Analysis Of Robotic Platforms: Data Transfer Performance Evaluation

Andrei Gavrilov[1], Marlen Bergaliyev[2], Sergey Tinyakov[3], Kirill Krinkin[4]
[1,2,3,4]Saint Petersburg Electrotechnical University "LETI"
St. Petersburg, Russia
gavrilov.andrew1999@yandex.ru[1], marlen.ber@mail.ru[2], sergey.tinaykov2001@gmail.com[3], kirill@krinkin.com[4]

***Abstract*—Every robotic system has to have high performance, which in the first place depends on data transfer performance. Interaction between processes plays a big role and can introduce very significant delays in the operation of the system. Therefore, the choice of the platform on which the system will be built is very important. This paper shows the characteristics of data transmission for these platforms and helps to make a choice for a specific development and task. This paper considers the well-known robotic platforms, developed criteria for evaluating characteristics, and tests for measuring them. For each criterion was made a comparative analysis of the platforms. The conclusion is made about the applicability of a specific platform to solving various problems and building various systems.**

I. INTRODUCTION

At the heart of any robotic project or autonomous driving system is a framework that manages the resources of the software components of the system and transfers data between these components. Such frameworks can be used both for prototyping the system and for the release version, so an important problem is the speed of data transfer in a system built on their basis. Since the speed of data transfer can greatly affect the operation of the system, you should seriously consider choosing a platform for building your system.

The purpose of this paper is to evaluate the data transmission characteristics of such management frameworks. Such characteristics as delay at different lengths of the transmitted message, jitter, and round trip time are evaluated.

The object of research in this paper is Nvidia Isaac, Apollo Cyber RT, ROS2. ROS2 and Apollo Cyber RT are open source and already used in different projects.

The paper describes the methods and scenarios in which certain transmission characteristics were studied, and a description of these scenarios is presented in section II. The described scenarios should help to understand how well the framework performs in a particular situation. The following is a brief description of the platforms under consideration in section III to understand the features of each framework that may affect latency. Review and analysis of the results obtained in section IV. Based on the data from all the sections, it is possible to draw conclusions about the applicability of the framework in various tasks, depending on their features and data transfer characteristics.

RELATED WORK

One of the well-developed autonomous driving platforms is Apollo. A driving prediction architecture for different scenarios and for different learning models was developed based on Apollo [1]. A path planning module with resolution-complete collision avoidance capability was developed for Apollo [2]. Known bugs were analysed for Apollo and Autoware [3]. For supporting cross-vehicle applications dynamic modeling procedure also based on Apollo[4]. Safety mechanisms for autonomous vehicles were designed using Apollo [5]. The performance analysis of Apollo and Autoware was conducted in [6].

ROS is a widespread, easy to use framework for developing robotics platforms and it has a wide community. Also this framework can be used as a platform for autonomous vehicles. Not the whole platform may be developed by ROS, but. ROS can be used only for some parts of it, for example, car vision, navigation[7], stereo vision by 2D LiDAR and RGB-D Camera[8], or for the whole platform[9]. It can be implemented, for example, for FPGA board[10]. The design of a mobile robot based on ROS was described in [11].

There is a lot of middleware that is used in distributed systems that are widely used in IoT [12-14]. A comparison of ROS, Apollo and the recently developed Robust-Z was in [15]. Requirements and tests are needed to evaluate systems for real-time operations. The description of this was in [16]. Also there were descriptions of modularization and real-time architecture[17],[18].

II. METHODOLOGY

Test scenarios were developed to obtain the desired data transmission characteristics. Each test scene is set up to investigate certain data transmission characteristics. The most used type of data transmission in such systems is IPC, because most of the main software modules that analyze the received data run on the main and most powerful computer. It is on this computer that the largest amounts of data are transmitted, and IPC is the fastest way to transmit this data. In all cases, the IPC bus data transfer model was established, with the exception of ROS2, since it does not provide the ability to explicitly specify the data transfer method. Parameters that increase the reliability of data transmission or guarantee delivery have also been set.

The basic concept of all test scenarios is the interaction of several programs (nodes) with different message frequency and different message length. Some test scenarios use mechanisms such as process prioritization and CPU affinity setting using the CPUSET mechanism. This allows you to avoid unexpected interruptions of the program and get the most accurate data. Also, to get the most objective data, delays are calculated only when reading the message data, as the timestamp of sending is stored in the message itself. Thus, the resulting delays include such factors as reading and writing data, waiting for the

message queue, and delays in the functioning of the framework itself. Next, we will consider the main test scenarios in which the characteristics of data transmission were investigated.

A) Test: Queue processing

The test is aimed at investigating the processing of the message queue. In the test, the first node sends messages to the other node without any interval. Thus, the queue accumulates.

