Security and Performance Trade-offs for Data Distribution Service in Flying Ad-Hoc Networks

Radek Fujdiak[1], Jiri Pokorny[1], Lukas Zobal[1], Peter Popov[2], Vladimir Stankovic[2], Petr Mlynek[1], Pavel Mrnustik[3], Petr Blazek[1], Petr Musil[1], Jiri Misurec[1]

[1]Brno University of Technology, Antoninska 548/1 Brno 601 90 Czech Republic
[2]City, University of London, Northampton Square Clerkenwell London EC1V 0HB United Kingdom
[3]Trustport, Veveri 2581/102 Brno 616 00 Czech Republic

Email: {fujdiak, jiri.pokorny, lukas.zobal, mlynek, blazekpetr, xmusil56, misurec}@vutbr.cz,
{P.T.Popov, Vladimir.Stankovic.1}@city.ac.uk, {Pavel.Mrnustik}@trustport.com

***Abstract*—This paper focuses on the data distribution service (DDS) middleware and its publish/subscribe logic - a topic that has recently regained popularity in both academia as well as industry. DDS is a well-known approach based on publish-subscribe logic. Therefore, only brief introduction of the issue is given followed by practical evaluation of current, available and real implementations from the security and performance point of view. The analysis and evaluation is performed to aid comparison of competing DDS implementation, and thus could serve well as an input to decision-making about which of these solutions is best suited for a given situation. Finally, the practical performance evaluation is performed via several different scenarios to effectively compare the currently most-used DDS implementations.**

***Index Terms*—FANET, Drones, DDS, Security, Performance**

I. Introduction

Data Distribution Service (DDS) is a middleware protocol and a standard API for data transmission using publish-subscribe model of the object management group (OMG) [1]. There are various open source and commercial DDS implementations that provide APIs and data distribution services. Real-Time applications often need to be distributed to multiple compute nodes. The reasons are the distribution of computing power to where it is needed, simplification of design, management or maintenance of the application. An important part of all distributed applications is communication between application components on different nodes. For real-time applications, communication is subject to temporal constraints such as deadlines. Often, applications must be fault-tolerant and support redundancy in their architecture. Another common requirement is the dynamic nature of the application where the nodes are components that are linked or removed from the application during run-time. These, often contradictory, requirements make it difficult to design and implement the communication part of the application. Therefore, people often create applications on various communication middleware platforms that are in charge of communication.

Traditional middleware platforms such as CORBA [2] provide transparent access to remote objects via remote method calls. When an application prompts a method on a remote object, the middleware automatically aggregates method parameters and sends a request to the target process where the object is located. The calculation is done remotely, then the results are sent back. Although this simplifies the creation of distributed applications, there are many applications that cannot be effectively implemented on this request-response model [3]. Applications whose data-oriented operation means that part of the action or calculation is done when the data is ready would be suitable for middleware platforms that seamlessly manage the distribution of data from producers to consumers. Such applications are often designed according to the data-centered publish subscribe model (DCPS) [4], where middleware creates the concept of a global data space that is accessible to all interested applications (see Figure 1). Writing applications according to this model has many advantages. Most importantly, communication requirements are specified by applications in a declarative manner, and the middleware controls data exchange automatically according to declarations.

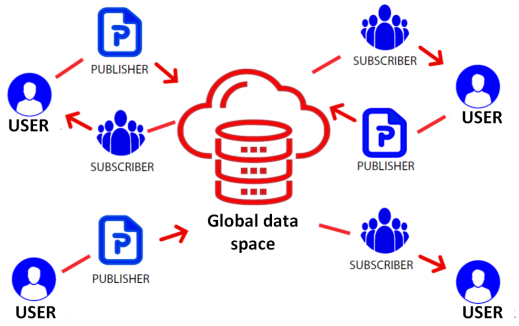

Fig. 1. Simplified data-centric application model of publish-subscribe.

The rest of the paper is organized as follows. Section II provides basic information about DDS systems. Section III contains brief introduction to the different DDS implementations. Section IV presents our results from feature-analysis, while, Section V provides performance-analysis. Finally, Section VI concludes our findings and contribution.

