Assessment of Communication Protocols' Latency in Co-processing Robotic Systems

Eduardo Pereira*, Lucas Luza*, Nicolas Moura*, Luciano Ost†, Ney Calazans‡, Fernando Moraes* and Rafael Garibotti*
* School of Technology, Pontifical Catholic University of Rio Grande do Sul, Brazil – fernando.moraes@pucrs.br
† Wolfson School, Loughborough University, United Kingdom – l.ost@lboro.ac.uk
‡ PGMICRO, Federal University of Rio Grande do Sul, Brazil – ney.calazans@inf.ufrgs.br

***Abstract*—The evolution of robotic systems supporting co-processing can limit a multitude of uses and their effective deployment in real life. Although computing time still accounts for a non-negligible part of system performance, communication latency is increasingly relevant, as the growing use of heterogeneous co-processing in complex systems makes communication among elements a performance bottleneck. To undertake this coming challenge, this work evaluates the efficiency of a set of relevant communication protocols supported in ROS-based systems with co-processing. Results show UDP is the best choice for the transport layer, especially for high-performance systems. TCP-based communication can nonetheless play a role in low-end co-processing robotics.**

***Index Terms*—Communication Protocol, Robot Operation System (ROS), Co-processing Robotic Systems.**

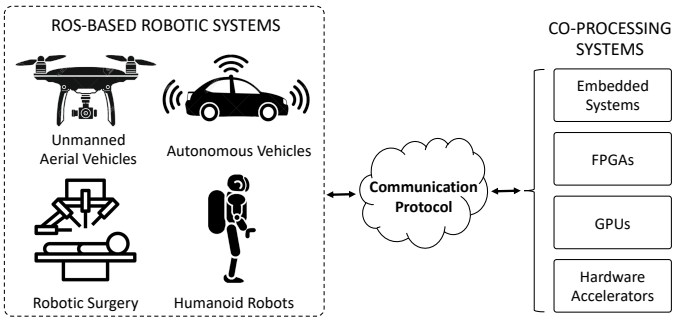


Fig. 1. Communication protocols – the bridge between advanced ROS-based and co-processing robotic systems.

I. Introduction

The last decades have witnessed an impressive industrial development and the adoption of robotic systems. These stood out in recent years in the emergence of a broad ecosystem of heterogeneous products and devices with increasing capabilities, ranging from simple remotely-controlled robots to complex robotic systems equipped with advanced computing capabilities and dedicated hardware to provide, e.g., high precision for robotic surgery and help control autonomous vehicles in unexpected situations or adverse conditions [1, 2].

Despite the mentioned and other recent achievements, the increase in robotic systems' capabilities brings challenges that were once secondary to become primary. Reducing the communication latency of co-processing robotic systems is one such challenge. This is due to the large increase in CPU core processing power, aided by co-processing systems such as graphical processing units (GPUs) and specialized hardware accelerators. This dramatic increase in processing capabilities can make the overall system inefficient as the communication speed does not scale as well as the computing speed [3, 4].

It is necessary to identify where communication bottlenecks lie to meet this challenge. This work assesses the main communication protocols supported in the robot operating system (ROS), targeting co-processing robotic systems. ROS is a *de-facto* standard middleware in robotic development for academia and industry [5]. Although communication performance using ROS has been investigated in the literature [3, 6, 7], to the best of the authors knowledge, no work adequately classifies the employed communication latency along various related works and demonstrates its impact on the use of co-processing robotic systems. Figure 1 illustrates the concept of co-processing ROS-based robotic systems.

The rest of this paper has four Sections. Section II presents related works that assess the communication latency in robotic environments. Next, Section III describes and justifies the communication protocols selection. Follows Section IV that explores the benefits and drawbacks of each communication protocol and relates these to different co-processing systems. Section V ends the work with conclusions and future work.

II. Related Work

Some previous works address the ROS communication latency problem. Maruyama *et al.* [6] proposed one of the first approaches to evaluate the performance of data transport, comparing ROS1 and ROS2 under various conditions. They claim that ROS1 is not suitable for real-time embedded systems. However, even though ROS running on Linux cannot provide real-time guarantees, this is not quite true. To support this new claim, Wei *et al.* [8] propose a real-time ROS architecture on multi-core processors for robot applications based on ROS1.

