MÄLARDALEN UNIVERSITY
SWEDEN

School of Innovation, Design and Engineering

MASTER THESIS IN
INTELLIGENT EMBEDDED SYSTEMS
30 CREDITS, ADVANCED LEVEL

Data Distribution Service for Industrial Automation

Author: Jinsong Yang
Carried out at: ABB Corporate Research Centre

Advisor at ABB: Kristian Sandström

Advisor at MDH: Moris Behnam

Examiner: Thomas Nolte

Date: 1 September 2012

ABSTRACT

In industrial automation systems, there is usually large volume of data which needs to be delivered to right places at the right time. In addition, large number of nodes in the automation systems are usually distributed which increases the complexity that there needs to be more point-to-point Ethernet-connections in the network. Hence, it is necessary to apply data-centric design and reduce the connection complexity. Data Distributed Service for Real-Time Systems (DDS) is a data-centric middleware specification adopted by Object Management Group (OMG). It uses the Real-Time Publish-Subscribe protocol as its wiring protocol and targets for mission- and business-critical systems. The IEC 61499 Standard defines an open architecture for the next generation of distributed control and automation systems. This thesis presents the structure and key features of DDS and builds a model of real-time distributed system based on the IEC 61499 Standard. Then a performance evaluation of the DDS communication based on this model is carried out. The traditional socket-based communication is also evaluated to act as a reference for the DDS communication. The results of the evaluation mostly show that DDS is considered as a good solution to reduce the complexity of the Ethernet connections in distributed systems and can be applied to some classes of industrial automation systems.

PREFACE

This project would not have been possible without the support of many people. First I would like to thank my supervisor Kristian Sandström at ABB Corporate Research Center, without whose research interest this project would not have been even started. I also want to express my gratitude to my examiner Thomas Nolte, with whom we had lots of interesting discussion and he gave me a lot of constructive advice. Gratitude is also due to my advisor Moris Behnam at Mälardalen University, who gave me a lot of useful advice during the writing process. Thanks to their efforts, we have the chance to present the results of the project in the IEEE Conference on Emerging Technologies and Factory Automation (ETFA) in Poland on September, 2012.

Deepest gratitude is due to Damir Isovic from Mälardalen University and Huifeng Wang from East China University of Science and Technology, who made my two years' study in Sweden possible and wonderful. I would also convey my thanks to my Supervisor Hongbo Shi from East China University of Science and Technology, who has always supported and encouraged me to make progress in my study.

I would like to express my love and gratitude to all my beloved families, for their understanding and endless love through the duration of my studies.

Special thanks also to all the colleagues at Mälardalen University, at East China University of Technology and Science and at ABB Corporate Research Center, for their invaluable assistance.

Västerås, August 2012

Jinsong Yang

NOTATIONS

Symbol	**Explanation**
OMG	Object Management Group
IEC	International Electrotechnical Commission
DDS	Data Distribution Service for Real-Time Systems
GDS	Global Data Space
RTPS	Real-Time Publish-Subscribe
SIFB	Service Interface Function Block
DDSI	Data Distribution Service Interoperability protocol
DCPS	Data Centric Publish-Subscribe
DLRL	Data-Local Reconstruction Layer
QoS	Quality of Service
RTT	Round Trip Time
JMS	Java Messaging Service
IDL	Interface Definition Language
PIM	Platform Independent Model
PSM	Platform Specific Model

CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction

In large scale industrial automation and control systems, a large number of controllers are usually distributed and connected with each other by an Ethernet-based network. The nodes in the network may share control data and interact with each other from different network hierarchy, including higher level operator stations. The inter-dependencies between these nodes may potentially create a complex architecture of the network in the distributed system especially if the point-to-point connection needs to be established. Moreover, most industrial automation systems need to be engineered with redundancy schemes, such as hot standby. The software components are duplicated on different nodes when the hot standby is used, which further increases the complexity of the network connections in the system. Publish-subscribe model shows some appealing properties [1], such as connectionless and multicast, that can be used to reduce some of the visible complexity in the software systems.

DDS [2] is an open specification of data-centric middleware defined by OMG to standardize the real-time publish subscribe communication model. This specification is designed to address the mission- and business- critical systems and has been widely used in both commercial and administrative field, including US Navy, EuroControl, etc. Some properties of DDS suggest the possibility to be used in the communication of the industrial distributed control systems.

It is essential to consider the applicability when introducing a new communication strategy in industrial distributed control systems. Communication strategy that is applicable to an international standard could be widely adopted by different vendors and users. IEC 61131 [3] and 61499 [4] are international standards for industrial automation. This work will focus on IEC 61499.

