Data Distribution Services Performance Evaluation Framework

Kirill Krinkin, Anton Filatov, Artyom Filatov, Oleg Kurishev, Alexander Lyanguzov
Saint-Petersburg Electrotechnical University "LETI"
St. Petersburg, Russia
kirill.krinkin@fruct.org, {ant.filatov, art32fil, battousai45736, xtail1996}@gmail.com

***Abstract*—DDS (data distribution service) is a middleware protocol and API standard for data transferring using a publisher-subscriber model from the Object Management Group (OMG). There exist various open source and commercial implementations of DDS standard that provides API and services for data distribution. Every developer claims that his implementation fits standard and provides the best possible parameters for data transferring. Three different implementations of DDS are compared to determine their usability and performance characteristics. This paper presents a testing framework that allows to evaluate different implementations in the same experiments and moreover to include another DDS.**

I. Introduction

Object Management Group [1] is an open membership, not-for-profit technology standards consortium, that was created by leading IT companies (including IBM, Apple Computer, Sun Microsystems etc.) for developing enterprise integration standards for a wide range of technologies in 1989. One of the developed standards – Data Distribution Service (DDS) – describes a data transfer protocol based on a publisher-subscriber pattern that is one of the most useful patterns for data transfer.

There are two base documents that describe DDS standard: the specification that describes a Data-Centric Publish-Subscribe (DCPS) model for distributed application communication and integration [2], and Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol [3]. According to these documents several vendors such as Vortex [4], eProsima [5], Prismtech [6] etc. provide their implementations of the protocol.

In this work three open source, the most popular and widespread implementations were chosen for comparison. They are presented below:

- OpenDDS (by Prismtech)
- OpenSplice (by Vortex)
- Fast-RTPS (by eProsima)

The testing framework for estimation DDS parameters was developed. The main idea is to put several implementations in the same conditions and to test them as a black box. To perform the same experiment on every DDS it was necessary to implement adapters that have the same input interfaces and provide the data distribution that is based on the corresponding implementation. The output measurements should be presented in the same format so that they may be compared.

The focus of the work is to compare the characteristics of message transportation using different implementations of DDS and to find the one that is the most successful. Moreover the aim is to present the framework that allows to perform this comparison for any DDS implementation even for one that is not considered in this paper.

The paper is structured as follows: in Section II there is the description of existing comparison methods and a relevance of this paper; Section III provides a description of the developed testing framework; the results of the DDS implementation comparison and framework usability are presented in Section IV.

II. State of art

According to DDS standard the transferring of any information happens in the area that is called domain. There is no way to exchange the information between different domains. There are several participants and topics in any domain. Topics - are the channels for exchanging the data while participants can send and receive the information from any topic in one domain. A participant consists of publisher(s) and/or subscriber(s). To have the opportunity of exchanging data in several topics each publisher and subscriber may contain several data writer and data reader that are directly responsible for transferring low level data through topics. In the Fig. 1 the scheme of a domain is presented.

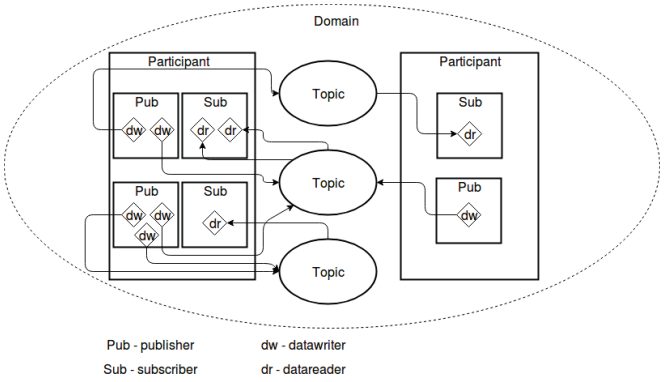


Fig. 1. Example of connections of participants in one domain

The Fig. 2 clarifies the types of connection between components of DDS

Commonly there are three indicators that are measured to test the DDS parameters:

