Data Distribution Service for Industrial Automation[1]

Jinsong Yang, Kristian Sandström
Industrial Software Systems
ABB Corporate Research
Forskargränd 7
Västerås, Sweden
jyg10001@student.mdh.se
kristian.sandstrom@se.abb.com

Thomas Nolte, Moris Behnam
Mälardalen University
Högskoleplan 1
Västerås, Sweden
thomas.nolte@mdh.se
moris.behnam@mdh.se

Abstract

The IEC 61499 is an open standard for the next generation of distributed control and automation. Data Distribution Service for Real-Time Systems (DDS) is a specification of a publish/subscribe middleware for distributed systems, created by the Object Management Group (OMG) to standardize a data-centric publish-subscribe programming model for distributed systems.

This paper evaluates the DDS communication performance based on a model built within the IEC 61499 standard and compares it with the traditional socket based solution for communication. According to the test results, the DDS communication has the potential to reduce the complexity and is suggested as a suitable solution for some classes of industrial control systems.

1. Introduction

In industrial distributed control systems a substantial number of controllers can be connected through an Ethernet-based control network. The controllers may share control data and also interact with nodes higher in the network hierarchy, like operator stations. In a large system these inter-dependencies will potentially create a complex architecture, especially if explicit point-to-point data connections have to be engineered. Moreover, redundancy schemes, such as hot standby, where software components are duplicated on different controllers, further increase complexity. If real-time performance can be achieved, publish-subscribe models provide some appealing properties that can reduce some of the visible complexity in the software systems and thereby provide more robust industrial automation systems that are easier to engineer and maintain.

When considering new communication strategies for industrial distributed control systems it is justified to consider their applicability to one or several of the standardized programming models, e.g., IEC 61131 or IEC 61499. In this work the focus is on IEC 61499.

The IEC 61499 standard defines an open architecture for the next generation of distributed control and automation. It has now been 10 years since the adoption of the standard, and there are already quite a few 61499-compliant products and tools available on the market from different vendors. IEC 61499 provide a component model where function blocks with well-defined input and output event and data interfaces can be defined. Further, an application can be assembled by connecting the input and output of function blocks. For distributed systems, specific Service Interface Function Blocks (SIFBs) can be used to define communication [1].

DDS is a new enabling technology for the real-time publish-subscribe model that has become more and more popular in applications such as, radar processors, air traffic control and management. Besides commercial usage, it is also used by some key administrations, such as the US Navy, the EuroControl, etc.

In this paper we look at applying DDS in the context of IEC 61499. The contributions presented in the paper are twofold: 1) we present a mapping of node-to-node communication in IEC 61499 to the DDS real-time publish-subscribe model. This also includes mapping of timing requirements to QoS attributes of the publish-subscribe model. 2) Moreover, we evaluate the performance of the publish-subscribe communication and we compare it with the more traditionally used socket-based Ethernet communication.

The outline of this paper is as follows: Section 2 presents the related work on DDS. Section 3 and 4 provide background of IEC 61499 and DDS respectively. Section 5 describes the mapping of IEC 61499 communication to DDS as well as the QoS. Section 6 presents the test setups while the performance evaluation results and analysis are given in Section 7. Finally, in Section 8 the conclusion and future work will be presented.

[1] This work is supported by Industrial Software Systems at ABB Corporate Research, Mälardalen University and The Knowledge Foundation (KKS).

2. Related Work

Douglas C. Schmidt and Hans van't Hag described the key feature of OMG DDS standard and how the standard could handle the challenges of data distribution and management of the next-generation distributed control systems. They also addressed how the OpenSplice DDS was implemented within this standard [2].

Guesmi et al. present an implementation of a publish-subscribe messaging middleware for real-time networking based on the DDS specifications [3]. They introduce an efficient approach of data temporal consistency and use DDS QoS-policies to schedule the network traffic by introducing a real-time network-scheduler. A local scheduling component (EDF scheduler) is used to schedule the access to resources within the node, which is adapted with soft real-time systems using CAN. Rekik and Hasnaoui present a detailed application of CAN BUS transport for a DDS Middleware [4].

Agirre et al. implement an additional layer based on top of the DDS middleware, to build a distributed component management platform for QoS enabled applications [5]. This additional middleware provides 4 kinds of application managements, deployment, application execution control, QoS parameter dynamic reconfiguration and fault tolerance considerations. However, the executable code is made manually, which may not be suitable for remote node software updating.