The IEC 61499 Standard defines an open architecture for the next generation of distributed control and automation systems. This standard provides component model with the basic unit function block, which has separate event and data interfaces. Based on function block, a system can be assembled by network of function blocks. IEC 61499 also defines some special function blocks to provide special services. For example, communication Service Interface Function Block (SIFB) is defined to achieve communication between different applications [4]. IEC 61499 is already used in quite a few products and there are some tools to develop IEC 61499 compliant applications.

This work would focus on apply DDS to an IEC 61499 compliant model.

1.2 Objective

This work focuses on applying DDS in the context of IEC 61499. The specific objectives are to: 1) present the structure and key feature of DDS, 2) map node-to-node communication in IEC 61499 to the DDS real-time publish-subscribe model including mapping timing requirements to QoS attributes of the publish-subscribe model, and 3) evaluate the performance of DDS communication while comparing it with the more traditional socket-based Ethernet communication.

1.3 Problem Formulation

As the industrial distributed control systems are usually large systems consisted of thousands of nodes deployed from different location, the communications between the nodes in the systems could be very complicated. For example the traditional socket-based Ethernet communication requires point-to-point connection, which is complex and the complexity could raise deployment risk during integration. It would be very interesting if a solution can be found to simplify the connections and at the same time maintain the performance of the systems.

Therefore, this thesis focuses on exploring the communication styles and tries to evaluate and find a solution that can simplify the communication and at the same time maintain the performance of the system.

1.4 Outlines

The outline of this thesis is as follows: Chapter 2 describes the background of DDS and IEC61499 Standard. Chapter 3 presents some related work. Chapter 4 explains the mapping of IEC 61499 communication to DDS as well as the QoS. Chapter 5 explains the test setups and the performance evaluation is given in Chapter 6. In Chapter 7, conclusions will be drawn and Chapter 8 discusses about the future work.

Chapter 2

BACKGROUND

This Chapter gives the background of IEC 61499 Standard and RTPS protocol. As a new enabling technology, DDS is introduced in detail including: structure, core entities, key features and footprint.

2.1 Background on IEC61499 Standard

As an international standard for industrial control and automation systems, IEC 61499 is consisted of 4 parts. Part 1 is about architecture, which defines the models and provides the declaration, configuration, usage and management of function blocks. Part 2 is about software tool requirements, which specifies the requirements of software to support IEC 61499 Standard. Part 3 gives some tutorial information to help users understand and use the standard by examples. Part 4 defines the rules for the compliance profile like the interoperability and portability [4].

The IEC 61499 Standard provides reference models for systems, devices, resources, applications, function blocks, distribution, managements and operation states.

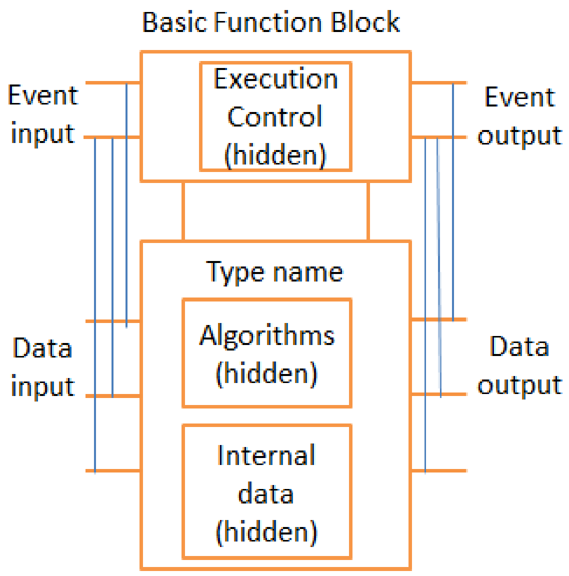

Figure 1. Structure of IEC 61499 Basic Function Block (see [4])

Function block is the key definition of IEC61499 Standard, as it is the basic unit of a distributed control system. Function block is defined to be a functional unit of software which can be connected to other function blocks to make up an application. IEC 61499 defines two types of function blocks, basic function block and composite function block. A composite function block is consisted of several other function blocks. Network of function blocks can be assembled to be an application and applications can be assembled to a distributed system.

Figure 1 shows the structure of basic function block. The function block separates event and data inputs. When the function blocks are assembled to application, the application can have separate event and data flow. However, each event is associated with none or more data. Each function block has algorithms that can be triggered by the events and are used to compute the output based on the input. These algorithms can be defined by Structural Text, C++ or Java, depending on which development tool is used.