More recent works focus on the evaluation of communication latency targeting distributed real-time systems. Kronauer *et al.* [3] investigate the end-to-end latency of ROS2 for distributed systems with default settings and different data

This work was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES (Finance Code 001), CNPq (grants 309605/2020-2, 317087/2021-5, 311587/2022-4 and 407477/2022-5), and FAPERGS (grants 21/2551-0002047-4 and 22/2551-0000570-5).

distribution service (DDS) middlewares. They claim that DDS middlewares are suitable for real-time distributed embedded systems, due to their various transport configurations (e.g., deadline and fault-tolerance) and scalability. Results show that end-to-end latency is heavily dependent on the specific DDS middleware, and that ROS2 can lead to 50% more latency overhead than low-level DDS communications. This indicates that the communication protocol implementation can significantly impact the real-time requirements of ROS-based systems. In this regard, Wu *et al.* [7] question whether today's open-source middlewares' communication latency is sufficient to guarantee the requirements of autonomous vehicles.

On the one hand, reviewed works question the communication latency in ROS-based systems. On the other hand, several works suggest frameworks and architectures to provide better performance capabilities to such systems. Azumi *et al.* [9] develop a ROS-based framework for embedded manycore platforms based on a network-on-chip technology. Dehnavi *et al.* [10] propose a hardware-software architecture for ROS-based robotic development with a multi-core system. Lienen *et al.* [11] present a ROS-based framework that features multithreaded programming of hardware and software threads for reconfigurable computers, and which accelerates parts of robotic applications in hardware.

Previous works focus on either evaluating the communication latency only or on trying to improve the computational performance of robotic systems by changing the communication process. This work proposes an initial step towards integrating the two approaches, assessing several communication protocols available for ROS and relating them to the performance of co-processing robotic systems.

III. Features of Selected Protocols

This Section describes the communication protocols selected for assessment in this work. The first assumption is that the use of ROS-based robotic system is currently a major trend. The second assumption is that only communication schemes in widespread use in the literature are relevant. Thus, the selected five co-processing communication protocols are:

A. *ROS to ROS:* This protocol relates two devices with ROS, e.g., two robots communicating. It is based on the publish/subscribe model provided by ROS library functions, where a ROS Node (termed the *Publisher*) sends messages to a given topic (i.e., a data object stored in a shared space that any ROS application can access). Then, another ROS Node (termed the *Subscriber*) subscribes to the topic and starts receiving messages whenever a message is published on the topic. In this ROS-to-ROS communication, the TCP protocol is used by both sides;

B. *ROS to MATLAB:* This communication protocol relies on the MATLAB ROS toolbox to simulate a co-processing platform based on an external device running MATLAB. This ROS toolbox provides an interface that connects MATLAB and Simulink with ROS through a network composed of ROS nodes. Similar to ROS-to-ROS, this communication is also based on the TCP protocol;

C. *ROS to ROS Bridge:* A ROS Bridge is divided into protocol and implementation. This communication protocol allows non-ROS applications to exchange messages with ROS applications. We devise here three variants and two effectively distinct protocols. Regarding implementation, the best known is roslibpy (termed here C1), a library that allows using Python to interact with ROS through WebSockets. This library is still based on the TCP protocol and provides publishing, subscribing, and other essential ROS functionalities. Another protocol relies on the use of UDP instead. This second option comprises two approaches, depending on the chosen encoding scheme: we call here C2 the choice of using JSON encoding, while we term C3 the choice of using CBOR-RAW[1];

D. *ROS to Network:* This protocol is the only one that is not based on the publish/subscribe model, but rather in message exchange on the network using only UDP sockets. The strategy provides less inter-mediation, reducing conversions or tools that can add communication overheads. However, as the method does not use ROS network tools (mainly based on TCP), it does not guarantee any reliability for data exchanges;

E. *ROS to ORCA:* This protocol relies on a custom library that allows message exchange between ROS and a multiprocessor system-on-chip (MPSoC) environment [12]. This library relies on ROS nodes and on the publish/subscribe model, but uses the UDP network for communication.