Thramboulidis et al. present the architecture of a field device that is compliant with the IEC model, a layered approach and a prototype reference implementation [6]. In this approach they propose 3 layers: Application Layer, Industrial Process-Control Protocol (IPCP) layer (CORBA adopted) and Mechanical Process Interface Layer. However, modifications are adopted with the Management Function Blocks and SIFBs. Also CORBA is a distributed object computing platform, while DDS is a data-centric real-time publish/subscribe platform. They are intended for different purpose and DDS simplifies some features of CORBA, for example, no broker is used in DDS.

Calvo et al. present guidelines for how to build communication SIFBs based on the OMG DDS middleware [7]. By using these DDS-SIFBs within IEC 61499 code generation tools, the communication with DDS can be achieved. However, they did not present any performance evaluation of the applicability of the technology in industrial control systems.

Moreover, in this paper we provide a detailed mapping between the IEC 61499 component model and the DDS data model as well as a mapping of real-time requirements to DDS Quality of Service (QoS) attributes.

3. Background on IEC 61499

The IEC 61499 standard provides a generic model for distributed systems. This standard specifies the models of system, device, resource, application, function block, distribution, management and operation state.

Function blocks are the basic units for modeling a distributed system. There are two types of function blocks, basic function blocks and composite function blocks. A composite function block may be composed with one or more other function blocks. A network of function blocks constitutes an application and applications form a distributed system.

In the function block model, the data flow and event flow are separated as every function block has an interface for event I/O and data I/O. Each function block has algorithms that can be triggered by the event and generate data output by calculating the data input. Figure 1 illustrates the structure of function block. Each event of the function block is associated with one or more data.

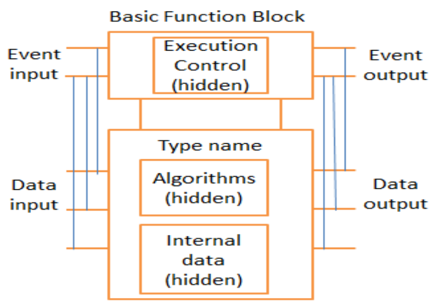

Figure 1. Structure of IEC 61499 basic function block.

IEC 61499 also define some special function blocks, SIFBs, which contain communication SIFBs and management SIFBs. These special function blocks can be implemented using platform specific API's.

4. Background on Data-Distribution Service for Real-Time Systems (DDS)

Data-Distribution Service for Real-Time Systems (DDS) is a specification of a publish/subscribe middleware for distributed systems created by the Object Management Group (OMG). This specification defines a data-centric publish-subscribe programming model for distributed systems. As an emerging technology, it has been of great interest by distributed real-time and embedded systems developers and implemented with both commercial and open source versions by different vendors.

The OMG DDS standard is composed by the DDS Interoperability Wire Protocol (DDSI V2.1) and DDS v1.2 API, which are built on top of the UDP/IP [8]. Figure 2 shows the OMG DDS standard structure.

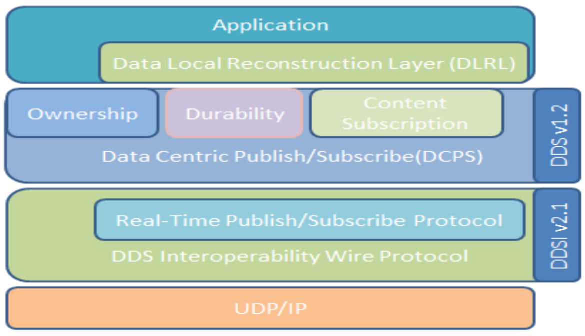
Application
Data Local Reconstruction Layer (DLRL)
Ownership
Durability
Content Subscription
Data Centric Publish/Subscribe(DCPS)
DDS v1.2
Real-Time Publish/Subscribe Protocol
DDS Interoperability Wire Protocol
DDSI v2.1
UDP/IP

Figure 2. OMG DDS standard structure.

The global view of the DDS infrastructure is shown in Figure 3, in which most important DDS entities are involved.

Global Data Space (GDS). The most important abstraction of DDS is the fully distributed GDS, as it stores all the data published in the domain. With GDS, the system can be scalable and avoid single point of failure.

Data writers/readers and publishers/subscribers. DDS uses data writer to publish data into the GDS and uses data reader to read data from the GDS. A publisher is a factory to create and manage data writers while a subscriber is a factory to create and manage data readers.