Some special function blocks – Service Interface Function Blocks (SIFBs) are defined to provide services. A SIFB is a function block that provides services to an application, which is based on a mapping of service to the function block's event and data input/output. There are two types of SIFB: the communications SIFBs are used to define communication while the management SIFBs are defined for management of applications and behaviours of functions.

There are already some IEC 61499 compliant development environments, like Function Block Development Kit (FBDK) [5], FBench [6], 4DIAC [7]. Most of them provide development of function blocks with separate event- and data-flow in C++ or Java and programming of algorithms using C++, Java or Structural Text. However, they still need more documentation and flexibility about these tools.

2.2 Real-Time Publish-Subscribe Protocol

This section talks about the existing communication styles and addresses the strength of Real-Time Publish Subscribe Protocol.

2.2.1 Existing Communication Styles

In distributed systems, there are different communication styles according to different purpose and requirements. Most of the distributed systems can be viewed as independent components but interact with each other using communication styles like client-server, publish-subscribe, peer to peer, service oriented style, etc. [8].

Client-server style [9]. Information for the client-server style is usually managed and exchanged centrally. The communication is usually achieved by request from the client to the central server with explicit connection. Figure 2 shows the connection of client-server style.

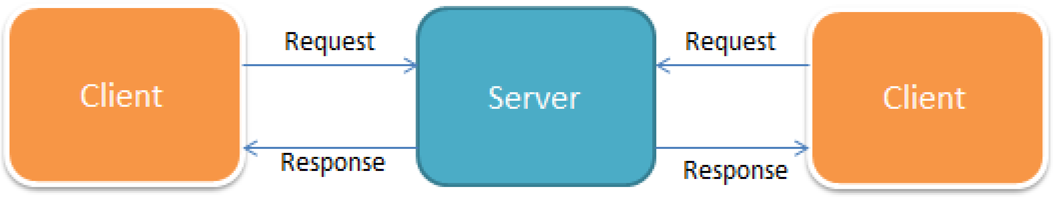

Figure 2. Client-server style

Peer-to-peer style [9]. Different from client-server style, peer-to-peer establishes connection between clients without central server. Every node in the network can offer and consume information. In addition, any node can join or leave the network at any time. Since no central server used, it avoids single point of failure. Figure 3 shows the connection of pure peer-to-peer style. There is also some combination between peer-to-peer network and client-server.

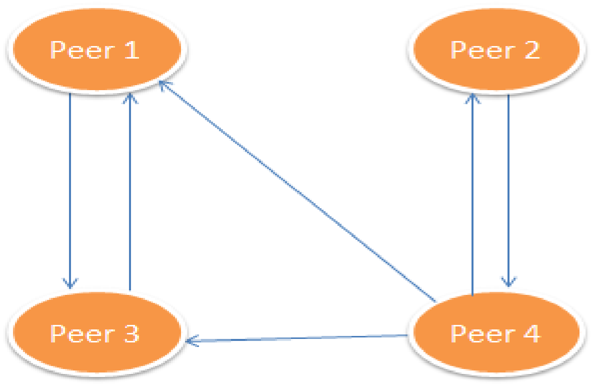

Figure 3. Pure peer-to-peer style

Publish-subscribe style [9]. The styles above are both based on explicit connections. Alternatively publish-subscribe style uses an implicit invocation mechanism which is a many-to-many network. The nodes (publishers/subscribers) are associated to each other by run-time events (topics) and this is called subscription. Many publish-subscribe systems use an intermediate broker to realize this subscription. The communication between associated publishers and subscribers can happen when this event takes place. Figure 4 shows the structure of the publish-subscribe style.

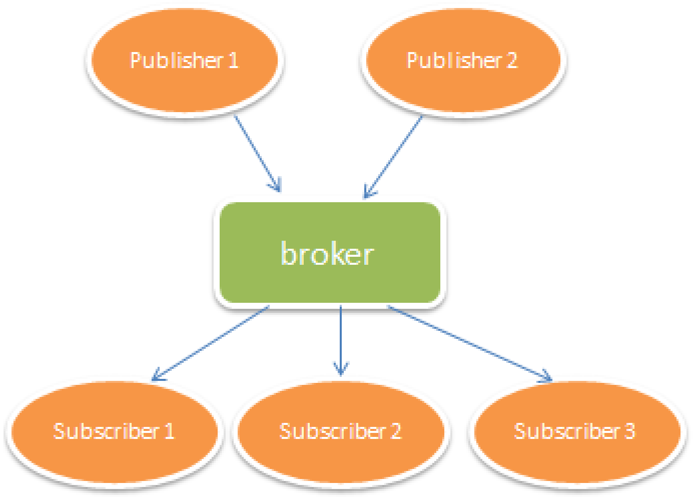

Figure 4. Publish-subscribe with broker

All of these communication styles have their own pros and cons when applied to a specific field. Sometimes a combination between them could be a fair solution for a particular problem. However client/server based socket communication is the most common choice in industry; especially the TCP based socket communication, which can establish a reliable communication.