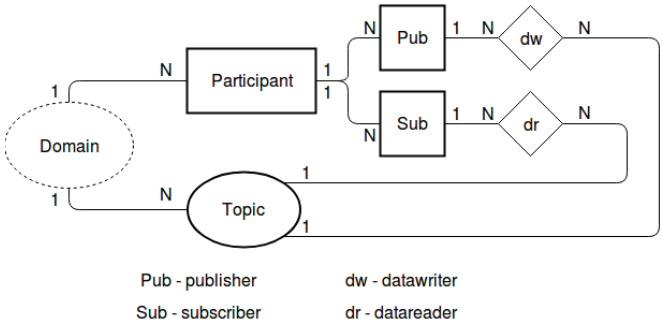


Fig. 2. Illustration of the references of DDS components

1) Latency for transporting a message in dependency on its size. There is a delay between sending messages that is enough to handle each one, so there is no queue of messages in this test.
2) Throughput of a system - amount of messages that can be handled per second in dependency on the size of a message. In this test messages are sent one by one with no delay.
3) Jitter - the difference between time for handling of two consecutive messages.

The developer of OpenSplice calculated the latency and throughput [7]. Latency was measured for two participant and messages no more than 32 kilobytes. Throughput was measured for a bigger graph of participants, but the size of a message was still little. P. Bellavista et all compared OpenSplice with another DDS, RTI [8]. They tested DDS implementations on small graphs with maximum 5 participants.

The tests that were performed by the developer of OpenDDS [9] and Fast-RTPS [10] are the same as those that were mentioned above: the latency was measured for small size of messages.

Rizano et all [11] evaluated the latency of several DDS including OpenDDS. In that work the latency was measured for the graph containing one publisher and several subscribers (their amount varied from one to five) and for message size not larger than 1000 bytes.

F. Martin et all [12] has compared the performance of Fast-RTPS in different settings such as 'best efford' or 'reliable' distribution. In that work authors measure the latency and also the percentage of messages looses. They performed the tests on different network technologies: Ethernet and wifi.

In this research two first indicators are united to make the system closer to the real one. The general scenario of testing is that a certain graph of publishers and subscribers is assigned, where each publisher writes 100 messages of some length into some topic without a delay. Each message contains a field where the time of sending a message is specified. All subscribers who read this topic calculate the time spent on the transfer as soon as they receive a message. Because there are many messages and sending can take less time than processing (especially for large messages), the processing time for each new message increases. This allows to determine both latency and throughput in one test. latency is the time to send the first messages, and the throughput can be estimated by the degree of increase in the time for processing messages. The jitter is also always measured.

III. The structure of testing framework

To solve tasks mentioned above, the testing framework was developed. The purpose of its creation was to provide opportunities to conduct identical tests under the same conditions over different DDS. Since the objects of research were exactly DDS implementations, the main measured characteristic was the time for transferring messages. The greatest interest is provided by the time that DDS middleware takes to transmit messages, not considering the processing. Thus, exactly the moment of time when the message was sent and also the moment when the message appeared should be evaluated precisely.

In order to uniformly conduct tests for all DDS, one must create an external interface for each implementation. The interface should provide the functionality of creating a separate publisher process and a separate subscriber process, specifying the topic with which this participant will interact. Such a design solution allows not to think about those cases where one participant can simultaneously write in two topics and read from three. To calculate the time required only for sending a message, it is sufficient to use participants, who represents only one reader or only one writer.

Thus, as input to the test framework, two components can be distinguished:

- a text file representing the connection between writers and readers;
- custom files that create the reader and writer processes for each of the tested DDS.

So a structure of the test framework could be presented with a following list:

- Implementation of common API for every considered DDS – this part will be called adapter in the following text. This implementation presents inheritances for abstract classes of publisher and subscriber which implementations are unique for every DDS but could be launched in a same way using this API.
- Scripts that allow to launch all publishers and subscribers consistently for every DDS and store their result outputs.
- Scripts that presents results from previous step with plots for friendly visual analysis.

This structure is presented on Fig. 3

Adapter should consists of BasePublisher and BaseSubscriber classes. One BasePublisher creates one participant in a common domain, one publisher in this participant and one data writer. BaseSubscriber creates one participant, one subscriber and one data reader correspondingly. So BasePublisher as well as BaseSubscriber are separate processes and moreover two for example BasePublishers are separate processes too. Separating processes could demonstrate true power of DDS, because one of ideas for DDS standard allows two participant exist as separate parts in a local network, where there is no ability to use shared memory.

