongran, tangyue, wangyang2}@cse.neu.edu.cn,

wangzl10@mails.neu.edu.cn, yuminghe@mail.neu.edu.cn, pangwg@sdas.org, Wang.Yi@it.uu.se

Abstract—The Data Distribution Service (DDS) provides data-centric publish-subscribe messaging with a mandatory discovery protocol, enabling distributed applications to automatically locate and communicate with each other. ROS 2, the de facto middleware for robotic systems, adopts DDS as its communication backbone. In this paper, we demonstrate that the DDS discovery mechanism can be exploited to mount Denial-of-Service attacks against ROS 2 applications. By repeatedly triggering discovery traffic, an adversary can significantly inflate pipeline latency during runtime. We validate the attack on ROS 2 Humble with two widely used DDS implementations and a real UAV case study, confirming its effectiveness across different configurations.

Index Terms—DDS, ROS 2, Discovery Protocol, Denial-of-Service Attack

I. INTRODUCTION AND MOTIVATIONS

ROS 2 has become the de facto middleware for robotic systems, powering safety-critical applications such as autonomous vehicles [1] and UAV systems [2]. It relies on the Data Distribution Service (DDS) [3], an open middleware standard with several implementations such as Fast DDS [4] and Cyclone DDS [5], as its communication layer. Within this architecture, nodes publish and subscribe to topics in a shared domain (Fig. 1), and DDS automatically handles discovery so that communicating participants can locate each other. This automatic discovery is a double-edged sword. DDS mandates that every new node announce its presence to all existing participants [6], which is convenient for connectivity but also means each node creation generates domain-wide broadcast traffic [7].

Standard ROS 2 CLI utilities such as `ros2 topic list` work by briefly creating a temporary node to query the domain, then immediately destroying it [8]. Since each node creation triggers discovery broadcasts, these transient nodes inject traffic bursts that can interfere with other applications. In trials, we observed end-to-end latency jitter in victim pipelines when such commands execute. This raised a question: if a single diagnostic command perturbs timing, what happens when an attacker deliberately repeats this pattern? We term this exploitation **DoS attack**, where an adversary abuses discovery traffic to degrade the temporal behavior of victim applications.

To validate this attack, we instrumented a ROS 2-based UAV tracking system (Fig. 2) where compromised callbacks (executable units within ROS 2 nodes, denoted as `cb` in figures) continuously created and destroyed nodes at 10 Hz. Fig. 3 shows the impact on a victim pipeline’s latency as we activate

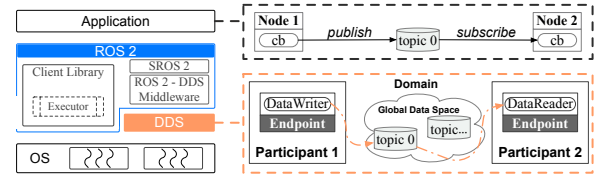


Fig. 1. ROS 2 and DDS system architecture.

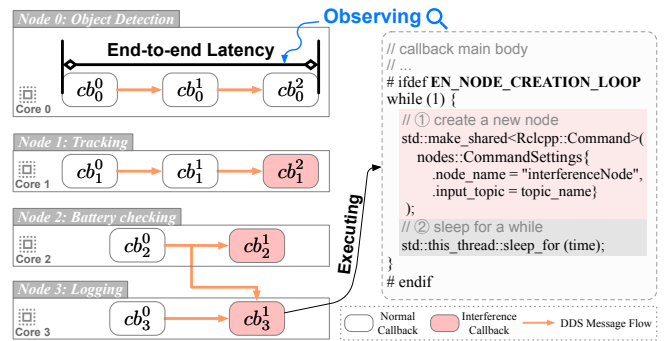


Fig. 2. Pipeline structure of a ROS 2-based UAV tracking system. Attack callbacks create ephemeral nodes in a tight loop.

one, two, and three concurrent attackers. End-to-end latency increased from 60 ms baseline to over 140 ms with three attackers. This degradation was observed in both Fast DDS and Cyclone DDS, confirming it is rooted in the DDS discovery mechanism rather than implementation-specific behavior.

This paper presents Denial-of-Service attacks against ROS 2, formalizes the threat model (including SROS 2 [9], the ROS 2’s optional security component), and validates the attack through benchmarks and a UAV case study.