2.2.2 Real-Time Publish Subscribe Protocol

The Real-Time Publish Subscribe (RTPS) protocol [10] is a part of the Real-Time Industrial Ethernet Suite IEC-PAS-62030 approved by IEC. This protocol has been widely used in thousands of industrial applications. The RTPS Wire protocol is mainly composed with the publish-subscribe protocol and Composite State Transfer (CST) protocol. The publish-subscribe protocol transfers data, while the CST transfers state.

The RTPS protocol aims to run over connectionless best-effort transport with support of multicast such as UDP/IP. It supports the unique requirements of data-distribution systems. Compared with traditional publish-subscribe, there are more real-time requirements on it. It is specified in both a Platform Independent Model (PIM) and a set of Platform Specific Model (PSM), which indicates that RTPS can run on different transport platform. The main features of RTPS are listed below:

• *Performance and QoS properties*: RTPS properties support RTPS to be run on both reliable and best-effort communications. The QoS policies give the users the flexibility to specify and control the communication behaviours.

• *Fault tolerance*: This property ensures the robustness of the system and avoids single points of failure.

• *Extensibility*: Some parts of the RTPS protocol can be extended.

• *Plug-and-play connectivity*: All the nodes can join or leave the network at any place and time.

• *Scalability*: Due to the connectionless and dynamic discovery properties, the system can easily be scaled to large number of nodes.

These properties of RTPS perfectly meet the requirements of DDS wire-protocol [10].

2.3 Background on DDS

DDS, as a publish-subscribe middleware standard was approved by OMG in 2003 and in the past a few years, it has been improved and enhanced. Nowadays, it has already been used in both commercial and administrative applications. DDS defines a programming model of RTPS protocol for distributed systems and has been implemented by several different vendors according to different applications.

2.3.1 DDS Standard

The OMG DDS defines a programming model including a wire protocol and a set of standard APIs. This standard includes the DDS Interoperability Wire Protocol [10] and DDS API [2]. Figure 5 shows the structure OMG DDS standard. The current working version of DDSI is 2.1 and the DDS API version is 1.2. However, DDS API Version 1.3 is now under revision [11].

The DDS Interoperability Wire Protocol Specification (DDSI) defines the interoperability protocol for DDS. DDS uses RTPS protocol as an underlying data transport protocol and all vendors of DDS follow this wire protocol, which ensures that different vendors' implementations of DDS can interoperate.

The DDS API standard provides the standard interface between DDS and applications. This standard ensures the source code portability between different vendor implementations. Currently OMG provides the standard DDS API interface in C, C++ and Java languages.

Lower level Data-Centric Publish-Subscribe (DCPS) is intended for efficient delivery of information. DCPS minimizes the need for data copy and dynamic resource allocation, which is predictable and efficient. Actually this layer defines all the activities for DDS communication, for example, defining topics, creating publisher/subscriber entities, writing/reading data, etc.

Optional higher-level Data-Local Reconstruction Layer (DLRL) provides more direct access to the exchanged data and simpler integration with the local language constructs. NDDS from Real-Time Innovations [12] and Splice from Thales Naval Nederland [13] are commercial implementations implemented not only the DCPS, but partially DLRL as well, though no standardized interfaces are provided [10].

The light-weighted UDP/IP is used as a transport for DDS as it has the following characteristics: universal availability, best-effort, connectionless, predictable behaviour, scalability and multicast support. Although DDS can also be implemented using other transport platform, it is supposed to work best over UDP/IP [10].