The test is performed with a message size of 50 and 60 000 bytes. The total number of messages for each subtest is 5000. The priority of the processes is set to 99. The first node is bound to CPU_0, the second node to CPU_1. So, each process is separate and works without interruptions during queue processing, which allows you to get the most accurate data about the studied characteristic.

B) Test: Changing sizes and frequencies of messages

The test is aimed at investigating the total latency on the message size and sending frequency. The test also shows the reaching peak of the throughput at different message sizes and sending frequencies. In the test, the first node sends messages to the other on a given interval. The message size increases from 128 bytes to 2 MB in 256 KB increments every 100 messages. The total number of messages is 800.

The test is performed with frequencies from 20 to 1000 messages per second. Nodes are not bound to cores. The priority of processes is 99.

C) Test: Ping-pong of the minimum message size

The test is aimed at investigating total latency, jitter and RTT (Round Trip Time) of each message of the minimum size. The test also shows the number of copies between the user space and the kernel space. In this scenario the ping-pong model is used: one node sends a message to the other, and then both exchange messages only after receiving a message from the other node. The test stops when the set number of transmitted messages is reached.

The total number of messages for each node is 10000. Size of each message is 10 bytes. Nodes are not bound to cores, and process priorities are not set.

D) Test: Ping-pong with different frequencies, message sizes and number of pairs

The test is aimed at investigating the dependence of the total latency on the message size at different message sending frequencies and on the number of process pairs. In this scenario also the ping-pong model is used, but the node sending the first message, also called the first node, does not expect to receive a message from the other node and sends new messages on a given frequency. The other node works as in the previous test. If the first node difference between the number of the sent message and the last received one is greater than the watermark, no message will be sent. The test stops when the set number of transmitted messages is reached.

The watermark equals 50 for the test. Nodes are not bound to cores and process priorities are not set.

The test consists of 2 types of subtests: 1) one pair exchanges messages on a given frequency; 2) on frequency 400 messages per second several pairs exchange messages. The first type of subtests is performed with frequencies from 20 to 1000 messages per second. Total number of messages is 2400, message size increases from 128 bytes to 2 MB in 256 KB increments every 300 messages. The second type of subtests is performed with the following number of pairs: 1, 2, 3. Total number of messages is 1200, message size increases from 128 bytes to 2 MB in 512 KB increments every 300 messages.

III. Overview of frameworks

A) Nvidia Isaac Engine

The Isaac Engine is a software framework developed by Nvidia to create robotics applications. It provides data processing and deep learning for intelligent robots.

In Isaac Engine you build applications by creating small components, which pass messages between each other. It uses a graph that tries to avoid memory copies on the host-device. Also graphs help to break down a complex task into small objects. Isaac Engine uses CUDA buffer objects for messages to increase performance.

Isaac Engine comes with a visualization framework that allows to easily create plots, drawings, 3D scenes and other. There is also Isaac WebSight, a web application for inspecting and debugging applications. Isaac Engine has a Python API that allows creating applications on Python without losing for functional or performance.

Isaac Engine fully supports NVIDIA GPUs and CUDA, TensorRT, NPP and other frameworks that allow you to build the robotics application.

B) Apollo Cyber RT

Apollo Cyber RT is an open source, runtime framework that was created specially for autonomous driving. It uses a centralized model.

The base of architecture is a set of components, which generate data outputs from defined data inputs. Apollo Cyber RT uses a DAG (Directed Acyclic Graph) dependency graph to extract components dependencies and link them. At runtime, framework takes these linked components and fused data from sensors to create lightweight user-level tasks. Each task is scheduled according to priorities and resource availability to optimize executing.

Apollo Cyber RT has a configurable and flexible user level scheduler, a set of development tools, a large sensor drivers and minimum dependencies.

The technology behind Apollo Cyber RT provides optimized data transmission and processing out of the box. This framework comes with a well-defined task interface and efficient data fusion, allowing developers to create solutions on top of it.

C) ROS2

ROS2 is a large-scale framework for prototyping a robotic platform. Also, a large number of ready-made modules have been written for this framework, which can help in designing your own platform. At the heart of data transmission, ROS2 uses Data Distribution Services. ROS2 is a modified version of the ROS that provides an interface for implementing any middleware for data transfer. Also, ROS2 is better suited for real-time systems than ROS.

ROS2 has many utilities, including utilities for visualizing the transmitted data and for simulations. Each module is independent and can only depend on the received data. In general, ROS2 provides a complete set of tools for easy and fast prototyping of your own system. It has a large community, which helps ROS to constantly develop and update.

IV. Evaluation

A) Test: Queue processing

Fig. 1-3 show the test results for this scenario.