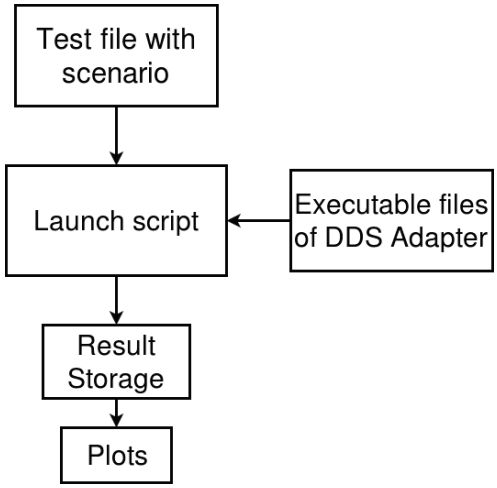


Fig. 3. Test framework structure

Except creating participants, readers, writers etc. adapter should send messages through named topic. Another unique idea of developed test framework is based on publishing messages non-stop. Every test from other sources is based on idea that there is a delay between publishing messages. So there is no opportunity to measure an execution time in stress, when there are hundreds participants and everyone publish messages in their own rhythm.

The important aim of the testing framework is to estimate the performance of DDS implementations. That means that it is necessary to calculate time taken for message transferring without handling. So BasePublisher should implement an instrument for storing information about message pushing and BaseSubscriber – about message pulling with an opportunity to link the time of sending with the time of receiving, avoiding collisions with a large number of messages.

Using an idea of separate processes where only one data reader or data writer exist it is very easy to present a graph of links between publishers and subscribers. In this case one publisher is able to write in only one topic, and one subscriber – read from only one topic. So graph of links is presented as a set of topic names and identical number of participant that should be connected to this topic.

IV. Results

The seven experiments that present quantitative characteristics of each DDS were considered in this work. The complexity of graph increased from first test to the last one to show the changing of time that is required to handle messages in an unloaded system and an overloaded one. All settings of DDS were set as default, so the transaction mode was set as ”reliable” which means that every subscriber must get every message, and every topic stores all the messages and doesn't drop or delete them. The description of considered tests is presented below.

1) The graph consists of one publisher and one subscriber. The size of the message varies from 100 bytes to 1 megabyte. Messages of one size are sent in isolation from messages of another size in the amount of 100 samples. This test is designed to check the latency value declared by a developer. The graph structure is presented in a Fig. 4a. The latency result is presented in a Fig. 4b, where the abscissa indicates the size of messages. The jitter is shown in a Fig. 4c.
2) The graph consists of 9 publishers, 9 subscribers, all linked through one topic. Thus, each publisher sends 100 messages, and each subscriber receives 900 messages. The bound for processing messages in this test is 30 seconds. The size of the message varies from 100 bytes to 1 megabyte. The graph structure is presented in a Fig. 5a. The latency result is presented in a Fig. 5b, where the abscissa indicates the size of messages. The jitter is shown in a Fig. 5c.
3) There are 45 publishers and 45 subscribers that are linked through 5 topics. Each publisher sends 100 messages, thus at one moment the middleware should handle a huge amount of messages. The size of the message varies from 100 bytes to 100 kilobyte. The graph structure is presented in a Fig. 6a. The latency result is presented in a Fig. 6b, where the abscissa indicates the size of messages. The jitter is shown in a Fig. 6c.

After the testing, the obtained data was analyzed in the following manner. Figures 4b,5b,6b, shows the overall dynamics of changing of the execution time. It shows that openDDS performs the most quickly in all tests, and in addition, it demonstrates a little change in the time during the processing of messages.