Topics. A topic contains a unique name, a data type and a set of QoS and it connects data writers and data readers. The data writer and data reader can only communicate if the data reader subscribes to the same topic as the data writer has published. The data type is usually defined by an OMG Interface Definition Language (IDL) structure.

QoS. DDS provides a rich set of QoS, with which users can configure and control the local and end-to-end properties of DDS entities to meet the application requirements.

There are quite a few DDS implementations from different vendors. Among them are Real-Time Innovation (RTI), PrismTech and Twin Oaks, who have implemented DDS and used it for commercial purposes. PrismTech also offers an open source version of their implementation, OpenSplice DDS Community, which will be used in the performance evaluation of this paper.

4.1. Real-Time Publish Subscribe Protocol

The Real-Time Publish Subscribe (RTPS) protocol was approved by the IEC as part of the Real-Time Industrial Ethernet Suite IEC-PAS-62030. It has been used in thousands of industrial applications [9].

Compared with traditional publish subscribe models, more real-time timing parameters and properties are added to this RTPS protocol. RTPS contains two main communication models: the publish-subscribe protocol and the Composite State Transfer (CST) protocol, which are for transferring data and States respectively.

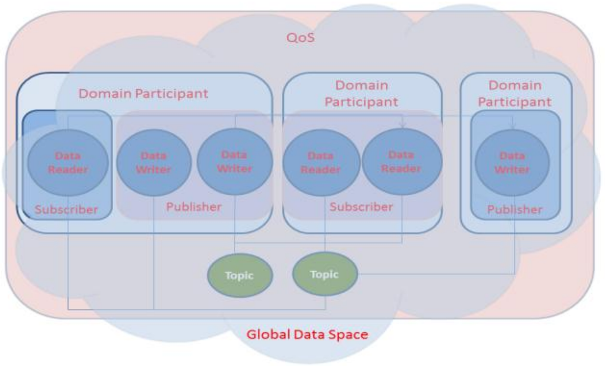

Figure 3. OMG DDS global view.

This protocol is intended for multicasting and connectionless best-effort transports like UDP/IP. The main features of the RTPS protocol include:

- Performance and QoS properties. These properties enable best-effort and reliable publish-subscribe communications for real-time applications. And the QoS makes it more flexible for the user to control the communication according to the requirements.
- Fault tolerance. With this property, the system can be more reliable and avoid single point of failure.
- Plug-and-play connectivity. As the publish-subscribe protocol is connectionless, the applications can join and leave the network at any time.
- Scalability. The system can be scaled to a large network without degrading the performance of the whole network.

These features make RTPS very suitable for a DDS wire-protocol.

5. Mapping

In this section we outline existing solutions for communication in distributed IEC 61499-based systems, and we present how we map RTPS to the IEC 61499 standard.

5.1. Existing Communication Styles

For distributed systems, the architectural styles vary according to different applications. Most of them can be viewed as independent components, using communication styles such as client-server, publish/subscribe, peer to peer, or service oriented. In some cases, shared memory is used, e.g., in blackboard, whiteboard, and databases [10].

Concerning the communication styles of distributed control systems, there are lots of options. The interaction could be based on e.g., procedure/method call, message passing, streaming, or events.

The IEC 61499 standard defines a model of devices, resources and applications. The data and event communication between applications can be achieved via

SIFBs specialized for this purpose. To implement this SIFB, the communication styles mentioned above can be utilized.

For applications located on different nodes in a network, the client/server socket communication style is a common choice. Associated to an IP address and port number, socket communication is used in the transport layer of the internet model and can be implemented based on either the TCP or the UDP protocol, of which the difference is that UDP is connectionless.

For applications that are running on the same node of a network, shared memory communication is a typical solution. With a relatively low latency and high bandwidth, shared memory is an efficient means of data communication between applications.

5.2. Mapping RTPS to IEC 61499

The IEC 61499 standard provides a generic model for distributed systems. In order to use the Real-Time Publish/Subscribe protocol in applications based on this standard, a mapping between the IEC 61499 model and the RTPS concept is proposed.

IEC 61499 applications are built by networks of function blocks. The communication between applications from different nodes can be achieved via communication SIFBs. A typical IEC 61499 model can be configured as Figure 4.