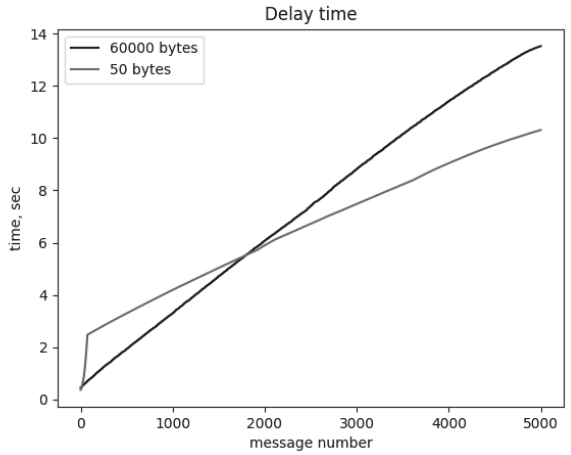

Fig. 1. ROS2 delay time

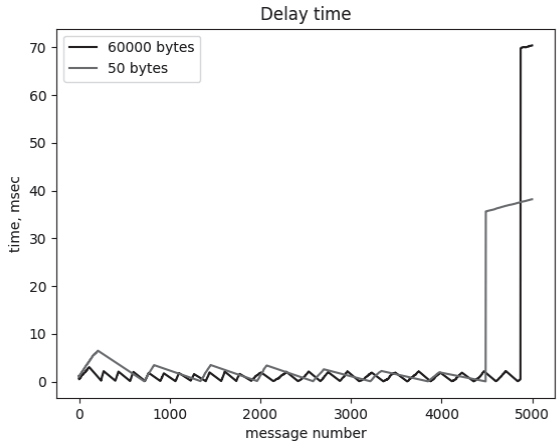

Fig. 2. Apollo delay time

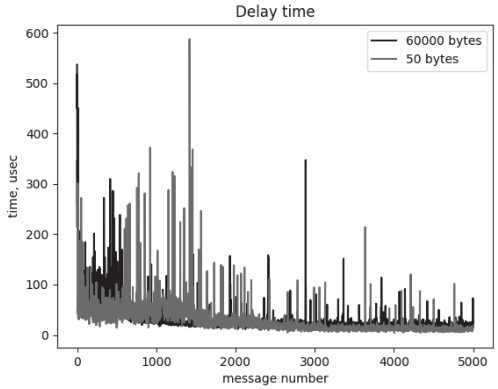

Fig. 3. Isaac delay time

There we can see that ROS2 cannot deal with a lot of messages sent in a small time period, so delivering and processing messages takes too much time. Apollo is more stable than ROS2, but delay fluctuates during the whole test. There is also an increase in the delay at the end, which is associated with the features of testing. Isaac shows excellent results: a very small delay and no jumps or drops. As you can see from Fig. 3, graphs overlap each other. This means that there is no difference in delay for messages of 50 and 60000 bytes.

B) Test: Changing sizes and frequencies of messages

In Fig. 4-5, you can see the effect of changing the sending frequency on delays in ROS2. As the frequency increases, the delays continue to increase, but the queue is formed with a smaller message size.