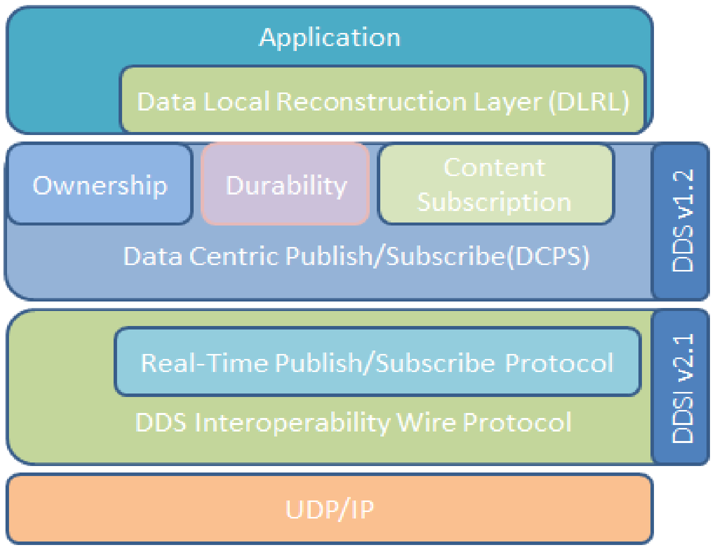
Application
Data Local Reconstruction Layer (DLRL)
Ownership
Durability
Content Subscription
Data Centric Publish/Subscribe(DCPS)
DDS v1.2
Real-Time Publish/Subscribe Protocol
DDS Interoperability Wire Protocol
DDSI v2.1
UDP/IP

Figure 5. OMG DDS standard structure

2.3.2 Core Entities of DDS

Figure 6 shows the global view of a DDS system with 3 domain participants and two topics. It shows almost all the important entities of DDS, such as data writer, data reader, publisher, subscriber, etc.

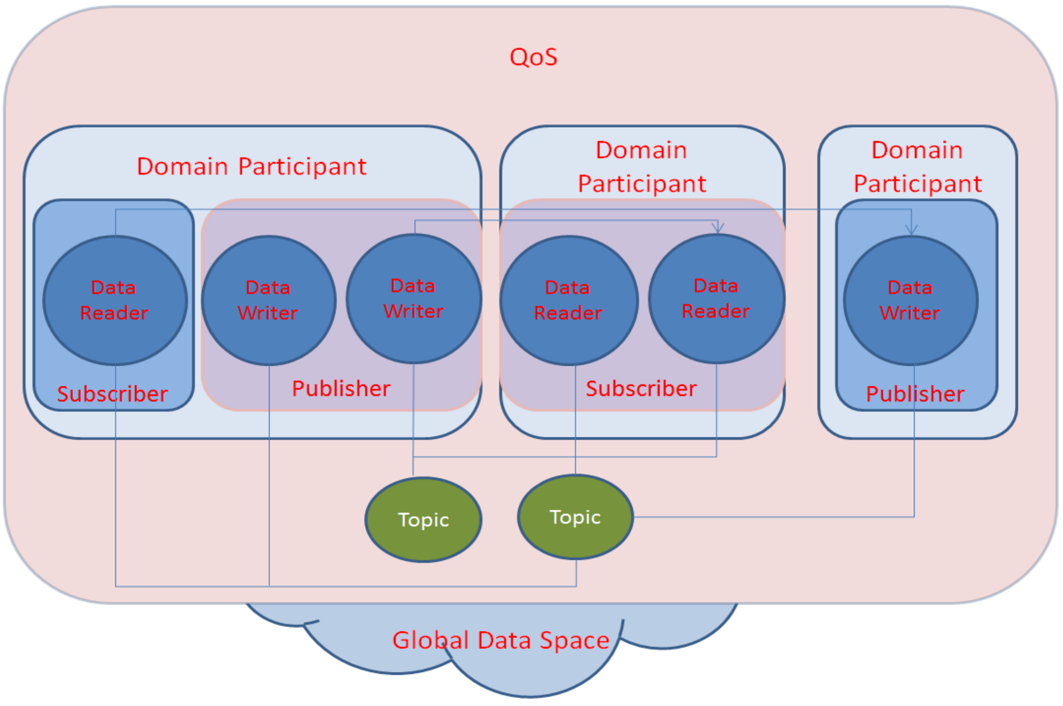

Figure 6. OMG DDS global view

Global Data Space (GDS) [2]: GDS is an abstraction of data space where all the data can be stored. Applications who want to publish or modify the data need to be publisher and who want to receive and use the data need to be subscriber. There are different domains in the GDS and each domain can be further divided into different partitions logically. Domains between themselves are isolated, so an entity (like topic) can only be in one domain, but can appear in several partitions. The GDS needs to be fully distributed so as to avoid single points of failure for the systems.