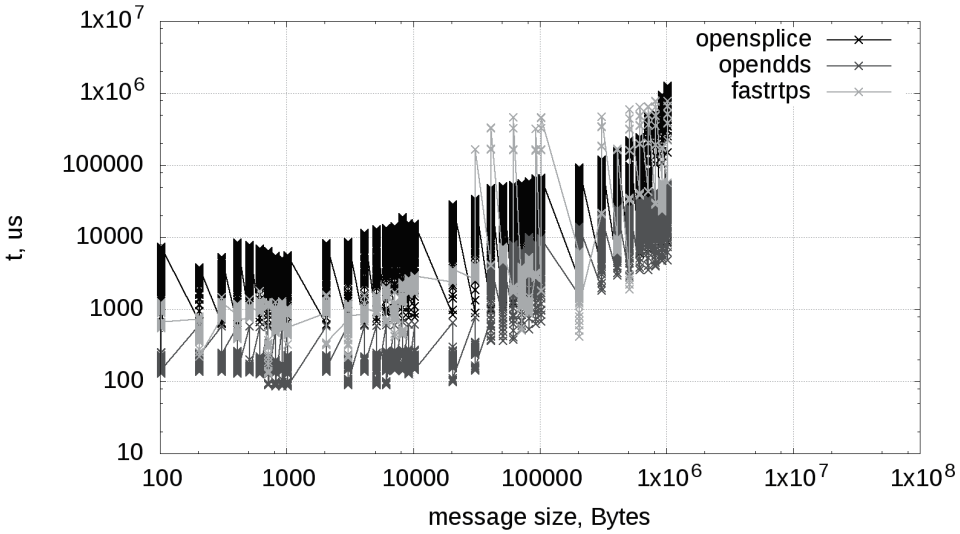
Plots in Figure 4 present results in simplest test case - one publisher and one subscriber connected with common topic. This is an example where it is possible to measure delivery time for one message and to evaluate the time that is required for DDS when it is not overloaded. The average time of all tests in this test case begins near 1 ms and ends near 100 ms. OpenDDS is the fastest, OpenSplice is slower, but generally they behave the same way. Time increases going through message size, and transferring messages with 100 bytes size are in ten times faster than 1 megabyte size.

On Fig. 5 and 6 amount of participants is bigger –18 and 90 correspondingly. These are stress tests, when there could be requested transfer of 1 GB of data in one moment. Latency obviously become bigger too. In these test cases average time is measured in seconds, tens of second or hundred of seconds. This time is taken only for receiving messages and there is no handling.

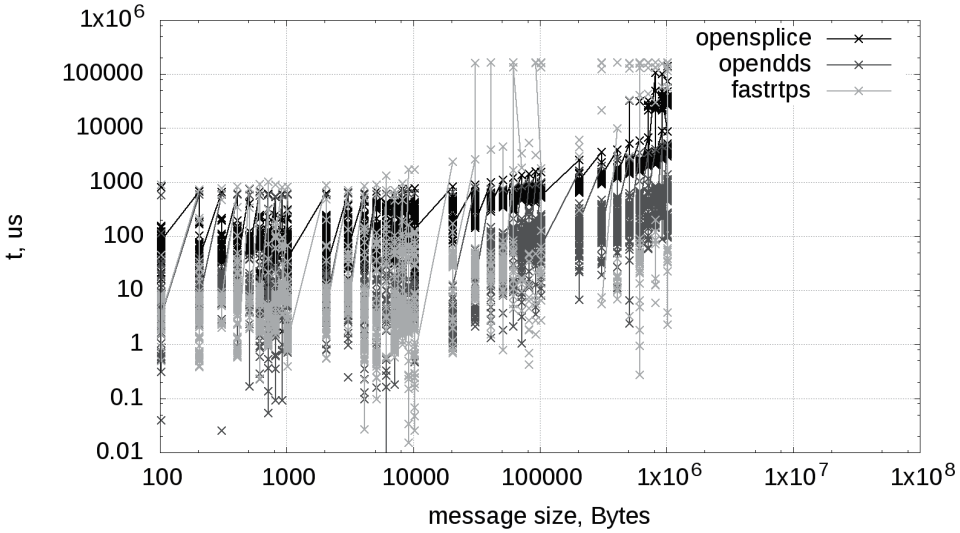
Obviously increasing amount of participants leads to the increasing amount of delivery time. But there is no ability to run for example OpenSplice on huge amount of participants because this DDS has a upper bound of it. There could be only 120 participants [13].

In the presented figures one can extract the time required to deliver the first message. It can be assumed that this time is latency. Thus, it is possible to compare the data obtained in the considered test cases with the data presented in [8] [12] [11]. It should be noted that in the current work and in the works listed above, various graphs of writers and readers were examined, but it is possible to note that the data obtained using the

(a) Links in 1st test case



(b) Latency for 1st test case



(c) Jitter for 1st test case

Fig. 4. Output plots for 1st test case

developed testing framework coincide with the data obtained by the authors of other works.