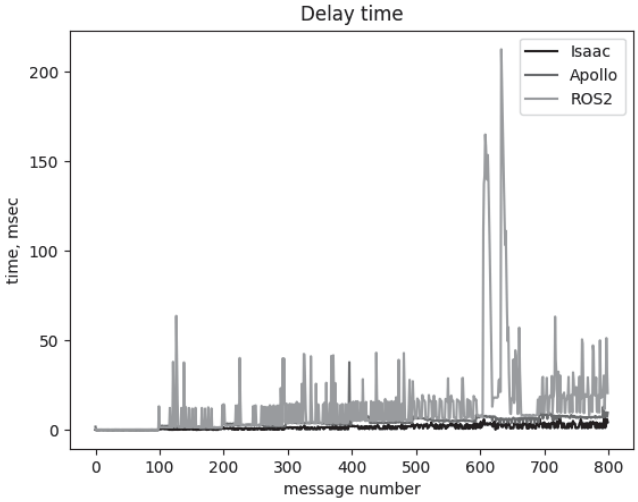

Fig. 4. Delay time with 40 msgs/sec

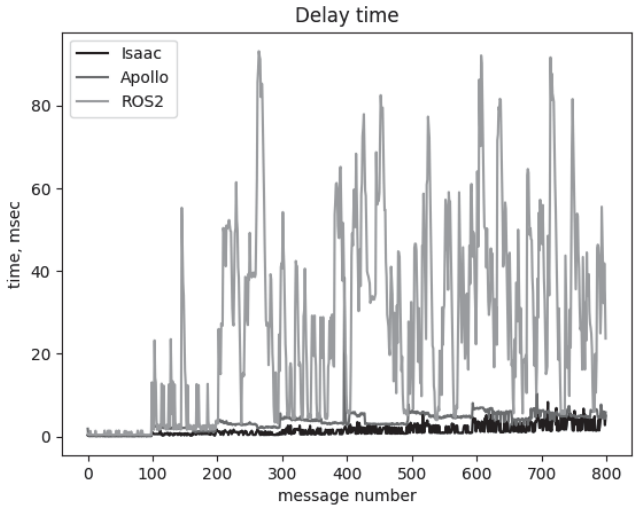

Fig. 5. Delay time in with 100 msgs/sec

In Fig. 6, you can see that with large messages at a frequency of 200 messages per second, the delays in ROS2 increased very fast and a queue appeared. As the frequency increases, the queue begins to grow with a smaller message size, so we don't present results with a greater frequency.

If the message size is over 1 MB, Apollo and Isaac have the same delay, otherwise Apollo has greater delay as we can see in the Fig. 7. Also Apollo's delay has the jumps at every message increment. Isaac has a small delay and no growth or jumps with frequency increasing.

Apollo has systematically large delays on 400 messages, where the message length increases to 256 KB. This phenomenon is repeated every time and at a different frequency of sending. This may be due to reserving memory for sending large messages and can't be related to the testing system.

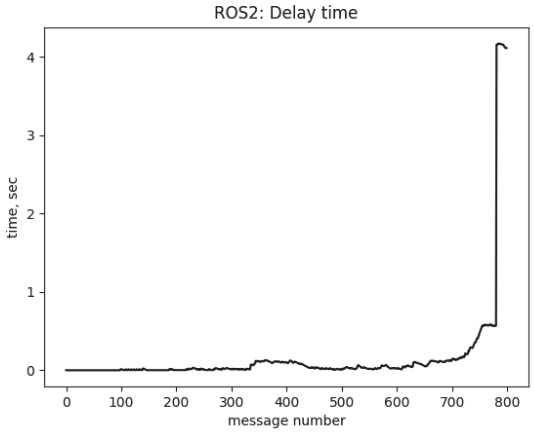

Fig. 6. ROS2's delay time with 200 msgs/sec

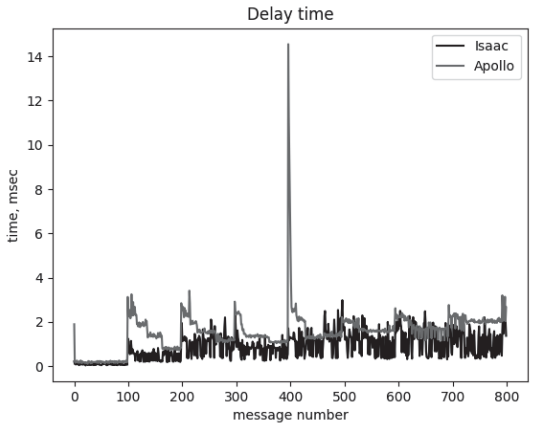

Fig. 7. Isaac's and Apollo's delay time with 1000 msgs/sec

The following conclusion about stability can be done from the graphs. ROS2 has a very unstable delay which depends on the message sending frequency. ROS2 has a fast growth of delay at message sending frequency over 200 messages per second. Growth of delay does not increase quickly on message size below 1 MB, but then the delay starts to increase faster. Apollo and Isaac, unlike ROS2, have a stable delay that does not depend on the frequency of sending messages.

C) Test: Ping-pong of the minimum message size

In this test, we are looking at RTT, latency and jitter for very small messages. So, Fig. 8-12 shows us these characteristics.