Table I summarises the characteristics of the selected protocols and links them according to the literature. Note that both sides are classified by environment (i.e. higher layers of the OSI model [13]), the transport layer and tools needed to provide communication.

TABLE I
Summary of the communication protocols' characteristics.

Communic.	**Robotic System Side**			**Co-processing Side**		
Protocol	**Env.**	**Transp.**	**Tool(s)**	**Env.**	**Transp.**	**Tool(s)**
A [14]	ROS	TCP	ROS, Python	ROS	TCP	ROS, Python
B [15]–[17]	ROS	TCP	ROS, Python	MATLAB	TCP	ROS Toolbox
C1 [18]	ROS	TCP	ROS Bridge, Python	Linux	TCP	Python
C2/C3 [18]	ROS	TCP	ROS Bridge, Python	Linux	TCP/UDP	Python
D [19]	ROS	UDP	ROS, Python	Linux	UDP	Python
E [12, 20]	ROS	TCP	ROS, Python	Linux	TCP/UDP	Python, Custom Lib.

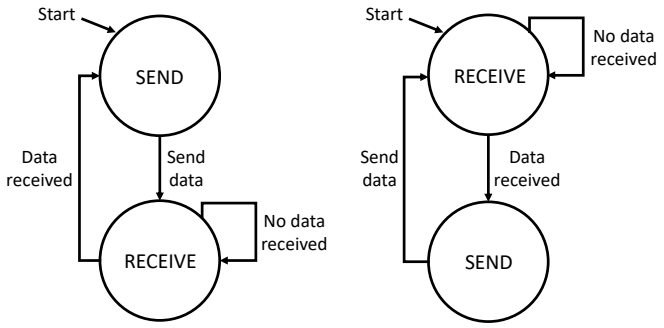
[1]Documentation available at https://cbor.io.

IV. Experiments, Results and Discussion

A. Experimental Setup

To properly evaluate communication protocol latency, all results were extracted from a single computer with the following configuration: (*i*) Intel Core I7 9700KF, an 8-core CPU with 12 MB cache; (*ii*) CPU cores run at 3.6 GHz (4.9 GHz Max Turbo); (*iii*) 16 GB 2666 MHz DDR4 RAM; and (*iv*) Ubuntu 20.04.5 LTS OS (Focal Fossa).

The robotic system side is configured with ROS Noetic and Python 3.8. To simulate the robotic side, roscore must first be instantiated for ROS nodes to communicate. Then, scripts are launched to start the communication process. Figure 2 illustrates the communication process between the robotic system and the co-processing sides. Note that regardless of whether or not the protocol is based on the publish/subscribe model, it has the same behaviour. First, the robotic side starts, by sending a data packet and waiting until the sent message returns. On the other side, co-processing starts in a waiting loop consuming data and responding with the same message.



(a) Robotic system side. (b) Co-processing side.

Fig. 2. Communication process state machines.

B. Communication Protocols Latency

The protocol evaluation experiments showed that almost all protocols relying on the ROS publish/subscribe model with TCP display the same latency and behaviour. Figure 3 shows the same latency results for protocols A and C1/C2/C3 implementations, with a data size range from 8B to 1MB. Note that the latency behaves as a stairway, where large data size intervals display almost the same communication time. For example, a round trip communication with data between 8B and 512B takes about 10ms, while from 1kB to 256kB, takes always around 42ms. In addition, Figure 3 shows an intriguing characteristic. When the data size increases from 512B to 1kB, latency jumps from 10.5ms to 41.62ms, i.e., almost 4× more. Therefore, it is clear that sending two 512B messages is faster than sending one 1kB one. However, this approach is not applicable for other message sizes, say 2kB.