II. DDS Systems Description

DDS is a natural contrast to the existing, already well-known CORBA model. In DDS we access method requests on remote objects via the interface defined in the interface description language (IDL). In CORBA, data is communicated indirectly through arguments in method requests or via their return values. However, in a large number of real-time applications, the communication template is often modeled as a data-centered exchange, where applications publish stream data, which is then accessible to remote applications that are interested in the data. The main focus is on efficient data distribution with minimal extra activity and the need to enlarge to hundreds or thousands of subscribers in a robust way (see Figure 2).

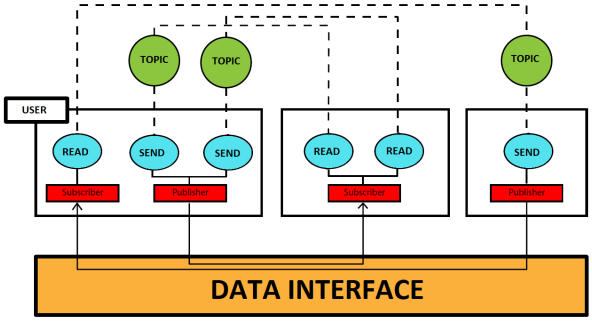

Fig. 2. Simplified diagram of elements within a DDS domain.

The classic publish-subscribe model has been supplemented by RTPS protocol with time parameters and a parameter defining the mode of data delivery. The publish-subscribe distribution model uses the interlayer to create nodes. Publisher is a node that manages data production. Publisher uses the intermediate layer to register the title and type of the data it will publish. Data is produced as fast as the technology used to retrieve it (e.g., the sensor releases data every 10ms). There is no parameter to set the data publishing speed. Publishers that produce identical data may be present on the network. Subscriber is a node that obtains the data. Subscriber registers for the topic and type (type name) data of publication it needs using the interlayer. Subscriber defines two time parameters: minimum separation and deadline. Minimum separation is the time interval measured since the last message received. New data is not received for the duration of the interval. This period is for Subscriber to process the previously received publication. After this time, Subscriber will start receiving data again. The primary discriminator between DDS and other approaches is the unique Quality of Service (QoS) condition set. This set contains 21 key parameters that enable a dynamic, tunable and scalable real-time network. However, this large set also points to the possibility of confusion about which settings allow optimal performance in a given configuration. Even though specific DDS implementations already have default settings (or sometimes they may not fully support all DDS QoS parameters), it is still in active research stage (see Figure 3).

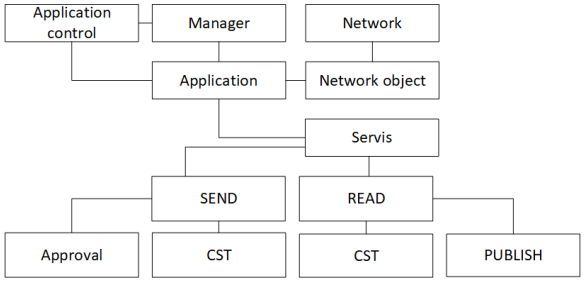

Fig. 3. The object model that underlies the RTPS protocols.

Another requirement for real-time applications is the need to control QoS properties that affect predictability, overhead and resources used. Distributed Shared Memory is a classic model that offers data-centered exchanges. However, this model is complex and ”unnatural” for efficient use over the Internet. Thus, another model - data-centric publish subscribe (DCPS) - has become popular in a large number of real-time applications. Although there existed several commercial and open-source software systems built on this principle, no generalized data distribution standard was available until OMG DDS. Thus, no conventional models directly support a data-centered information exchange system. OMG DDS is an attempt to solve this situation. The specification also defines the operations and QoS attributes that these objects support and the interfaces that the application can use to be notified of data changes or to wait for specific changes. The High-Level Architecture (HLA), also known as OMG Distributed Simulation Facility, is a standard from both IEEE and OMG [5]. It describes data-oriented publish-subscribe facility and data model. The OMG specification is an IDL specification and can be mapped to multiple transports. The specification describes some of the requirements of data-focused publish-subscribe: the application uses the publish-subscribe interface to interact with middleware, includes a data model, and supports content-based subscriptions. The HLA data model supports the specialization hierarchy but does not support the aggregation hierarchy. A set of defined types cannot expand over time. Data elements are unwritten and disordered, they are pure octet sequences. HLA also does not offer QoS facilities. The DDS infrastructure layer that allows different types of applications to communicate with each other is displayed in Figure 4.