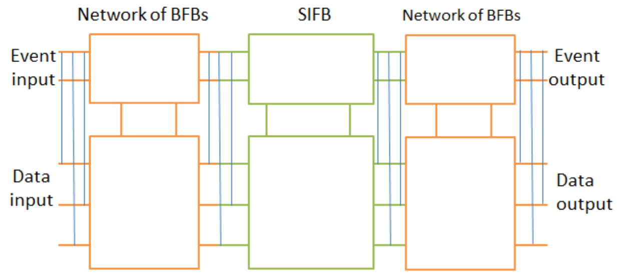

Figure 4. Typical IEC 61499 application model. (BFB: basic function block.)

The Real-Time Publish/Subscribe protocol is proposed to be implemented within the SIFB of the IEC 61499 standard. In this case, the SIFB in Figure 4 is replaced by the Real-Time Publish/Subscribe SIFBs and global data space, as shown in Figure 5.

In the RTPS SIFB, the events are realized by creating publisher/subscriber, reading/writing message, setting QoS, etc. The data input is the message that needs to be sent. The publisher and subscriber are not directly connected, due to the DDS connectionless feature.

5.3. Mapping the IEC 61499 Interface Declaration to the DDS Topic Declaration

The IEC 61499 function block inputs are translated to DDS topic types as illustrated by the example in Figure 5. Each input event is translated to a DDS topic name and the data inputs associated with the event are declared as the topic type (message type). This mapping will transfer only that data that is needed for each event across the network.

In Figure 5, data inputs associated to EventA and EventB are registered as topic types EventA and EventB respectively. The topic types in DDS are declared as an OMG Interface Definition Language (IDL) structure. DInsA and DInsB are the message instances of topic types EventA and EventB, which during the communication will be published by Publisher and subscribed by Subscribers.

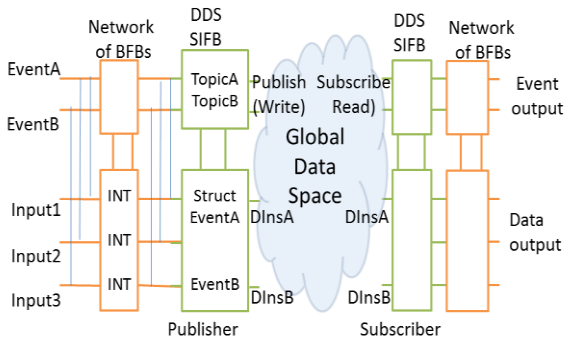

Figure 5. Mapping event and data inputs into DDS topic.

The function block model in Figure 5 can be structured as:

```
FUNCTION_BLOCK my_SIFB
EVENT_INPUT
  EventA : EVENT WITH Input1, Input2;
  EventB : EVENT WITH Input3;
END_EVENT
VAR_INPUT
  Input1 : INT;
  Input2 : INT;
  Input3 : BYTE;
END_VAR
```

The following two OMG IDL structures are mapped from the model above. If defined the keylist, the topic can be multiple instances. Different keylists specify different instances and a specified instance can be modified by the data writer. If the keylist is not defined, then there can only be one instance for the topic.

```
struct EventA {
int tID;
integer Input1;
integer Input2:};
#pragma keylist EventA

struct EventB {
int tID;
char Input3;};
#pragma keylist EventB
```

5.4. Mapping Temporal Requirements on IEC 61499 Communication to DDS QoS

As an industrial automation standard, systems based on IEC 61499 usually require real-time performance. The

DDS QoS can be utilized to fulfill these real-time requirements. With lots of content in the DDS QoS, we are particularly interested in the transport priority, deadline and latency budget QoS.

The transport priority in DDS is defined as a long integer and thus an algorithm can be defined to decide the transport priorities. We assume that each event, *e*, that is transferred between two nodes, have an associated latency requirement, $\mathrm{Latency}^e$. We define a policy as the smaller the latency, the higher the priority. If we assign the highest transport priority as the maximum value of a long integer, we have the following formula to determine the priority of the underlying transport.

$$\mathrm{TransportPriority} = \mathrm{MaxPrio} - \mathrm{Latency}^e$$

Assuming that a long integer is 32-bit and $\mathrm{Latency}^e$ is at microsecond level, then, there can be more than 73 million different priorities.

For industrial automation systems, most messages need to be delivered to a specific node at a specific time, such as sampling signals of sensors. Therefore the deadline QoS can be introduced to monitor the delivery of the messages. If assigned a deadline, the entity could be notified when deadline is missed.