Figure 4 shows the latency results for communication from ROS to MATLAB, where latency behaviour can be divided into two classes. First, data sizes smaller than 4kB take between 18 and 20 ms. Second, data sizes equal to or greater than 8kB takes longer than 730 ms. Furthermore, an extra measurement using a 6kB message was also tried, taking

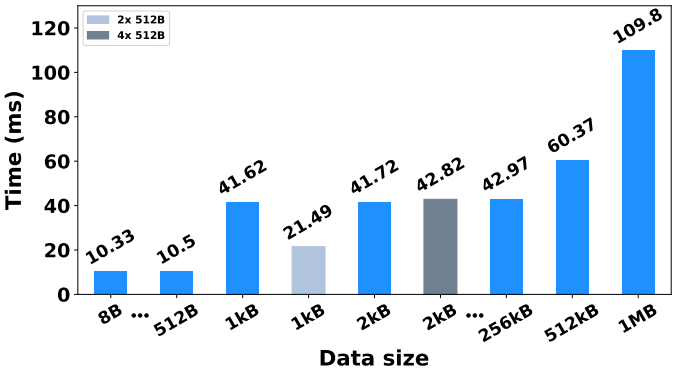


Fig. 3. Latency results for protocols based on the publish/subscribe model with TCP, which have the same behaviour and include A, C1, C2 and C3.

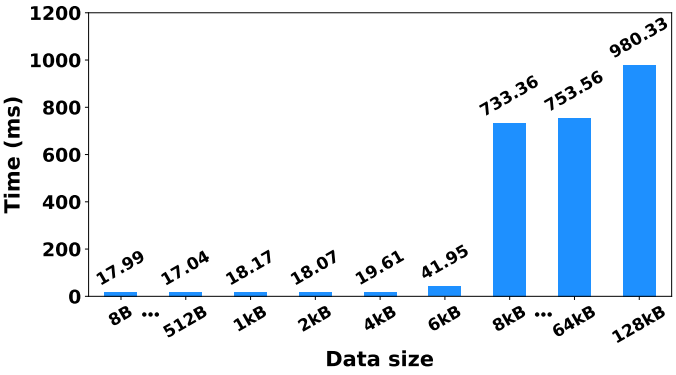


Fig. 4. Latency results for communication from ROS to MATLAB (B).

twice as long as to send a 4kB message. Therefore, it is recommended to break any message larger than 4kB into 4kB blocks.

Figure 5 shows the latency results for a custom library case study. Although this communication protocol relies on the same ROS publish/subscribe model with TCP used in the A and C1/C2/C3 protocols, the overhead imposed by the extra communication layer to exchange messages between processors takes its toll (compare the results with Figure 3). Figure 5 shows that even for small data sizes such as 8B, the overhead built on top of the ROS node makes it unfeasible for co-processing solutions to work with such an approach.

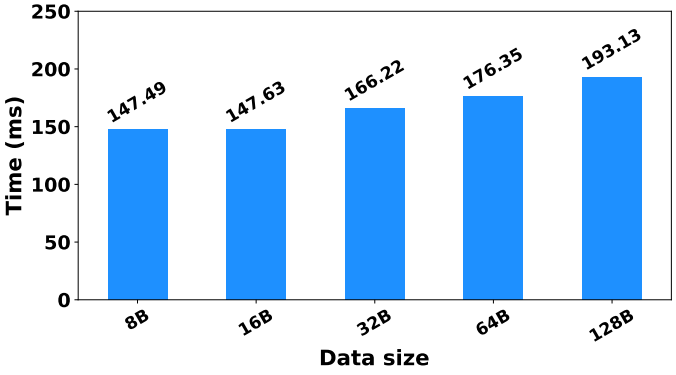


Fig. 5. Latency results for communication from ROS to ORCA (E).

Next, Figure 6 shows the latency results for the only communication protocol that uses the UDP protocol on both sides. First, it is striking that the latency has dropped by about 3 orders of magnitude, going from the ms range to the μs range. Also, the overall range is very small, going from 7.06 to 13.58μs. The UDP transport layer for ROS is based on the standard UDP datagram packet, i.e. the maximum data size is slightly less than 64kB, because of the header. Therefore, larger messages must be broken.

