Computing Science

Latency Performance Comparison of Open Source Real-Time Publish-Subscribe Data Distribution Service Implementations

Marcel Zak

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: November 28, 2017

CS4040 Report

Department of Computing Science
University of Aberdeen
King's College
Aberdeen AB24 3UE

November 2017

Latency Performance Comparison of Open Source Real-Time Publish-Subscribe Data Distribution Service Implementations

Marcel Zak

Department of Computing Science
University of Aberdeen

November 28, 2017

Abstract: Will be

1 Introduction

In this research paper I decided to investigate a difference in latency performance in two popular open source implementations of Data Distribution Service (DDS) protocol. DDS is middleware developed by Object Management Group (OMG) [7]. It is specifically design as machine to machine, data centric, high performance and reliable protocol. The aim was to create a protocol that does not have a problem with scalability and it fulfils requirements for modern Internet of Things (IoT) applications [9]. Later a Real-Time Publish Subscribe (RTPS) DDS was specified as the interoperability protocol. RTPS DDS was designed as real time, low latency protocol for best effort and reliable communication over unreliable transports such as UDP/IP and it is standardized that different RTPS implementations can communicate each other [8]. For its quality and robustness, it is often used in aerospace and defence sectors for real-time applications [5].

For real time applications it is critical to deliver the right message in the right time at the right place. Such systems are widely used in automation, robotics, space industry, medicine and many other places. Time and reliability are one of the most important factors in these systems. Therefore, the right choice of a library is also crucial to the application.

These two open source libraries implement the same RTPS protocol specification. They are implemented in same programming language C++. The first implementation is OpenDDS [12] created by OCI that is a full implementation of DDS. The second library was developed by eProsima and it is standalone RTPS implementation called Fast RTPS [3]. It claims to be the fastest one because it is lightweight [5]. In this paper I will be investigating latency performance of both open source RTPS implementations.

2 Background and related work

As a first thing I describe the underlying design principle behind DDS protocol. Participants (applications) using DDS can publish or subscribe to a specific ”topic” of information. A single participant can publish and subscribe to multiple ”topics” at the same time. The advantage of DDS against other similar protocols is that it implements rich set of Quality of Service parameters [7]. It is possible to set up different aspects of the communication. Be it persistence, reliability, redundancy, transport settings, lifespan and many others. The communication model is not difficult to understand. A topic is a name that refers to a specific piece of information we want to share. It also contains the definition of Data we want to share [4]. See Figure 1.

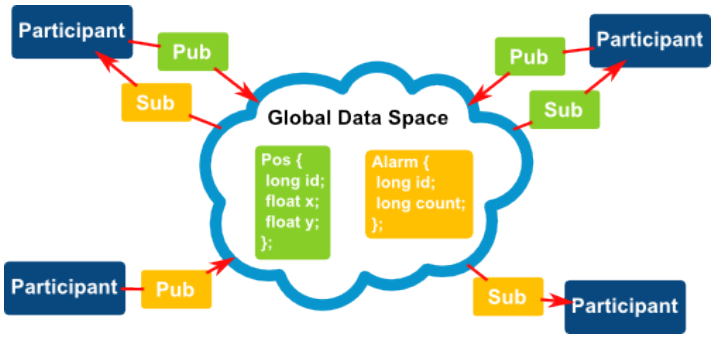

Figure 1: DDS Model: The Global Data Space [4]

DDS provide a simple modular design that is shown in Figure 2. In these two open source implementation is DDS only a library that is linked to an application. There is no need to install any service or daemon. It is important to note that RTPS is on top of transport layer in the OSI model. Therefore, DDS can be implemented over any kind of underlying transport [4]. DDS specification does not outline how the message should be delivered. Therefore, multiple implementation exist.

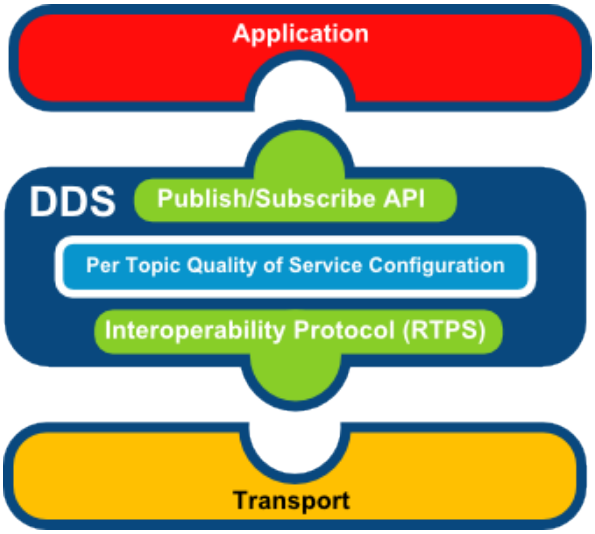

Figure 2: DDS Architecture [4]

I have to say that the scientific community lacks papers about DDS benchmarking. However, there are two papers worth mentioning. The first one is *Performance assessment of OMG compliant data distribution middleware* [6]. In this paper it is thoroughly discussed the problematic and complexity of benchmarking different DDS implementations. In the second paper *Data Distribution Service (DDS): A performance comparison of OpenSplice and RTI implementations* [1] is described the process of throughput, latency and CPU load comparison of OpenSplice and RTI DDS implementations.