In order to maximize the throughput and minimize the resource usage, the latency budget QoS is introduced. This allows DDS to optimize throughput. In the test we compare the performance when the latency budget QoS is set to 0 and a certain value. When the latency budget QoS is set to zero, it minimizes the latency at the cost of total throughput. And when it is set to a certain value, it could relax the budget and improve the throughput.

6. Test Setup

In the test the differences with respect to performance of socket communication and DDS communication are evaluated and compared. The main objective of this evaluation is to measure the latency of the communication in several different scenarios.

All the tests were done on a network of desktops all running Ubuntu 11.10. Each of the Ubuntu OS is patched with real-time package RT-29 to construct real-time Linux environments. All the Linux nodes are connected to a switch to realize Ethernet communication. The Ethernet bandwidth used is 1 Gbps for each machine.

There are quite a few DDS implementations available. However, in the tests the open-source version OpenSplice DDS Community 5.4.1 from PrismTech is used. Although there are a few IEC 61499 compliant IDEs available, they have not been used due to the lack of documentation. Instead the function blocks are implemented with C++ and the GCC compiler is used to compile the code.

6.1. Network without Extra Network Load

The first scenario is made to evaluate the performance of the DDS in a network.

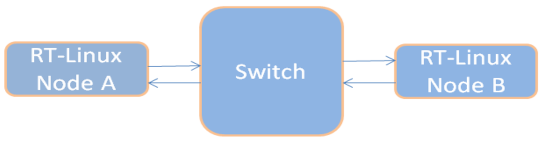

Figure 6. Two-node network with switch.

Two desktop machines are connected with a switch. Figure 6 shows the connection.

For the implementation, we meant to follow the IEC 61499 standard to model a real-time distributed industry automation system. The communication interface between the nodes in the system is via the communication SIFB implemented with DDS. Thus a class named iec2dds is implemented to represent the DDS communication SIFB. These public members and functions of the class are declared as:

```
Class iec2dds
{Public:
        TopicName          topicName;
        TopicQos           Qos;
Public:
bool createPublisher(char* topicName, TopicQos tQos);
bool createSubscriber(char* topicName,TopicQos tQos);
bool read(Msg *dataIn);
bool write(Msg dataOut);
}
```

In order to make less DDS SIFB interfaces, the methods createPulisher() and createSubscriber() actually create all the entities needed for DDS as internal variables and also set the related QoS. To make the QoS of the topic, publisher and writer consistent, the creatPublisher() function has the parameter topicName and will set QoS of publisher and writer. createSubscriber() function did the same to the Subscriber and reader. Write() and Read() represent the interfaces to write and read data. An application here can have both a publisher and a subscriber. This class implementation follows the mapping in Section 5.

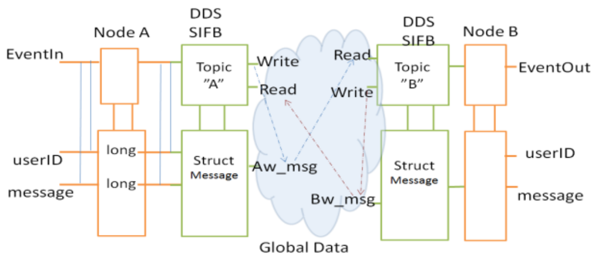

Figure 7. DDS Network communication.

Figure 7 explains the setup. The data input from Node A contains a userID and a message, so they are wrapped as an OMG IDL structure, which serves as the topic type. The dashed arrows beginning from Node A form a round trip of the message. To make things easier, all the Round Trip Times (RTTs) are measured on Node A, that is, put time stamp preReadTime before the message write and postReadTime stamp after the message read function in the resolution of microsecond. So a RTT is calculated by the following formula:

$$RoundTripTime = postReadTime - preWriteTime$$

Node B just forwards the received message without doing anything else, thus the overhead is minimized.

The IDL structure Message is defined as:

```
struct Message{
        long userID;
        long message[LENGTH];}
```

This IDL structure serves as the topic type and in the test the publishers and subscribers are associated to different instances of the topic type. Here the input message is defined as an array because in the test we scale the message frame to check if the performance varies with different message size.

A client-server communication is put up between the two machines to compare with DDS. The same message structure as above is used in the socket communication.