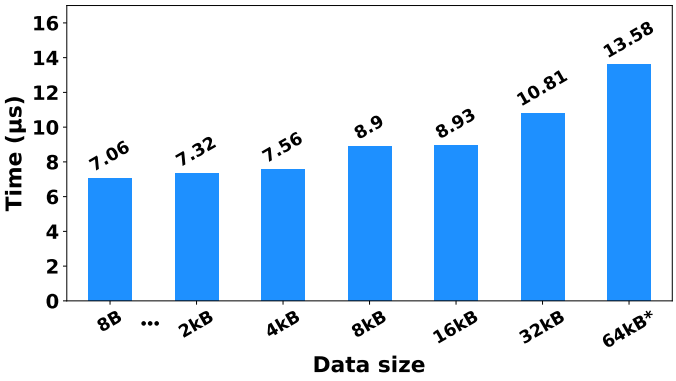


Fig. 6. Communication time for the UDP-based protocol (D). Note that due to UDP header, the maximum data size is slightly less than 64kB (64kB*).

Figures 3 to 6 demonstrate that UDP must be the choice for co-processing-based systems. However, it is worth mentioning that this protocol does not guarantee any data communication. In contrast, the TCP protocol ensures reliable message exchange at a high cost in communication latency.

C. Application Throughput Analysis Case Study

To understand how latency impacts co-processed robotic systems, a case study was created based on the AES encryption algorithm (AES-128 with ECB mode) [21]. This case study sends a 128-bit plaintext and a 128-bit key to be encrypted by the co-processing side, which must return the 128-bit ciphertext. Table II shows the AES execution time on various devices that can be used for co-processed robotic systems, such as embedded systems, FPGAs and GPUs. Note that the AES-128 compute time ranges from 2.52μs to 55.69ms, a throughput ranging between 0.33kB/s and 6.35MB/s.

Relating Section IV-B with the results shown in Table II, reveals a possibility of using the TCP protocol for embedded systems based on microprocessors (i.e. Arm Cortex-M devices). This is because the TCP latency is 10 ms, while 3 out of 4 of the embedded system boards display longer processing times. However, for high-performance co-processing systems, such as hardware accelerators and GPUs, the communication delay of the TCP protocol is about 1000$\times$ greater than the AES computation time. This leads to the conclusion that UDP is the preferable communication protocol since it provides the higher performance, due to both computation and communication time, which here would be in the μs range.

TABLE II
THROUGHPUT SUMMARY OF CO-PROCESSING RUNNING AES-128.

Co-processing Devices	Frequency (MHz)	Execution Time (ms)	Throughput (MB/s)
Arm Cortex-M3 board	120	44.07	0.00036
Arm Cortex-M4 board	80	55.69	0.00029
Arm Cortex-M7 board	400	7.85	0.00204
Arm Cortex-M33 board	110	48.93	0.00033
Raspberry Pi 4 board	1,500	0.01241	1.29
Intel Core I7 9700KF (CPU standalone)	3,600	0.00252	6.35
FPGA MPS2+ [22] (hardware accelerator)	25	0.00416	3.85
NVIDIA Jetson TX2 (GPU)	854	0.01034	1.55

V. CONCLUSIONS AND FUTURE WORK

This work showed the advantages of using UDP as basis of communication protocols for high-end co-processing robotic systems. Meanwhile, results also show that as TCP latency is about 10ms for small data sizes, which justifies its use only in combination with low-end co-processing systems. As experiments here used a variety of platforms (boards, robotic systems and computers), results are expected to be trustworthy. However, a more detailed analysis of causes of the "stairway" behaviour of protocols can be instrumental in devising a more thorough strategy to determine the message's right size in robotics systems. Aspects that play a part in this can be: (*i*) buffer sizes; (*ii*) max protocol message size; and (*iii*) message padding strategies. This is an interesting future work.

REFERENCES

[1] C.-K. Chui, C.-B. Chng, and D. P. Lau, "Parallel Processing for Object Oriented Robotic Simulation of Tracheal-Oesophageal Puncture," in *SII*, 2011, pp. 144–149.

[2] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems," in *ICCPS*, 2018, pp. 287–296.