Domain participant [2]: A domain participant is a representation of membership of the application in the domain, while a domain is a concept that links all the applications together so that they could communicate. Domain participant is the container for all other entities and isolates applications from each other when there are multiple applications running on the same node.

Publisher and data writer [2]: A publisher is defined to be an object for data distribution which publishes data into the GDS. But publisher itself could not do that, it can only publish data using data writer. Data writer is always associated with a publisher. A publisher may contain different typed data writers to publish different data types.

Subscriber and data reader [2]: A subscriber is defined to be an object to receive data from the GDS. It could not directly access the data as well, so it has to receive data via data reader. Data reader is always associated with a subscriber. A subscriber may have many different data readers.

Topics [2]*:* A topic is defined to be the object to associate publishers and subscribers and it is usually consisted of a unique name, a data type and a set of QoS. Only when a subscriber is subscribed to the same topic that a publisher has published, the message can be written by the data writer or read by the data reader. Besides, another requirement is that the QoS of the publisher and subscriber needs to be compatible.

QoS [2]: Every entity of the DDS has a set of QoS, so the endpoints of the communication can only communicate when they have compatible QoS configuration. DDS provides a rich set of QoS policies, ranging from data availability, data delivery, data timeliness to resources and configuration.

Overall, DDS is designed to be scalable and configurable with a rich set of QoS for real-time distributed systems.

2.3.3 Key Features of DDS

As a networking middleware, DDS provides a model for sending and receiving messages, events and commands in the network. The key features [10] are presented as below:

Data-centric middleware. DDS is a data-centric middleware and a fully distributed GDS is adopted to store the data. Thus DDS could improve the communication time and avoid single point of failure.

Connectionless. DDS utilizes the real-time publish-subscribe protocol, which is connectionless. Thus there is no need to establish point-to-point connection in the network. Compared with traditional point-to-point connection, DDS requires low cost to integrate a large system.

Automatically discovery. With DDS, the nodes in the network are anonymous to each other, so they can join or leave the network at any time. Subscribers for the application can subscribe to a topic at anytime and anywhere.

Explicit model. For applications it is not necessary to know the communication mechanism of the DDS, as DDS will handle message packing, delivery, unpacking, etc.

Interoperability. DDS is a standard model with interoperability and the interoperability is achieved via the standard DDS API, the RTPS wire protocol and the QoS. The RTPS wire protocol is essential to the interoperability of DDS as it specifies the important aspects of DDS, including dynamic discovery, platform independence, etc. The standard DDS API ensures the portability between DDS middleware and the application, while the QoS provides different configurations of QoS to fulfill different requirements [14]. Over all, these three aspects define all the necessary parts of the standard and ensure the interoperable implementation of DDS. With this feature, the users can easily choose different implementation from different vendors.

Rich set of QoS policies. DDS provides a rich set of QoS, with which users can specify and control the behavior of the communication. The QoS policies can be configured to all DDS entities like publisher, subscriber, data writer, data reader, topic and so on. The DDS communication can only be established when the QoS configurations are compatible between the publisher and subscriber. In fact, the QoS policy follows the subscriber-requested, publisher-offered pattern. Table 1 shows the supported QoS policies. These QoS parameters show concerns to different aspects of DDS, including data delivery, data availability, data timeliness, configuration and resources.

Table 1. Supported QoS polices (see [2])

QoS Policy	Meaning	Concerns	RxO	Changeable
USER_DATA	User data not known by middleware, but distributed by built-in topics	DP, DR, DW	No	Yes
TOPIC_DATA		Topic		
GROUP_DATA		Publisher, Subscriber		
DURABILITY	Defines whether to keep the data samples for late-joining data reader or not.	Topic, DR, DW	Yes	No
DURABILITY_ SERVICE	Configuration of the durability service	Topic, DW	No	No
LIFESPAN	The maximum valid duration of an data instance	Topic, DW	N/A	Yes
HISTORY	Determines whether to deliver the most recent value or all the intermediate changes	Topic, DR, DW	No	No
DEADLINE	Write or receive a new sample at least once every deadline period	Topic, DR, DW	Yes	Yes
LATENCY_ BUDGET	Specifies the maximum acceptable delay that the from the data written until received by data reader	Topic, DR, DW	Yes	Yes
TRANSPORT_ PRIORITY	Hint to set the priority of the underlying transport	Topic, DW	N/A	Yes
OWNERSHIP	Decides whether and how multiple data writers are allowed to modify an	Topic, DR, DW	Yes	No