For modelling and enforcing real-time requirements, the DDS QoS is an important feature. DDS provides a large set of QoS, in the test we only consider the reliability, latency budget and transport priority QoS. DDS communication performance is measured for different QoS configurations.

6.2. Network with Network Load

In Figure 8 two additional nodes (C and D) are added to generate extra network loads. All these nodes work with 1 Gbps Ethernet. Node A and B remain the same as Figure 7 shows, but Node C and D are generating network traffic between themselves. This setup is intended to evaluate the scalability and stability of the DDS system.

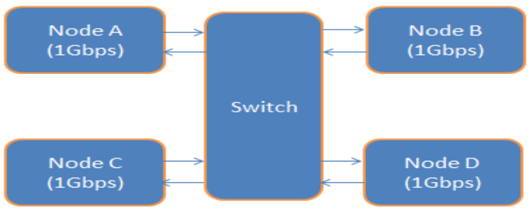

Figure 8. Four-node network with switch.

7. Results and Analysis

This section discusses the result of all the tests. During the experiments the main metrics measured are latency, jitter and worst case latency. Each of the metrics is given by average result of 1.000.000 iterations.

Latency is the elapsed time that a message is sent by a node until it is received by another node. The factors might affect the transmitting of the messages are the transmitting protocol, Ethernet bandwidth limitations and other traffic on the network. There might also be some overhead related to the implementation of the communication primitives.

Jitter is the variation in round-trip time between two consecutive message transfers. In the work presented in this paper, the average and maximum jitter is measured, and the distribution is calculated.

The worst case latency is defined to be the largest latency measured in 1.000.000 samples.

7.1. Result with Two Nodes

Table 1 shows the RTTs and latencies of the message transfer between the two nodes with socket communication and DDS communication respectively. The DDS is configured with reliable QoS setting and utilizes waitset (a DDS event handling mechanism) to read messages.

Message (bytes)		8	16	32	64	128	256
DDS (µs)	RTT	294	302	302	313	315	318
	Latency	147	151	151	156.5	157.5	159
Socket (µs)	RTT	162	162	163	162	164	169
	Latency	81	81	81.5	81	82	84.5
Message (bytes)		512	1024	2048	4096	8192	16384
DDS (µs)	RTT	322	365	411	767	1475	2930
	Latency	161	182.5	205.5	383.5	737.5	1465
Socket (µs)	RTT	170	196	236	290	362	490
	Latency	85	98	118	145	181	245

Table 1. Round trip times and latencies of reliable DDS and socket communication.

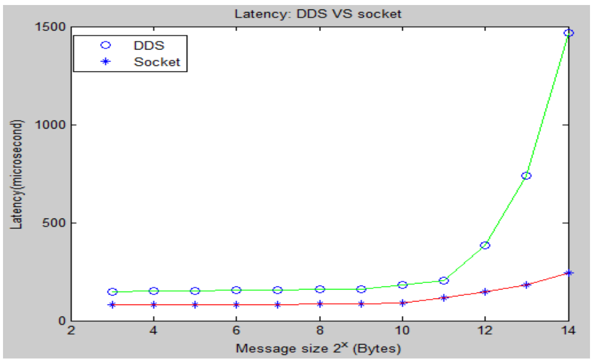

Figure 9. Latencies of reliable DDS and socket communication (Note: lines connect points are only for illustration).

As described in the setup in Section 6, we measure the RTT for sending and receiving one message and then we can calculate the latency of each communication style by the following simple formula:

$$Latency = R/2$$

Figure 9 gives the latencies of different message sizes. The latencies of both DDS and socket communication do not change much when sending messages in the range of 2^3 to 2^{10} bytes, since they fit in a single Ethernet frame. When the messages become larger, the increase in round trip latency is evident and moreover, DDS is more sensitive to increased message size than socket communication is.

Considering at the size of 1024 bytes per each message, the DDS takes 182.5 µs to send one message, while the socket communication takes 98 µs, only half time of the DDS. Although considerable slower than socket based communication, DDS still can send an average of 5479 messages a second, which will meet the requirements of some industrial applications.

In the second setup, two sets of QoS are introduced in the DDS communication. Table 2 shows the different latencies when reliable QoS and best-effort QoS is set respectively.