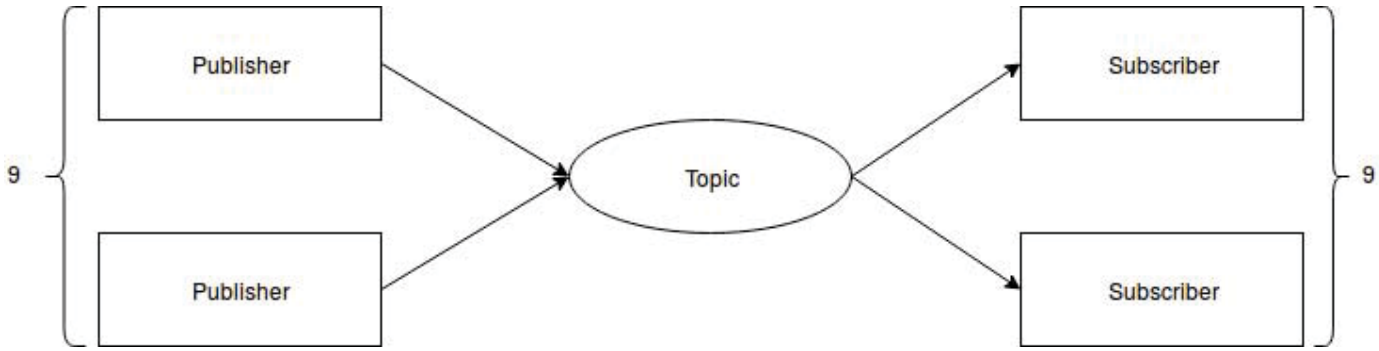
V. Conclusion

Three most popular and widespread implementations of DDS Standard were considered and compared in this paper: OpenSplice, OpenDDS and Fast-RTPS. Moreover the framework for testing these three implementations was implemented. The framework is constructed in the way that it is easy to include any another DDS.

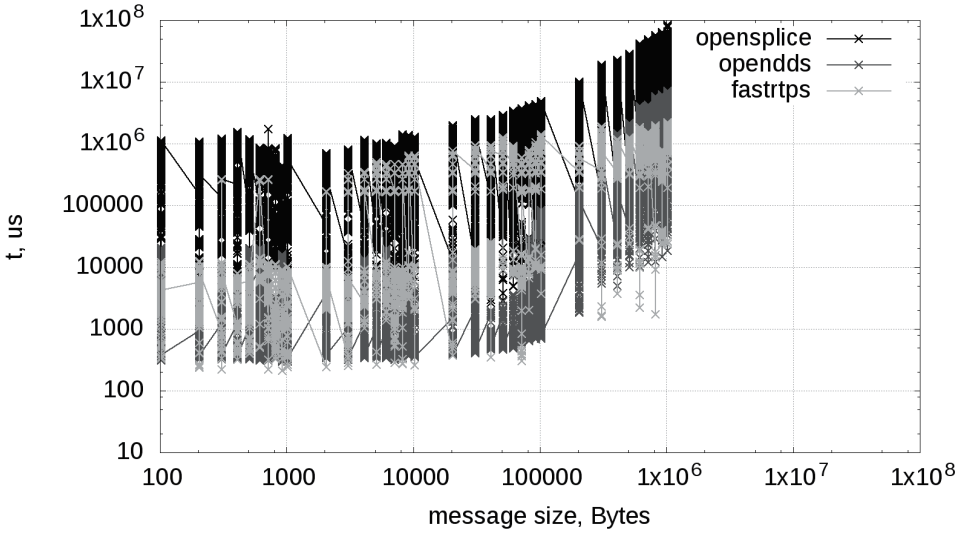
It is important to mention that in this work all implementations were tested in ”safe” mode which means that all messages were stored in memory all the time, every message was followed with a confirmation about delivery and that is the reason why the time of processing each next messages increased. This behavior is also caused by not real-time kernel of operating system.

Tuning parameters of DDS implementations, such as decreasing reliability of system, dropping messages after they have been received might decrease the latency and some DDS might work faster.