	instance.			
OWNERSHIP_ STRENGTH	Specifies the value of the strength to arbitrate which data writer to modify the instance.	DW	N/A	Yes
LIVELINENESS	Determines whether an entity is still active	Topic, DR, DW	Yes	No
PRESENTATION	Specifies how to present the change of instance to subscribers	Publisher, subscriber	Yes	No
PARTITION	Logical partition of the topics	Publisher, Subscriber	No	Yes
RELIABILITY	Specify the level of reliability	Topic, DR, DW	Yes	No
DISTINATION_ ORDER	Determines the logical order of the changes made to the instance	Topic, DR, DW	Yes	No
RESOURCE_ LIMITS	Specifies the resource available to be consumed	Topic, DR, DW	No	No
TIME_BASED_ FILTER	To specify the interest of a particular subset of the data instances	DR	N/A	Yes

These QoS parameters mainly focus on the following aspects: data availability, data delivery, data timeliness, resources and configuration.

2.3.4 Footprint of DDS Implementations

There are quite a few DDS implementations from different vendors. Among them are Real-Time Innovation (RTI), PrismTech [15] and Twin Oaks [16], 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.

DDS has been implemented with different size due to different purpose. CoreDX is a light-weighted version from Twin Oaks Computing, which is available for embedded space. It is developed directly with native Operating System interfaces using C language, which can reduce latency. Also due to its modular design, the DDS feature can be reconfigured by the user according to the requirements. Thus the library can be measured in kilobytes.

CoreDX requires only small runtime memory as it is able to run on a single Intel Pentium CPU with 640 KB memory. An application with 1 domain participant, 6 topics, 6 data writers and 4 data readers requires less than 100 KB heap memory [17]. Thus it is able to run on embedded platforms.

Connext DDS from RTI is a full implementation of DDS standard and it can be run on multithreaded operating systems like Windows, Linux, VxWorks, etc. It requires 250 MB disk and 256 MB memory [18]. Also it has a subset ConnextMicro, of which the source code is around 130 KB and 20.000 lines. Therefore it is possible for it to run on many single-chip, 32-bit microcontrollers [19].

PrismTech provides one commercial and one open source version both with full implementation of OMG DDS. The open-source version Opensplice DDS Community 5.4.1

support Windows CE 6.0 armv4i platform and VxWorks 5.5.1 Pentium target hosted on WindowsXP [20]. For this implementation, it requires at least 2 MB of shared memory.

Chapter 3

RELATED WORK

This Chapter presents related work on DDS and IEC 61499 Standard.

Schmidt and Hag first introduced the key features of OMG DDS [2] standard and then introduced the implementation of Opensplice DDS. They also introduced the use cases of Opensplice DDS in TACTICOS combat management system built on top of Opensplice DDS and it is also selected as the publish/subscribe middleware for distributing flight data plans in next generation European Flight Data Processor – CoFlight [21]. That indicates that OMG DDS is mature and has already been used in the defense system and flight control system. This thesis tries to explore the possibility of applying this middleware in industrial automation systems.

Castellote not only introduced the main aspects of the DDS model, but also described the relationship between DDS and other OMG specifications like the Notification Service and the High-Level Architecture (HLA) [22]. This thesis will focus on the performance evaluation and application DDS to IEC 61499 Standard.

Guesmi et al. presented an implementation of DDS-based middleware for real-time control systems. They provided a new scheduling strategy as a solution for data distribution. This strategy consists of an algorithm for determining scheduling parameters and a real-time scheduler based on DDS QoS policies such as deadline, transport priorities. This implementation used real-time network (CAN) as transport. Together with the EDF scheduling strategy, this implementation is adapted with soft real-time network [23]. Rekik and Hasnaoui presented a detailed implementation of DDS API on a CAN bus transport. This implementation could use CAN bus to send a number of sensor data samples in one message and the connectionless DDS property is well suited for complex distributed system [24].

Agirre et al. suggested an additional layer on top of the DDS middleware, to provide application management functionalities with QoS support for real-time distributed systems. These application managements include application deployment, maintenance, execution control, final un-installation. Besides, this layer also provides QoS reconfiguration and fault tolerance [25]. Though this additional layer has the advantage to ease the application managements, however it further raises the use complexity of the DDS middleware. Also the manually deployment of the executable code currently does not support remote software updating.