Message size (bytes)	8	16	32	64	128	256
Reliable QoS Latency (µs)	147	151	151	156.5	157.5	159
Best-effort Latency (µs)	140	140.5	142	143.5	147.5	150
Message (bytes)	512	1024	2048	4096	8192	16384
Reliable QoS Latency (µs)	161	182.5	205.5	383.5	737.5	1465
Best-effort Latency (µs)	157	175	195.5	377	734	1463

Table 2. Latencies of reliable and best-effort DDS communication.

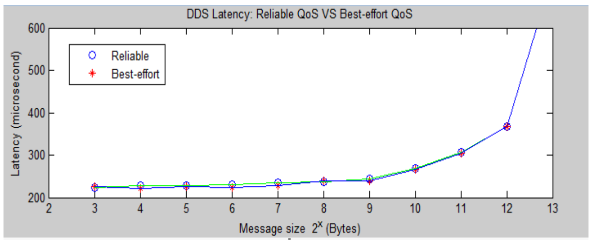

Figure 10. DDS latencies of reliable QoS and best-effort QoS Communication. (Note: lines connect points are only used for illustration).

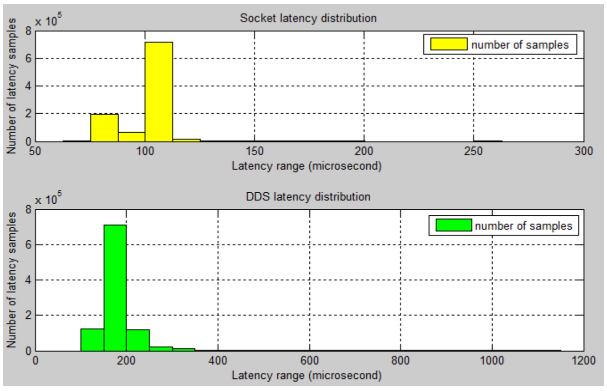

Figure 11. DDS and socket latency samples distribution.

Figure 10 illustrates the latency comparison when the two sets of QoS are used. Although not so obvious, we can still see that the best-effort QoS set has lower latency. That is because the mechanism of reliable reliability QoS would cause more overhead.

Figure 11 shows the distribution of measured latency for DDS and socket communication of 1024-byte messages based on 1.000.000 samples.

However the Socket communication latency mostly ranges from 62.5 to 112.5 µs while DDS latency mostly ranges from 50 to 400 µs.

The standard deviation of the DDS latency is 109 µs, which is 59.8% of the average latency 183 µs. The socket latency standard deviation is 10.184 µs, 10.3% of the average latency 99 µs. Hence, the socket communication introduces less variation in message transmission times than DDS.

Table 3 shows the RTT, jitter and standard deviation of DDS communication with different setting for latency budget QoS when messages with 1024 bytes are sent. As from the previous test, the average RTT of sending 1024-byte message is no more than 400 µs, here we set the deadline QoS value to 200 µs and the latency budget to 100 µs.

When setting the latency budget to 100 µs, the average RTT, average jitter and standard deviation of the RTT are all larger than that of when the latency budget is set to zero. This is a trade-off between throughput and latency, smaller message and lower latency budget can improve the latency, but they also cause low throughput.

Message (1024 bytes)	RTT (µs)		Jitter (µs)		Standard deviation (of RTT)
	Average	Worst-case	Average	Max	
Latency budget=0	365	11983	76.152	11531	202.396
Latency budget=100	407	11431	87.214	11025	227.337

Table 3. Round trip time, jitter and standard deviation of reliable DDS communication with latency budget 0 and 100 µs.

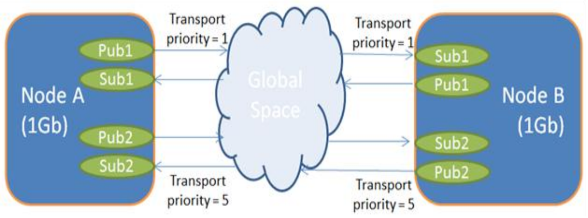

Figure 12. Publishers and subscribers with different transport priorities.

Topics	TP1=TP2=1	TP1=1, TP2=5	
Latency(Topic1)	520	534	-
Latency(Topic2)	519	-	500

Table 4. Latency with different transport priorities.

Table 4 shows that when two topics in one node are assigned the same transport priority, the average latencies for the two topics are almost the same. But when Topic2 is assigned a higher transport priority than Topic1 (as shown in Figure 12), the average latency of Topic2 would be smaller than that of Topic1. Transport priority QoS gives users the flexibility to decide which topic or messages should be delivered first.