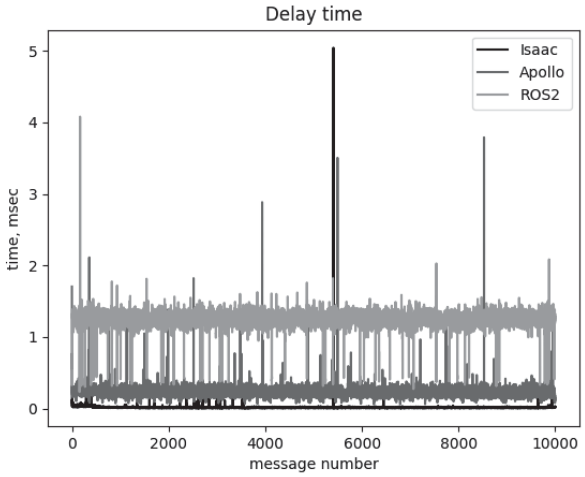

Fig. 8. Isaac's, Apollo's and ROS2's delay time

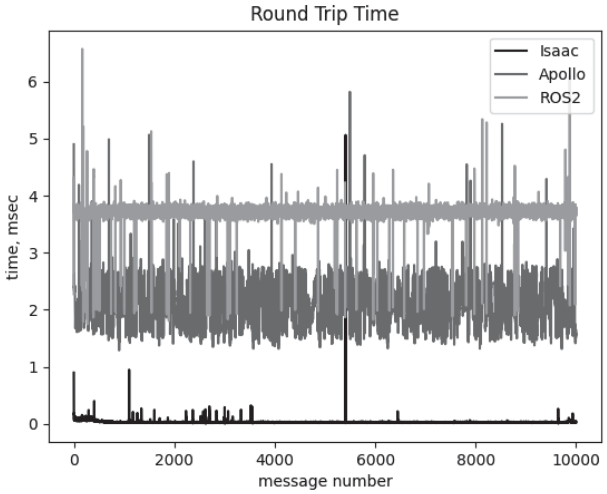

Fig. 9. Isaac's, Apollo's and ROS2's RTT

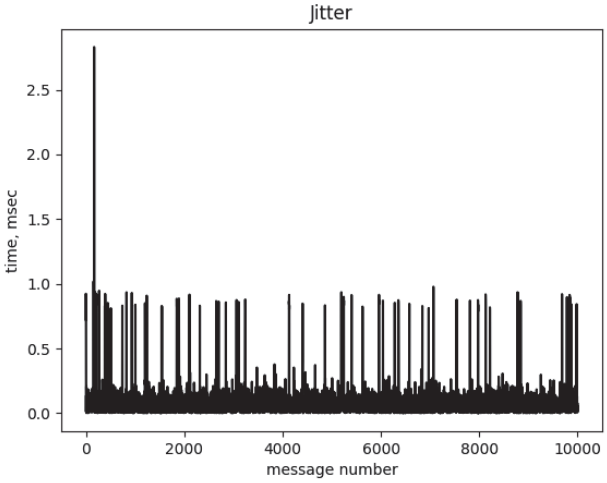

Fig. 10. ROS2's jitter

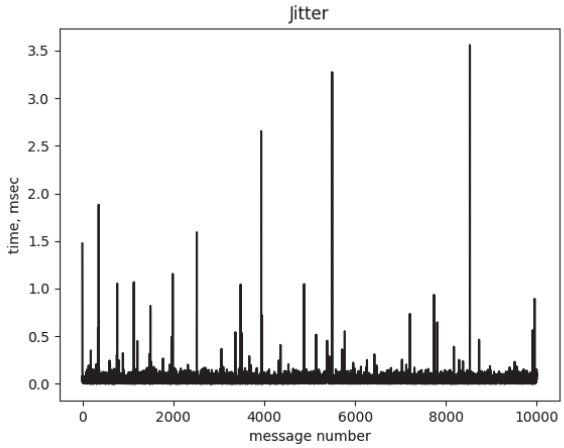

Fig. 11. Apollo's jitter

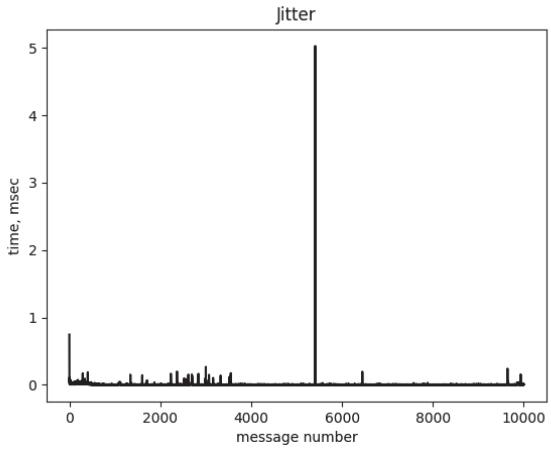

Fig. 12. Isaac's jitter

In this test queue not accumulates, so no one has the jumps and increases in delay or RTT, except for isolated cases that may be related to the testing system. Isaac has the best delay time and RTT. Apollo and ROS2 are far away from it, but Apollo has better results than ROS2. The Table I shows the median values for jitter.

TABLE I. MEDIAN JITTER VALUE FOR EACH FRAMEWORK

Framework	Median jitter value, μsec
ROS2	78.9803
Apollo	49.217
Nvidia Isaac	5.871

ROS2 has the largest jitter – up to 1 millisecond in some cases. The situation is better with Apollo – jitter is up to 0.25 millisecond, but there are regular jumps up to 1 milliseconds. Isaac has the smallest jitter — less than 0.1 millisecond.