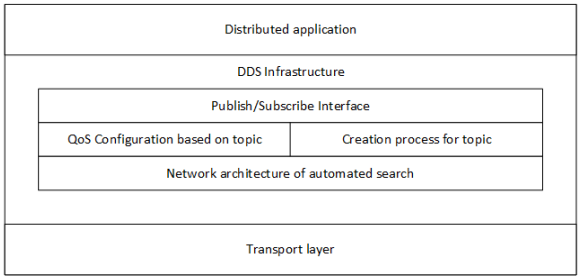

Fig. 4. DDS infrastructure layer.

TABLE I
OVERVIEW OF SELECTED ATTRIBUTES OF DDS IMPLEMENTATIONS.

DDS Implementation	Organization	License	DDS Security	Transport protocols	Operating systems	APIs
OpenDDS	OCI	Open Source	Implemented	Not Specified	Windows, Linux, Solaris, MacOS	C++
sDDS	Community	Open Source	Not implemented	Not Specified	Linux, RIOT-OS, Contiki	C
DDS Community	ADLINK	Open Source	Not implemented	TCP, UDP	Windows, Linux	C, C++, Java, C#, STREAMS, GPB
Vortex OpenSplice	ADLINK	Commercial	Implemented	TCP, UDP	Linux, Windows, AIX, Solaris, RTOS, VxWorks, RTLinux, Integrity, PikeOS, ElinOS	C, C++, Java, C#, STREAMS, GPB, RMI, MATLAB
Connext DDS	RTI	Commercial	Implemented	TCP, UDP	Linux, Windows, Mac OS	C, C++, C#/.NET, Java
CoreDX DDS	Twin Oaks Computing	Combined	Implemented	Not specified	Linux, WIndows, Mac OS, Solaris, Integrity, Lynx OS, VxWorks, DeOS, QNX, Android, iOS, Free RTOS, Thread-X, NexusWare, Unison	C, C++, C#, Java
Eclipse Cyclone DDS	Eclipse Foundation	Open Source	Not implemented	Not specified	Linux Ubuntu, Windows 10	C
InterCOM DDS	Kongsberg Geospatial	Commercial	Not-specified	UDP	Linux, Windows, VxWorks	C++, Java, C#, ADA
Mil-DDS	MISOFT	Commercial	Partialy	UDP	Windows, Linux, Solaris, VxWorks	C++, C#/.NET, Java

III. OVERVIEW OF AVAILABLE DDS IMPLEMENTATIONS

The main available implementation of DDS currently used are: (i) OpenDDS, (ii) sDDS, (iii) DDS Community, (iv) Vortex OpenSplice, (v) Connext DDS, (vi) CoreDX DDS, (vii) Eclipse Cyclone DDS, (viii) InterCOM DDS, (ix) Mil-DDS, and others. The main and dominant implementations are selected in the Table I:

OpenDDS. OpenDDS [6], [7] is an open source C++ implementation of the Object Management Group (OMG) Data Distribution Service (DDS).

sDDS. "sensornetwork" Distributed Data Structures (sDDS) [8] is an approach to make DDS available for very small distributed embedded systems like wireless sensor network.

DDS Community. DDS Community [9] is fully-featured Apache License Version 2.0 Open Source Data Distribution Service (DDS) implementation.

Vortex OpenSplice. Vortex OpenSplice [10] is leading (Commercial and Open Source) implementation of the OMG DDS standard. It focuses on ensuring availability, reliability, safety and integrity in spite of hardware and software failures, by providing high performance - it is capable of distributing large amount of data under low latency, from simple systems to ultra large scale system-of-systems and from smart sensors to high end servers Provides the ability to maintain confidentiality, integrity and authenticity of exchanged data.