7.2. Result with Four Nodes

In this experiment, two additional nodes C and D are introduced into the network to generate network load and evaluate the scalability of DDS. These two are also with one Gbps Ethernet and sending the message with the same size 1024 bytes in both DDS and socket communication. According to the statistic by the software IPTraf on Node C, the total extra network load added is 94.242 Mbps for socket and 96.902 Mbps for DDS. Table 5 shows the latency and jitter of DDS and socket communication transferring 1024-byte messages with and without extra network load.

Message (1024 bytes)	RTT (µs)		Jitter (µs)		Standard deviation (of RTT)
	Average	Worst-case	Average	Max	
DDS (without load)	353	11719	58.047	11722	166.836
DDS (with load)	348	11641	59.863	11290	167.643
Socket (without load)	195	364	12.465	180	16.600
Socket (with load)	222	424	16.016	211	24.739

Table 5. RTT, jitter and standard deviation of reliable DDS and socket communication with and without extra load.

As can be seen in Table 5 the DDS round-trip time is largely unaffected by the increased network load. In contrast, the performance for the socket based communication is significantly reduced, especially the average latency and jitter.

Still, the socket based communication performs better than DDS, but the measurements indicate that the difference in performance is likely less in a heavily loaded distributed system.

8. Conclusions and Future Work

Real-time publisher-subscriber mechanisms have the potential to reduce the complexity of large scale industrial control systems by removing the need of engineering explicit point-to-point connections between large numbers of components. In this paper we show how a programming model for industrial control systems, IEC 61499, can be mapped on to the DDS publisher-subscriber model. The mapping considers the interface of IEC 61499 function blocks, mapping events and input variables to DDS data topics. Moreover, latency requirements on the IEC 61499 point-to-point connections are mapped to DDS QoS attributes for the specific topic that represent the connection. Finally, an initial performance evaluation of DDS is presented and compared with a socket based point-to-point communication scheme.

Although the DDS communication is slower than the socket communication, it is of great scalability and can reduce the complexity of the connections. In addition, the rich set of QoS can help users better control the communication. The evaluation presented here indicates that DDS could be used for a certain class of industrial control systems or for parts of industrial control systems.

In future work we will apply DDS to a distributed control systems and check the performance in industrial applications.

References

[1] International Electro-technical Commission, *International standard IEC 61499-1*, first edition, IEC, 2005.

[2] Douglas C. Schmidt and Hans van't Hag, "Addressing the Challenges of Mission-Critical Information Management in Next-Generation Net-Centric Pub/Sub Systems with OpenSplice DDS", *IEEE International Symposium on Parallel and Distributed Processing (IPDPS),* April 14-18, 2008, pp. 1-8.

[3] Tarek Guesmi, Rojdi Rekik, Salem Hasnaoui and Houria Rezig, "Design and Performance of DDS-based Middleware for Real-Time Control Systems", *International Journal of Computer Science and Network Security*, Vol.7, No.12, pp. 188-200, December 2007.

[4] Rojdi Rekik and Salem Hasnaoui, "Application of a CAN BUS transport for DDS Middleware", *2nd International Conference on the Application of Digital Information and Web Technologies (ICADIWT)*, August 4-6, 2009, pp. 766-771.

[5] Aitor Agirre, Marga Marcos and Elisabet Estévez, "Distributed component management platform for QoS enabled applications", *16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA),* September 5-9, 2011, pp. 1 – 4.

[6] Kleanthis C. Thramboulidiset, George S. Doukas and Tassos G. Psegiannakis, "An IEC-Compliant Field Device Model for Distributed Control Applications", *2nd IEEE International Conference on Industrial Informatics (INDIN), June 24-26, 2004*, pp. 277 – 282.

[7] Isidro Calvo, Federico Pérez, Ismael Etxeberria and Guadalupe Morán, "Control communications with DDS using IEC61499 Service Interface Function Blocks", *IEEE Conference on Emerging Technologies and Factory Automation (ETFA),* September 13-16, 2010, pp. 1 – 4.

[8] Object Management Group, *Data Distribution Service for Real-time Systems Version 1.2*, OMG, 2007.

[9] Object Management Group (OMG), *The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification*, OMG, 2009.

[10] Oliver Vogel, Ingo Arnold, Arif Chughtai and Timo Kehrer, *Software Architecture: A Comprehensive Framework and Guide for Practitioners*, Springer, 2011.