Authors of the first paper decided to develope a tool called DDSBench that should provide framework for objective performance comparison of different DDS implementations. DDSBench offers variety of possible set-ups such as message type, size, publication period, message iteration, QoS and many others. The idea is that a developer model the test environment based on requirements of the specific application and run the test. Results of the test show which DDS implementation is suitable for the application. It is also discussed different methods for latency measurement. Standard method is *Round-Trip time*. It is the time elapsed between sending a message and receiving it back. However they decided to use *One-way Transit Time* that represent time elapsed between sending the message from publisher to subscriber. The advantage is that it represents more accurately the elapsed time from subscriber's point of view. On the other hand, there is disadvantage and difficulty involved with precise clock synchronisation on both machines [6]. I don't agree that for the purpose of benchmarking two implementations is necessary to use *One-way Transit Time*. Based on the related literature *Round-Trip time* reveals the difference in implementations [10]. At the end DDSBench is used to evalueate two implementations. The first one is RTI DDS and OpenDDS [6]. The results clearly showed that the RTI implementation has better performance in all dimensions. I tried to use DDSBench with the newer versions of OpenDDS. Unfortunately, I did not manage to run it probably because the last update of DDSBench was in 2007.

At the beginning of the second paper the DDS standard is well described. The next part focuses on metrics and measurement techniques. Authors decided to measure five different parameters to completely evaluate the performance differences of OpenSplice and RTI implementations. The measured parameters are *Samples per sencon, Throughput, Round-Trip time, CPU and Memory usage*. For latency measurements is used standard *Round-Trip time*. Authors also mentioned that this measurement of latency is widely accepted [10]. At the end they performed benchmarking and the results revealed that OpenSplice performs significantly better than RTI in throughput tests and samples per second, but only with messages of smaler size(up to 2048 bytes). If the size of the messages is higher then RTI performs much better. OpenSplice utilizes federated architecture [14] and by default uses batching optimization this enables better performance for small size data exchange. On the other hand decentralized models (no need for separate daemon), such as RTI implementation, do not have single point of failure or latency issue. They are also more suitable for large size data exchange [1].

3 Research question

It has been shown that DDS implementations differ in performance even though they implement the same communication standard RTPS. The research papers mentioned above always compared proprietary with open source or partially open source implementation of DDS RTPS protocol. But in the situation when a developer have to choose only open source implementation, there has never been done any performance comparison. Latency is one of the most important factors in real time or critical applications. Therefore, I decided to investigate latency performance of OpenDDS [12] and eProsima Fast RTPS [3] implementations.

I chose these two implementation for their popularity and still active development. OpenDDS implements federated model meanwhile Fast RTPS uses fully distributed model. As described in previous section, each has its advantages and disadvantages. It is not possible to only look at performance tests of both implementations shown on their web pages because these tests were not done in the same testing environment [2, 11]. Therefore, the latency times vary and cannot be directly compared.

The research questions are as follows:

- Does federated model of OpenDDS introduces latency overhead in comparison to fully distributed model?
- How does the size of a message influence latency time in federated model vs fully distributed one?
- Which of those two implementation should a developer choose if the latency performance is the most important factor?

In order to answer these questions, I conducted and experiment that measured the latency of both implementations. I used several standard message sizes to evaluate the behaviour under different scenarios. For the purpose of measurement latency I utilized already implemented performance tests in both libraries. They are using *Round-Trip time* measurement for estimation of latency.

4 Experimental Design

What are your hypotheses? How are you going to test them? What is your target population? What are your datasets; i.e. your sample of the target population. What are the dependent and independent variables?

Guide length: 500 words.

Based on the literature review and studied materials my directional hypothesis is that eProsima Fast RTPS will have better latency performance than OpenDDS implementation regarding all tested data sizes.

I will measure the latency using loopback interface from start of send to end of read. It is the same concept as if I had two separate hosts. In this scenario host1 = host2. See Figure 3

For this experiment I used one laptop HP EliteBook 2540p with the following parameters:

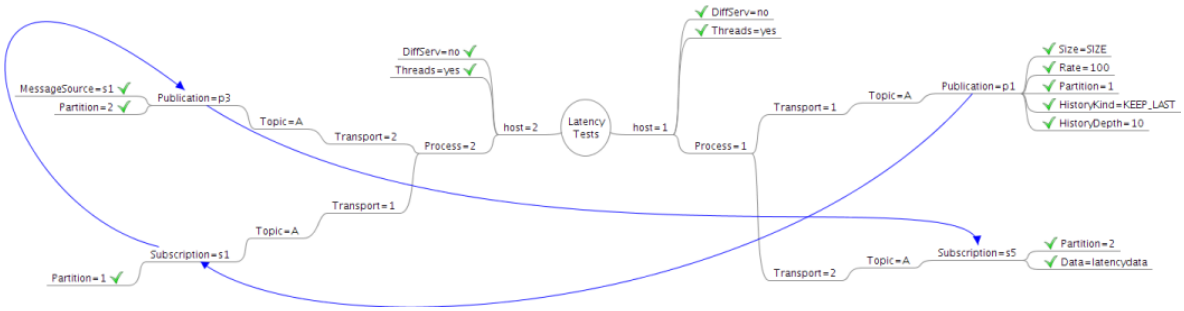

Figure 3: Latency test topology [13]

- CPU: Intel(R) Core(TM) i5 CPU M 540 locked at 1.2GHz frequency
- Memory: SAMSUNG 8GB 2x4GB PC3-10600 DDR3 1333MHZ
- Operating System: Fedora 27 64-bit
- Swappiness: 0
- Used compiler: gcc version 7.2.1 20170915 (Red Hat 7.2.1-2) (GCC)
- CMake: version 3.9.6
- OpenDDS version 3.12 (commit: 3057ef9dbb032bc51895f70e5c350af4b10093fa)
- eProsima Fast RTPS (commit: bec4a979c088a1f978af3dfdea2a72ae2a78b52a)

The latency will be measured with *Round-Trip time* and the divided by two. This will represent latency of a message from a publisher to a subscriber. I will discard first 1000 time measurements and then collect subsequent 10 000 measurements of *Round-Trip time* divided by two. This process will repeat for every message size. The sizes of a message will be 16,32,64,128,256,512,1024,2048,4096,8192 and 16384 bytes.

After all data is collected, I will calculate mean, max, min and standard deviation for every message size time measurements. Then I also perform t-tests for the same message size time measurements.

5 Results

Present the results. A good way to organise this is via subsections for each hypothesis you tested. Include graphs of results (e.g. Figure **??**), tests of significance, etc. If you have negative results, include them. A negative results is just as informative and useful as a positive one, sometimes more so.

Guide length: 500 words.

6 Discussion

What do the results say? What have you learned from the experiments? Have you identified a correlation between variables, or causation? What are the limitations of what you've done? What further experiments might be of benefit?

Guide length: 400 words.

7 Conclusion

What have you done and why? What have you shown through your experiments?

Guide length: 100 words.

References

[1] Paolo Bellavista, Antonio Corradi, Luca Foschini, and Alessandro Pernafini. Data distribution service (dds): A performance comparison of opensplice and rti implementations. pages 000377–000383, 07 2013.

[2] eProsima. eprosima fast rtps performance. http://www.eprosima.com/index.php/resources-all/performance/40-eprosima-fast-rtps-performance. Accessed: 23 Nov, 2017.

[3] eProsima. Github eprosima fast rtps. https://github.com/eProsima/Fast-RTPS. Accessed: 23 Nov, 2017.

[4] eProsima. Introduction to dds. http://www.eprosima.com/index.php/resources-all/dds-all. Accessed: 23 Nov, 2017.

[5] eProsima. Rtps introduction. http://www.eprosima.com/index.php/resources-all/rtps. Accessed: 23 Nov, 2017.

[6] Christian Esposito, Stefano Russo, and Dario Di Crescenzo. Performance assessment of omg compliant data distribution middleware. pages 1–8, 04 2008.

[7] Object Management Group. What is dds? http://portals.omg.org/dds/what-is-dds-3/. Accessed: 23 Nov, 2017.

[8] Object Management Group. Dds interoperability wire protocol. http://www.omg.org/spec/DDSI-RTPS/, 2014. Accessed: 23 Nov, 2017.

[9] Object Management Group. Data distribution service. http://www.omg.org/spec/DDS/, 2015. Accessed: 23 Nov, 2017.

[10] James Edmondso n Hieu Nguyen Douglas C Schmidt Ming Xiong, Jeff Parsons. Evaluating the performance of publish/subscribe platforms for information management in distributed real-time and embedded systems. http://portals.omg.org/dds/sites/default/files/

Evaluating_Performance_Publish_Subscribe_Platforms.pdf/. Accessed: 28 Nov, 2017.

[11] OCI. Comparing opendds and zeromq usage and performance. http://mnb.ociweb.com/mnb/MiddlewareNewsBrief-201004.html. Accessed: 23 Nov, 2017.

[12] OCI. Github oci opendds. https://github.com/objectcomputing/OpenDDS. Accessed: 23 Nov, 2017.

[13] OpenDDS. Opendds - performance test descriptions - latency tests. http://opendds.org/perf/tests/latency_tests.html. Accessed: 23 Nov, 2017.

[14] Wikipedia. Federated architecture. https://en.wikipedia.org/wiki/Federated_architecture. Accessed: 28 Nov, 2017.