D) Test: Ping-pong with different frequencies, message sizes and number of pairs

In previous tests, ROS2 has shown the worst results, so at this point it is clear that ROS2 is an outsider. This test is similar to test *B*, but there is a bidirectional node communication, unlike test *B*, so the results will be worse. For this reason, ROS2 has not been tested.

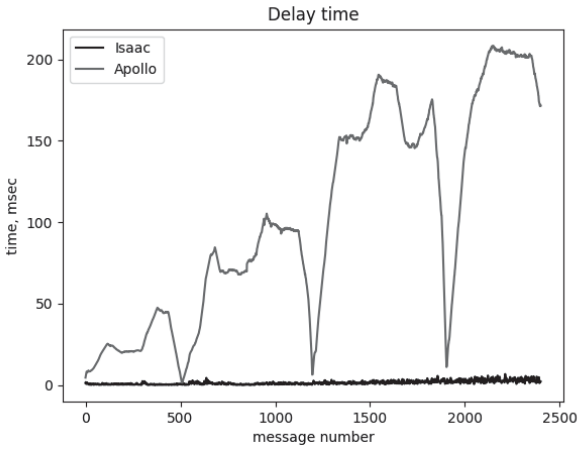

Fig. 13. Isaac, Apollo delay time with 1000 message per second frequency

Since in this test, in comparison with test *B*, we receive a response to each message, the resource consumption increases and the amount of data transmitted also increases by 2 times. Based on this, Apollo has significant differences in comparison with test *B*, which can be seen in Fig. 7 and 13.