II. ATTACK DESIGN

Threat Model. We consider ROS 2 applications where multiple nodes communicate via DDS within the same domain [10]. Nodes are scheduled by single-threaded executors [11], ROS 2’s callback scheduling components that execute callbacks in a non-preemptive, single-threaded manner. An adversary must first obtain a legitimate identity within the victim’s DDS domain. Without SROS 2, this is trivial. The adversary can join by enumerating well-known PDP port numbers [12], as ROS 2

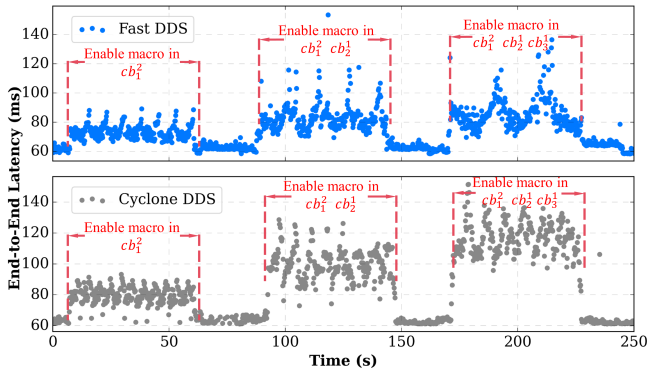


Fig. 3. Pipeline end-to-end latency under different levels of attack intensity.

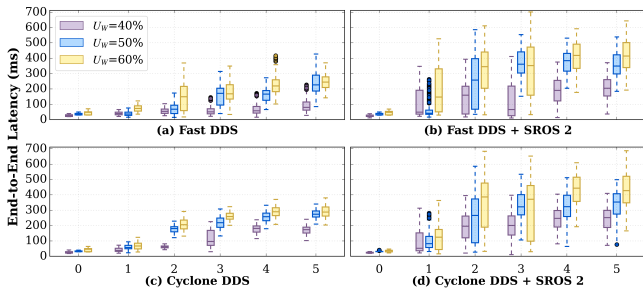


Fig. 4. Node 0's pipeline end-to-end latency under varying numbers of attacks, across DDS implementations and with/without SROS 2.

does not restrict new participants by default. With SROS 2 enabled, authentication blocks unauthorized outsiders [9]; however, adversaries can still gain legitimate access through known vulnerabilities such as permission file replacement, ROS 1 bridge exploitation, or context creation bypass [13], [14].

Attack Vector. Once admitted, the adversary can compromise existing callbacks or inject new ones. The attack logic is straightforward. Mimicking the CLI pattern, it rapidly creates and destroys temporary nodes. No privilege escalation, code injection, or spoofing is needed. Only the ability to instantiate ROS 2 nodes. In our implementation, each process creates 50 nodes at 10 Hz (empirically chosen). As ROS 2 supports distributed systems, attacks can originate locally or remotely.

III. EVALUATION

Platform. Experiments ran on NVIDIA Jetson NX [15] with Ubuntu 20.04 and ROS 2 Humble [16], using Fast (v2.6) and Cyclone DDS (v0.10). All cores were set to 1.7 GHz.

Setup. We used the UAV pipeline (Fig. 2) as our testbed, focusing on Node 0 as the critical task. Both experiments below measured Node 0's end-to-end latency: the first established attack boundary conditions across different configurations, while the second demonstrated real-world impact.

Scalability and Stress Evaluation. Three victim nodes executed on Core 0 at 40–60% utilization; Attackers deployed on Cores 1–5, with one attack per core. As shown in Fig. 4, latency increased with both the number of attackers and victim utilization. Taking Fast DDS at 60% utilization as an example, latency rose from 35 ms baseline to over 250 ms under

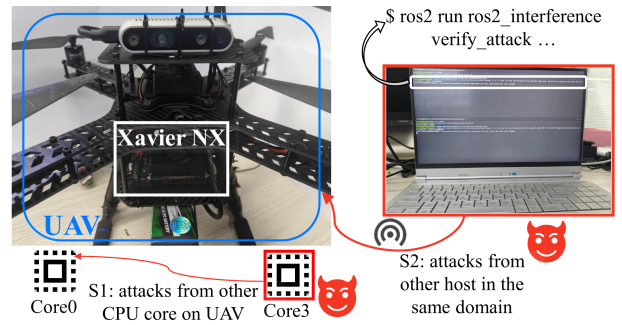


Fig. 5. UAV tracking system under S1 (same-host) vs. S2 (cross-host).

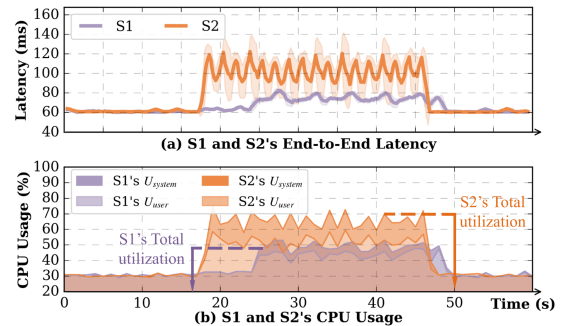


Fig. 6. Comparison of latency and CPU utilization under S1 and S2.

five attackers. This degradation was consistent across both DDS implementations. When SROS 2 was enabled, the attack impact was further amplified due to cryptographic processing overhead: at two attackers, 40% utilization with Cyclone DDS, latency reached 193 ms versus 63 ms without SROS 2.