[3] T. Kronauer, J. Pohlmann, M. Matthé, T. Smejkal, and G. Fettweis, "Latency Analysis of ROS2 Multi-Node Systems," in *MFI*, 2021, pp. 1–7.

[4] R. Garibotti, L. Ost, A. Butko, R. Reis, A. Gamatié, and G. Sassatelli, "Exploiting memory allocations in clusterised many-core architectures," *IET Computers & Digital Techniques*, vol. 13, no. 4, pp. 302–311, 2019.

[5] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA*, 2019, pp. 1–6.

[6] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *EMSOFT*, 2016, pp. 1–10.

[7] T. Wu, B. Wu, S. Wang, L. Liu, S. Liu, Y. Bao, and W. Shi, "Oops! It's Too Late. Your Autonomous Driving System Needs a Faster Middleware," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7301–7308, 2021.

[8] H. Wei, Z. Shao, Z. Huang, R. Chen, Y. Guan, J. Tan, and Z. Shao, "RT-ROS: A real-time ROS architecture on multi-core processors," *Future Generation Computer Systems*, vol. 56, pp. 171–178, 2016.

[9] T. Azumi, Y. Maruyama, and S. Kato, "ROS-lite: ROS Framework for NoC-Based Embedded Many-Core Platform," in *IROS*, 2020, pp. 4375–4382.

[10] S. Dehnavi, M. Koedam, A. Nelson, D. Goswami, and K. Goossens, "CompROS: A composable ROS2 based architecture for real-time embedded robotic development," in *IROS*, 2021, pp. 6449–6455.

[11] C. Lienen, M. Platzner, and B. Rinner, "ReconROS: Flexible Hardware Acceleration for ROS2 Applications," in *ICFPT*, 2020, pp. 268–276.
[12] P. H. Vancin, A. R. P. Domingues, M. Paravisi, S. F. Johann, N. L. V. Calazans, and A. M. Amory, "Towards an Integrated Software Development Environment for Robotic Applications in MPSoCs with Support for Energy Estimations," in *ISCAS*, 2020, pp. 1–5.
[13] J. Day and H. Zimmermann, "The OSI reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.
[14] R. Amini, R. Sulaiman, and A. H. A. R. Kurais, "CryptoROS: A Secure Communication Architecture for ROS-Based Applications," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, pp. 189–194, 2018.
[15] K. Kumar, S. I. Azid, A. Fagiolini, and M. Cirrincione, "Erle-copter Simulation using ROS and Gazebo," in *MELECON*, 2020, pp. 259–263.
[16] M. Longinos-Garrido, N. A. Tolentino-Medrano, J. F. Guerrero-Castellanos, J. Castañeda-Camacho, and R. Álvarez González, "Event-triggered coordination of omnidirectional robots over Gazebo," in *EBCCSP*, 2020, pp. 1–8.
[17] N. Mahdian, S.-H. Attarzadeh-Niaki, and A. Salimi-Badr, "A Systematic Embedded Software Design Flow for Robotic Applications," in *ICCKE*, 2021, pp. 217–222.
[18] K. S. Sikand, L. Zartman, S. Rabiee, and J. Biswas, "Robofleet: Open Source Communication and Management for Fleets of Autonomous Robots," in *IROS*, 2021, pp. 406–412.
[19] S. Shin, D. Yoon, H. Song, B. Kim, and J. Han, "Communication system of a segmented rescue robot utilizing socket programming and ROS," in *URAI*, 2017, pp. 565–569.
[20] A. R. P. Domingues, D. A. Jurak, S. J. Filho, and A. De Morais Amory, "Integrating an MPSoC to a Robotics Environment," in *LARS*, 2019, pp. 204–209.
[21] NIST, "Specification for the Advanced Encryption Standard (AES)," Federal Information Processing Standard (FIPS) Publication 197, 2001.
[22] V. Bandeira, J. Sampford, R. Garibotti, M. G. Trindade, R. P. Bastos, R. Reis, and L. Ost, "Impact of radiation-induced soft error on embedded cryptography algorithms," *Microelectronics Reliability*, vol. 126, p. 114349, 2021.