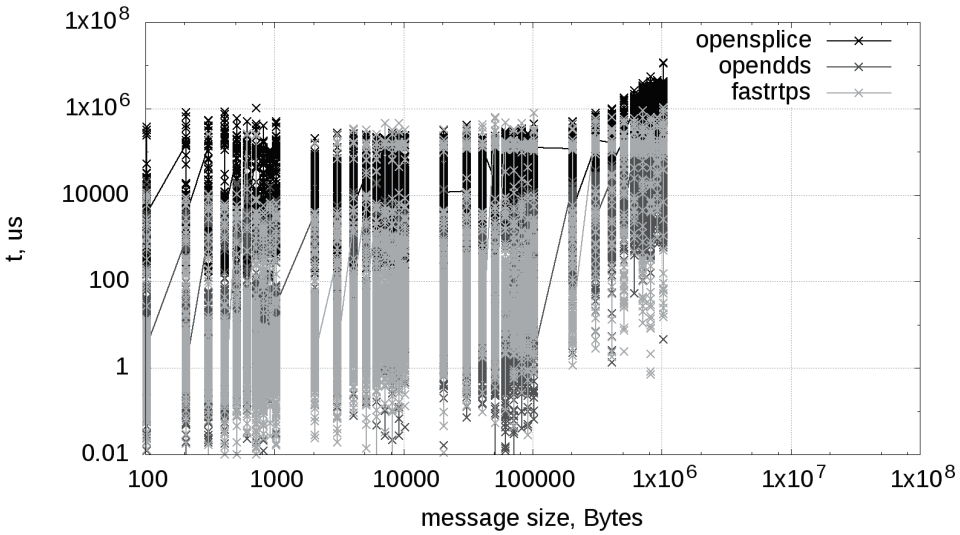
Looking through the received results one can say that in



(a) Links in 2nd test case



(b) Latency for 2nd test case



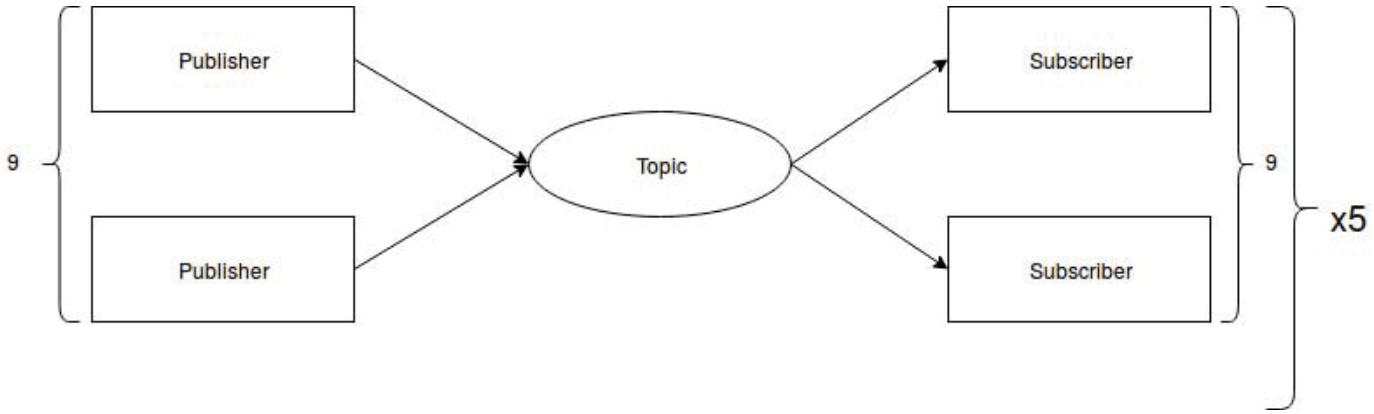
(c) Jitter for 2nd test case

Fig. 5. Output plots for 2nd test case

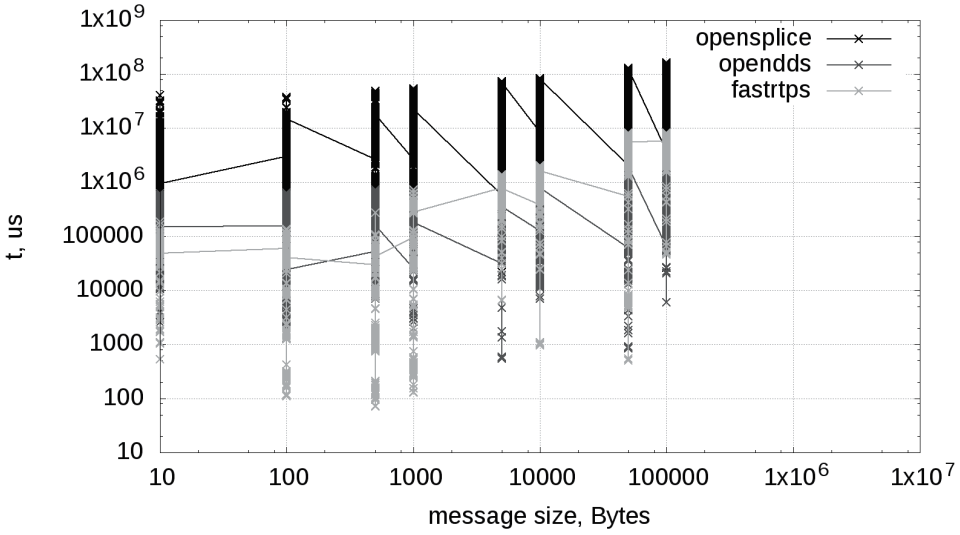
unloaded system the delivery time is about 0.1 ms that might be too much for real-time systems. In the future it is required to run the same test on the real-time operating system, since DDS standard was developed on a real-time publisher-subscriber protocol and it should suit real-time tasks.