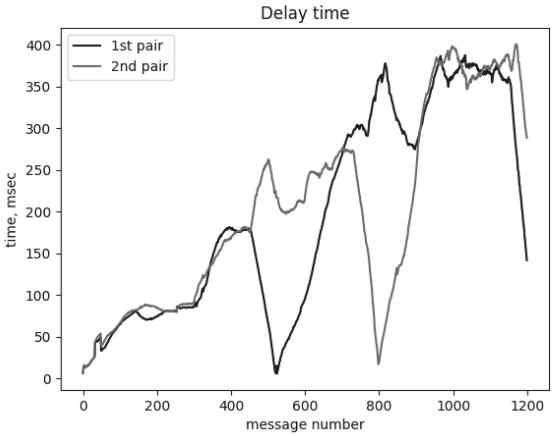

Fig. 14. Apollo's delay time with 2 pairs

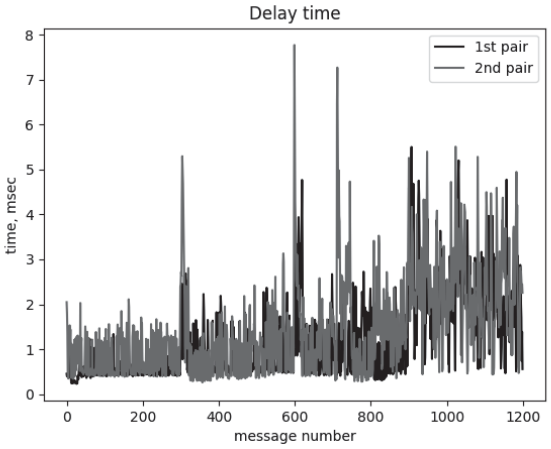

Fig. 15. Isaac's delay time with 2 pairs

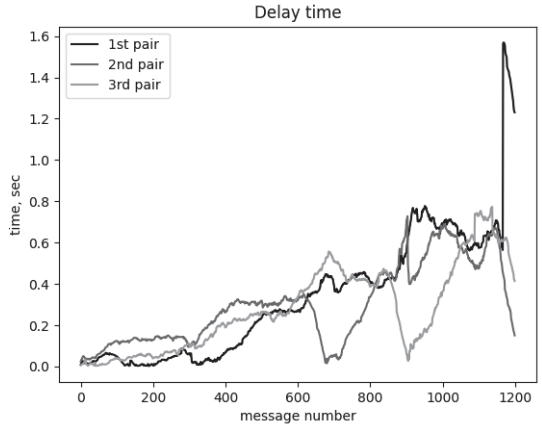

Fig. 16. Apollo's delay time with 3 pairs

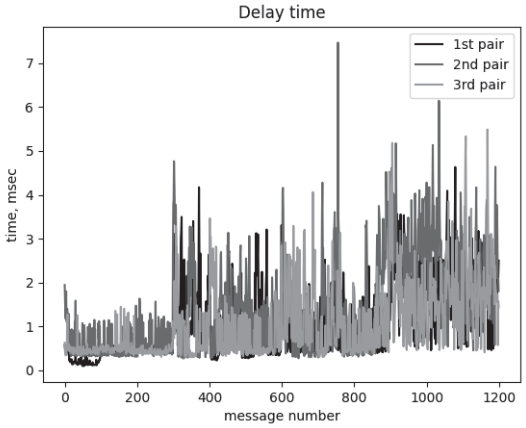

Fig. 17. Isaac's delay time with 3 pairs

As you can see from Fig. 14-17, the delay of Apollo and Isaac increases with each message, but Apollo has a larger delay than Isaac. Isaac's delay is in range of 10 milliseconds unlike Apollo which has a delay of up to 200 milliseconds with rare falls, that is in all cases greater than the delay of Isaac. We present only one case with 1000 messages per second, because the other results are very similar, and there is no significant difference between them.

You can see that graphs for Apollo in Fig. 14, 16 behave differently and do not overlap each other. On the other hand, graphs for Isaac in Fig. 15, 17 have very similar signatures and values, so they are concentrated in one place. This means that Apollo's pairs work in different ways, unlike Isaac's pairs which work in the same way. The following conclusion can be done for subtest with a different number of pairs, Apollo is unstable: when one of the pairs of Apollo has a delay drop, the others have a peak. In all other cases, all of the pairs have a peak of delay. Isaac, on the other hand, is very stable, all pairs have the similar delay. Also with an increase in the number of pairs, Apollo delay peak grows in contrast to Isaac, whose peak does not depend on the number of pairs.

The delay of more number of pairs is predicted based on analysis of 1,2 and 3 pairs. The Apollo's delay has dependence on the number of pairs and increases by 200 milliseconds with each new pair. Isaac, on the other hand, is very stable and does not depend on this. This way you can increase the number of pairs without performance loss.

CONCLUSION

The resulting data makes it clear that ROS2 is not well suited for creating a reliable real-time system. But the main advantage of ROS2 is the simplicity of designing any system, as well as a large number of publicly available packages that solve various tasks, which speeds up development. ROS has a large amount of training documentation and a fairly large community that can help you solve problems.

Apollo Cyber RT showed good results, but still worse than Nvidia Isaac. This framework is developed for autonomous vehicles and is generally intended for this. Highly specialized software is written for it, which solves the problems of transport with autopilot, and this solution is one of the leading open source frameworks for autonomous transport.

The leader in almost all test scenarios was Nvidia Isaac. This solution is positioned by Nvidia as a simple tool for designing robots. In the architecture of this solution, only the IPC bus can be used for data transmission, for which it is necessary to know in advance all the modules that communicate and enter them in the configuration, and the TCP/IP protocol for other cases. This solution makes it impossible to dynamically connect modules for IPC interaction without making edits to the source code, but as a result, we get extremely small delays.

TABLE II . MAXIMUM/AVERAGE LATENCY AND JITTER OF FRAMEWORKS

Test scenario	**Framework**	**Latency max/avg, ms**		**Jitter max/avg, ms**
Ping-pong of the minimum message size	ROS2	4.08/1.25		2.83/0.078
	Apollo	3.789/0.231		3.56/0.049
	Isaac	0.917/0.012		0.904/0.04
Changing sizes and frequencies of messages (Frequency = 1000 msgs/sec)	ROS2	**Message size, KB**		
		128	13.99/1.49	12.49/0.25
		512	112.38/76.80	46.17/25.02
		1536	2030.06/1880.62	479.30/136.72
	Apollo	128	3.13/0.24	2.89/0.09
		512	3.41/1.651319	1.76/0.39
		1536	2.76/1.81	0.95/0.24
	Isaac	128	1.15/0.09	1.05/0.03
		512	2.28/1.04	1.24/0.35
		1536	2.50/1.19	1.31/0.47

Table II shows the main characteristics for the test *B* and *C*, which show the difference in latency and jitter. In the other 2

scenarios, you can clearly see the graphs presented in the previous section. Analyzing this data, we can say that in all scenarios, Nvidia Isaac showed the best results and is significantly ahead of its competitors.

As a result, the simplest and fastest combined solution for building an autonomous system is Nvidia Isaac. This solution is the youngest and the community is just beginning to learn and use this solution. In general, it is created to replace ROS completely, since it has analogues of all the utilities necessary for development and testing, and also has compatibility with ROS, which allows you to develop modules for a system built on ROS using ROS bridge. In addition, this platform provides additional opportunities for using neural networks in their solutions. It seems that Nvidia Isaac is the most promising platform for creating robots.

REFERENCES

[1] K. Xu, X. Xiao, J. Miao and Q. Luo, "Data Driven Prediction Architecture for Autonomous Driving and its Application on Apollo Platform," 2020 IEEE Intelligent Vehicles Symposium (IV), Las Vegas, NV, USA, 2020, pp. 175-181, doi: 10.1109/IV47402.2020.9304810.