Cross-Host Case Study. We evaluated two attack scenarios on the same UAV system with a single attack at 60% victim utilization (Fig. 5): same-host attacks (S1) and cross-host attacks (S2). As shown in Fig. 6, compared to S1, S2 incurred 15–20 ms higher average latency and increased CPU utilization due to the overhead of processing additional incoming network packets. Notably, even S1 caused significant latency degradation, confirming that discovery protocol processing, not just network transmission, is the primary source of interference. For control loops operating at 30–50 Hz (20–33 ms periods), this extra delay can cause commands to arrive a full cycle late, leading to delayed trajectory corrections and potentially catastrophic consequences in dynamic flight environments.

IV. CONCLUSIONS

We demonstrated that the mandatory DDS discovery mechanism can be weaponized into timing interference attacks against ROS 2 systems. By mimicking routine CLI behavior, an adversary can sustain discovery storms that significantly degrade pipeline latency, an effect observed across DDS implementations and amplified when SROS 2 is enabled.

V. ACKNOWLEDGEMENTS

This research was funded by the National Natural Science Foundation of China (NSFC) under Grants 62302083 and 62302087.

REFERENCES

- [1] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kit-sukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPs)*. IEEE, 2018, pp. 287–296.
- [2] S. Sandoval and P. Thulasiraman, "Cyber security assessment of the robot operating system 2 for aerial networks," in *SysCon '19*, 2019, pp. 1–8.
- [3] Object Management Group, "Data distribution service (dds), version 1.4," Object Management Group, OMG Specification formal/2015-04-10, April 2015, oMG Document Number: formal/2015-04-10. [Online]. Available: <https://www.omg.org/spec/DDS/1.4/PDF>
- [4] eProsima. (2025) Fast dds documentation. Accessed: 2025-04-29. [Online]. Available: https://fast-dds.docs.eprosima.com/en/stable/fastdds/dds_layer/domain/domainParticipant/domainParticipant.html
- [5] E. C. DDS. (2025) Cyclone dds documentation. Accessed: 2025-04-29. [Online]. Available: <https://cyclonedds.io/docs/cyclonedds-cxx/latest/>
- [6] OMG, "Dds interoperability wire protocol (ddsi-rtps) specification," Tech. Rep. formal/2022-04-01, 2022. [Online]. Available: <https://www.omg.org/spec/DDSI-RTPS>
- [7] W.-P. Nwadiugwu, D.-S. Kim, W. Ejaz, and A. Anpalagan, "Mad-dds: Memory-efficient automatic discovery data distribution service for large-scale distributed control network," *IET Communications*, vol. 17, no. 12, pp. 1432–1446, 2023.
- [8] Open Source Robotics Foundation, "ROS 2 CLI Node Strategy Implementation," 2023, accessed: 2025-12-12. [Online]. Available: <https://github.com/ros2/ros2cli/blob/humble/ros2cli/ros2cli/node/strategy.py>
- [9] V. Mayoral-Vilches, R. White, G. Caiazza, and M. Arguedas, "Sros2: Usable cyber security tools for ros 2," in *IROS '22*, 2022, pp. 11 253–11 259.
- [10] S. Macenski, T. Foote, B. P. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robotics*, vol. 7, no. 66, 2022. [Online]. Available: <https://doi.org/10.1126/scirobotics.abm6074>
- [11] H. Choi, Y. Xiang, and H. Kim, "Picas: New design of priority-driven chain-aware scheduling for ros2," in *RTAS '21*. IEEE, 2021, pp. 251–263.
- [12] Open Robotics. (2024) The ros_domain_id. ROS 2 Humble Documentation. [Online]. Available: <https://docs.ros.org/en/humble/Concepts/Intermediate/About-Domain-ID.html>
- [13] G. Deng, G. Xu, Y. Zhou, T. Zhang, and Y. Liu, "On the (in)security of secure ros2," in *CCS '22*, 2022, pp. 739–753.
- [14] L. Xia, X. Gao, and W. Shi, "Investigating security threats in multi-tenant ros 2 systems," 2025, unpublished. [Online]. Available: <https://weisongshi.org/papers/xia25-ICRA.pdf>
- [15] "Nvidia jetson xavier nx," <https://www.nvidia.cn/autonomous-machines/embedded-systems/jetson-xavier-nx/>, 2021, last accessed: June. 1, 2025.
- [16] ROS2 Community, "Ros2 humble," GitHub repository, 2024, accessed: [2025-06-01]. [Online]. Available: <https://github.com/ros2/ros2/tree/humble>