VI. Acknowledgment

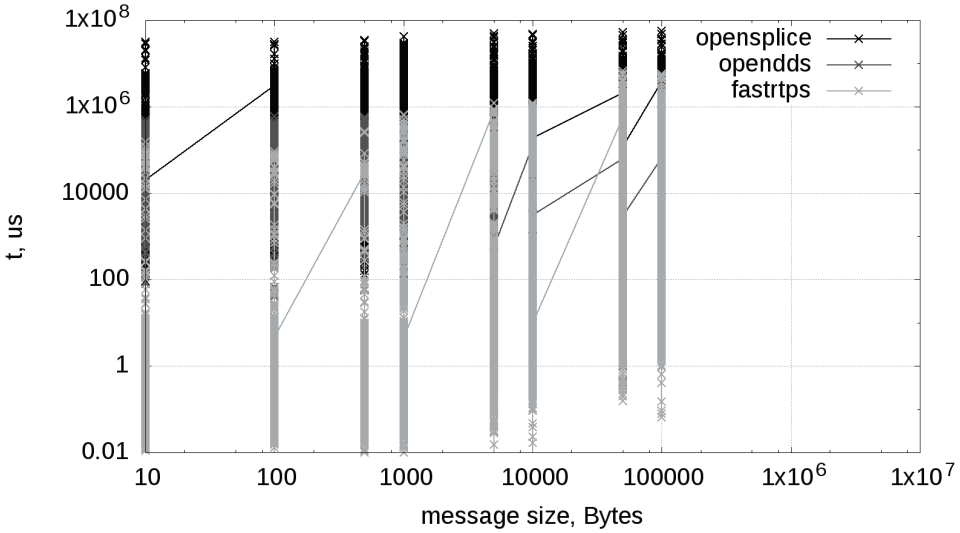
Authors would like to thank TRA Robotics for provided support and materials for working on this research.



(a) Links in 3rd test case



(b) Latency for 3rd test case



(c) Jitter for 3rd test case

Fig. 6. Output plots for 3rd test case

REFERENCES

[1] "Object management group." https://www.omg.org/.
[2] "Data distribution service specification version 1.4." http://www.omg.org/spec/DDS/, 2015.
[3] "Dds interoperability wire protocol specification version 2.2." http://www.omg.org/spec/DDSI-RTPS/, 2014.

[4] "Vortex opensplice." http://www.prismtech.com/vortex/vortex-opensplice.

[5] "Eprosima fast-rtps." http://www.eprosima.com/index.php/products-all/eprosima-fast-rtps.

[6] "Prismtech opendds." http://opendds.org/.

[7] "Vortex opensplice performance results." http://www.prismtech.com/vortex/vortex-opensplice/performance.

[8] P. Bellavista, A. Corradi, and L. Foschini, "Data distribution service (dds): A performance comparison of opensplice and rti implementations," in *Computers and Communications (ISCC), 2013 IEEE Symposium on*, IEEE, July 2013.

[9] "Prismtech opendds performance results." http://opendds.org/perf/lab100125/latency_results.html.

[10] "Eprosima fast-rtps performance results." http://www.eprosima.com/index.php/resources-all/performance/40-eprosima-fast-rtps-performance.

[11] T. Rizano, L. Abeni, and L. Palopoli, "Experimental evaluation of the real-time performance of publish-subscribe middlewares," 2013.

[12] F. Martin, E. Soriano, and J. M. Canas, "Quantitative analysis of security in distributed robotic frameworks," *Robotics and Autonomous Systems*, vol. 100, pp. 95 – 107, 2018.

[13] "The opensplice deployment guide." http://download.prismtech.com/docs/Vortex/html/ospl/DeploymentGuide/guide.html#participantindex.