[2] Y. Zhang, H. Sun, J. Zhou, J. Pan, J. Hu and J. Miao, "Optimal Vehicle Path Planning Using Quadratic Optimization for Baidu Apollo Open Platform," 2020 IEEE Intelligent Vehicles Symposium (IV), Las Vegas, NV, USA, 2020, pp. 978-984, doi: 10.1109/IV47402.2020.9304787.

[3] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia and Q. A. Chen, "A Comprehensive Study of Autonomous Vehicle Bugs," 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), Seoul, Korea (South), 2020, pp. 385-396.

[4] J. Xu et al., "An Automated Learning-Based Procedure for Large-scale Vehicle Dynamics Modeling on Baidu Apollo Platform," 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 2019, pp. 5049-5056, doi: 10.1109/IROS40897.2019.8968102.

[5] T. Bijlsma et al., "A Distributed Safety Mechanism using Middleware and Hypervisors for Autonomous Vehicles," 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2020, pp. 1175-1180, doi: 10.23919/DATE48585.2020.9116268.

[6] V. M. Raju, V. Gupta and S. Lomate, "Performance of Open Autonomous Vehicle Platforms: Autoware and Apollo," 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Bombay, India, 2019, pp. 1-5, doi: 10.1109/I2CT45611.2019.9033734.

[7] S. Jose, V. V. Sajith Variyar and K. P. Soman, "Effective utilization and analysis of ros on embedded platform for implementing autonomous car vision and navigation modules," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 877-882, doi: 10.1109/ICACCI.2017.8125952.

[8] B. Abhishek, S. Gautham, D. Varun Rufus Raj Samuel, K. Keshav, U. P. Vignesh and S. R. Nair, "ROS based stereo vision system for autonomous vehicle," 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), Chennai, 2017, pp. 2269-2273, doi: 10.1109/ICPCSI.2017.8392121.

[9] S. Gatesichapakorn, J. Takamatsu and M. Ruchanurucks, "ROS based Autonomous Mobile Robot Navigation using 2D LiDAR and RGB-D Camera," 2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP), Bangkok, Thailand, 2019, pp. 151-154, doi: 10.1109/ICA-SYMP.2019.8645984.

[10] K. Hasegawa, K. Takasaki, M. Nishizawa, R. Ishikawa, K. Kawamura and N. Togawa, "Implementation of a ROS-Based Autonomous Vehicle on an FPGA Board," 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 2019, pp. 457-460, doi: 10.1109/ICFPT47387.2019.00092.

[11] M. Köseoğlu, O. M. Çelik and Ö. Pektaş, "Design of an autonomous mobile robot based on ROS," 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, 2017, pp. 1-5, doi: 10.1109/IDAP.2017.8090199.

[12] N. M. Htaik, N. A. M. Maung and W. Zaw, "Enhanced IoT-based Interoperable and Configurable Middleware using Semantic Web Techniques," 2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Chiang Rai, Thailand, 2018, pp. 90-93, doi: 10.1109/ECTICon.2018.8620032.

[13] E. S. Pramukantoro, A. Kusyanti and Yazid, "Performance Evaluation of Semantic IoT Middleware," 2018 International Conference on Sustainable Information Engineering and Technology (SIET), Malang, Indonesia, 2018, pp. 230-233, doi: 10.1109/SIET.2018.8693193.

[14] E. S. Pramukantoro, W. Yahya and F. A. Bakhtiar, "Performance evaluation of IoT middleware for syntactical Interoperability," 2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS), Bali, 2017, pp. 29-34, doi: 10.1109/ICACSIS.2017.8355008.

[15] W. Liu, H. Wu, Z. Jiang, Y. Gong and J. Jin, "A Robotic Communication Middleware Combining High Performance and High Reliability," 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Porto, Portugal, 2020, pp. 217-224, doi: 10.1109/SBAC-PAD49847.2020.00038.

[16] D. Yu and H. S. Park, "Real-time middleware with periodic service for industrial robot," 2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, 2017, pp. 879-881, doi: 10.1109/URAI.2017.7992853.

[17] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku and Woo-Keun Yoon, "RT-middleware: distributed component middleware for RT (robot technology)," 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, AB, Canada, 2005, pp. 3933-3938, doi: 10.1109/IROS.2005.1545521.

[18] D. Choi, S. Kim, K. Lee, B. Beak and H. Park, "Middleware architecture for module-based robot," 2006 SICE-ICASE International Joint Conference, Busan, Korea (South), 2006, pp. 4202-4205, doi: 10.1109/SICE.2006.31477