Connext DDS. Connext DDS Professional is the leading connectivity framework for demanding IIoT systems. It shares information between devices and applications in real-time, delivering performance, reliability, scalability and security.

CoreDX DDS. CoreDX DDS [11]

is specific implementation designed for embedded applications and single-process architectures. It focuses mainly on performance parameters such as low memory footprint and latency with allowing operation without any operating system (without major limitations). However, compared to the other solutions such as OpenSplice or Connext, the CoreDX is relativelly new implementations. Moreover, the documentation contains just several pages without any closer or deeper information or examples of application.

Eclipse Cyclone DDS. Eclipse Cyclone DDS [12] implements the OMG Data Distribution Service (DDS) specification and the related specifications for interoperability,

InterCOM DDS. InterCOM DDS [13] uses an open standard protocol to effectively network inputs from a wide range of sensors and controllers on a complex platform. It focuses on performance, reliability, modifiability/scalability, availability, and testability.

Mil-DDS. Mil-DDS [14] is a middleware software providing data centric publish-subscribe mechanism for distributed applications. The API is available in multiple languages and platforms, and includes the following capabilities: high-reliability publish and subscribe API, detailed quality of service controls, easy-to-use debug tools, complete reference solutions, intuitive training tutorials, and complete documentation.

IV. Security analysis of selected DDS Implementations

The security analysis of selected DDS implementations is displayed in Table II. We provide information about the DDS security stack and its parameters. The RTI has implemented only the most current algorithms. However, OpenSplice provides several schemes, which helps in providing support for a greater number of scenarios. Finally, OpenDDS implemented the DDS Security stack recently and it is containing only the basic cryptographic algorithms.

TABLE II
Results from security analysis of selected DDS implementations.

Cryptographic primitive	**RTI**	**OpenDDS**	**OpenSplice**
Encryption	aes128gcm, aes192gcm, aes256gcm	Only sub-message and payload encryption is supported.	aes256 - verified GCM & aes128, aes192, aes256, blowfish, rsa-aes128, rsa-192, rsa-256, rsa-blowfish, rsa-null, null.
Authentication	2048-bit RSA, 2048-bit DSA, ECDSA	No origin authentication.	SSL X.509 Certificate Authentication
Auth. Format	X509	X509 for 2048-bit RSA, prime256v1 for 256-bit Elliptic curve.	X509

V. Performance Analysis of selected DDS Implementations

Topology of the laboratory environment is simplified in Figure 5. There are two virtual machines running the implementations separately.

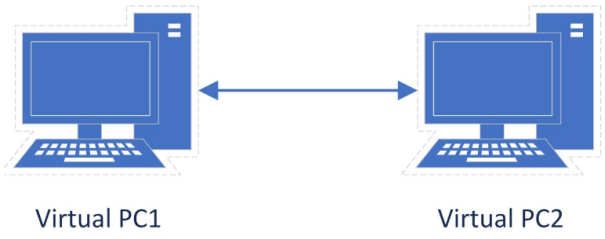

Fig. 5. Architecture of experimental laboratory network.

The first results from performance analysis and measurements are displayed in Figure 6. The figure shows comparison between three domain selected DDS technologies - OpenDDS, Connext and OpenSplice. The higher delay of OpenSplice is given by higher memory requirements. The last results shows average delay (Figure 7) and average jitter (Figure 8

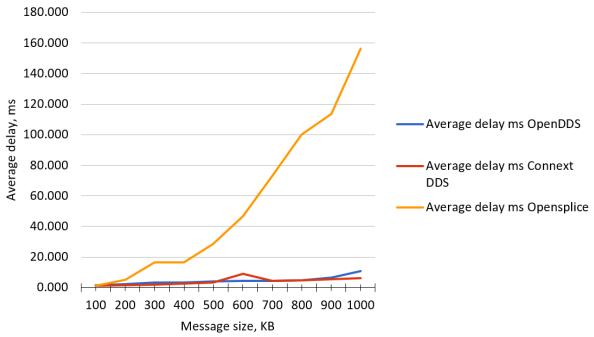

Fig. 6. Benchmark comparison of selected DDS technologies.

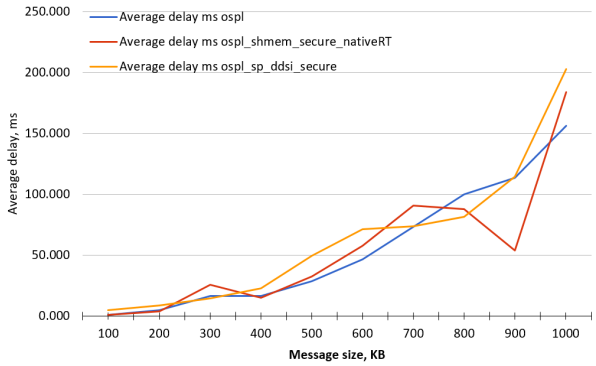

Fig. 7. Jitter of OpenSplice.

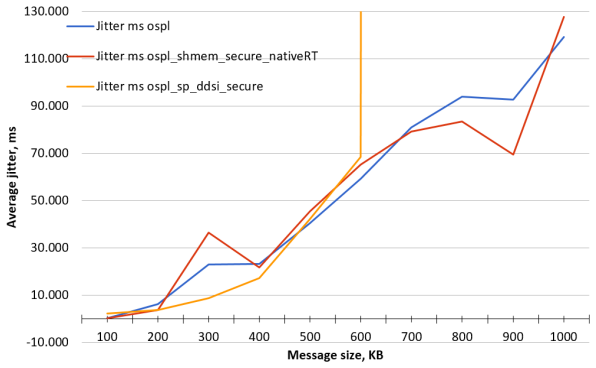

Fig. 8. Delay of OpenSplice.

VI. Conclusion

The paper summarized the data distribution topic together with information given about publish/subscribe logic and architecture. Moreover, we provided an analysis of DDS implementations, based on both security and performance characteristics. The future work should focus on failure scenarios, higher traffic load, different communication technologies, and more complex topology. Moreover, the security analysis might be extended as well. Attack vectors should be investigated over general DDS system and cyber-security risk analysis together with hazard (safety) analysis might be performed as well to give information about possible risks, which would help to build sufficient counter-measures for real systems.

Acknowledgment

This article has received funding within the National Sustainability Program under grant LO1401. Our research and the idea of the paper is coming from the research conducted and supported by research project Aggregated Quality Assurance for Systems (AQUAS H2020-EU.2.1.1.7 ID: 737475). For the research, the infrastructure of the SIX Center was used.

References

[1] Y. Park, D. Chung, D. Min, and E. Choi, "Middleware integration of dds and esb for interconnection between real-time embedded and enterprise systems," in *International Conference on Hybrid Information Technology*. Springer, 2011, pp. 337–344.

[2] O. Othman, O. Carlos, and D. C. Schmidt, "Strategies for corba middleware-based load balancing," *IEEE Distributed Systems Online*, no. 3, p. null, 2001.

[3] Q. H. Mahmoud, *Middleware for communications*. Wiley Online Library, 2004, vol. 73.

[4] G. Pardo-Castellote, B. Farabaugh, and R. Warren, "An introduction to dds and data-centric communications," *RTI, Aug*, vol. 26, 2005.

[5] G. Pardo-Castellote, "Omg data-distribution service: Architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE, 2003, pp. 200–206.

[6] O. Computing, "Inc. opendds developer's guide," 2009.

[7] D. Busch, "Introduction to opendds," 2012.

[8] K. Beckmann and O. Dedi, "sdds: A portable data distribution service implementation for wsn and iot platforms," in *2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES)*. IEEE, 2015, pp. 115–120.

[9] D. OpenSplice, "Adlink opensplice dds community edition."

[10] D. C. Schimidt and A. Corsaro, "Opensplice dds."

[11] R.-T. Innovations, "Rti connext dds professional," 2014.

[12] E. organization", "Eclipse cyclone dds," 2019.

[13] "Kongsberg", "Intercom dds: Real-time networking middleware," 2018.

[14] "MilSOFT", "Mil-dds: Data distribution services middleware," 2019.