Joshi compares Java Message Service (JMS) with DDS and provides a mapping between JMS and DDS. Both JMS and DDS are publish-subscribe model based middleware with a standard API, so the user experience of both middleware are similar. However, DDS is targeted for real-time systems while JMS is targeted for enterprise systems. In addition, they use different paradigms. JMS uses autonomous message as data model while DDS uses data-object. JMS uses destination (acts as “mini-broker”) to manage the message delivery while DDS just needs endpoints matching. JMS is reliable and always tries to receive acknowledgement while DDS does not. So JMS is best for TCP based transport and DDS is based for UDP based best-effort transport. JMS and DDS have much difference, but they can be used at the same application when necessary [26]. Even though both of JMS and DDS use publish-subscribe model, they are developed for different purpose with different paradigms. In this project, DDS is used alone.

Thramboulidis et al. proposed an IEC-compliant field device model for distributed control applications. The architecture of this device model is consisted with three layers: Application Execution (AE) layer is used for deployment and execution of the application's implementation model; Industrial Process-Control Protocol (IPCP) layer and Mechanical process Interface (MPI) layer which provides the communication infrastructure for the IEC-compliant device and for application to interfacing with controlled mechanical process respectively. Besides, they also presented a reference implementation on top of Real-Time CORBA Object Request Broker [27]. However, they did modifications to the service interface function blocks. Further, Real-Time CORBA and DDS are both OMG middleware specifications, they are designed with different mechanisms for different purposes. In addition, DDS provides some features that CORBA does not, for example, the rich set of QoS provided by DDS middleware. This work will mainly focus on the DDS.

Etxeberria et al. suggested using DDS-based Ethernet communication as backbone of the industrial control network, in which most of cases fieldbuses like CAN or Profibus are used as transport. They suggested mapping industrial communication elements like alarms, events, sample values and request service into DDS QoS elements [28]. However, they did not provide any use case and evaluation of the mapping, nor did these mapping follow any international standard.

Perez et al. presented guidelines to implement SIFBs of IEC61499 Standard utilizing OPC (Object Linking and Embedding (OLE) for Process Control) standard. With SIFBs, IEC61499 application can easily use the widely used OPC industrial communication standard [29]. Although addressing different standard, their work indicates integrating SIFBs with certain communication standard could be achieved.

Calvo et al. presented guidelines to build communication SIFBs based on the OMG DDS middleware. They suggested first using IEC 61499 tool (4DIAC, for example) to define SIFB interfaces; then developing the algorithms for the services; finally compiling and deploying the SIFB to an application [30]. However, in their work, they did not present any performance evaluation and applicability of the technology.

Moreover, in the thesis a detailed mapping between the DDS [2] data model and the IEC 61499 standard [4] is given. Besides, the real-time requirements are mapped to DDS QoS attributes.

Chapter 4

MAPPING

This Chapter presents the mapping of RTPS to IEC 61499 Standard, the mapping of IEC 61499 interface to DDS topic and the mapping of IEC 61499 real-time system requirements to DDS QoS.

4.1 Mapping RTPS to IEC 61499

In order to utilize RTPS in IEC 61499 compliant applications, a mapping between IEC 61499 and the RTPS specification is proposed.

As a new standard of distributed and control systems, IEC 61499 defines a model of function block, application and system. The applications and systems are all based on the basic unit, function block. A special function block, communications SIFB, is defined to provide the service of communications. So the communication between applications can be modelled by the communication SIFBs. Figure 7 shows a typical IEC 61499 application model connected by communication SIFB.

A communication SIFB has almost the same structure as the basic function block with event and data interfaces. The algorithm of the SIFB can be implemented according to the requirements.

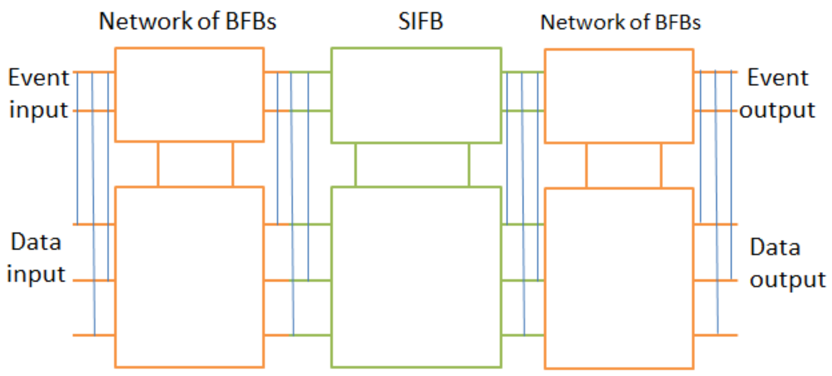

Figure 7. Typical IEC 61499 application model (BFB: Basic Function Block)

The RTPS protocol is proposed to be a connection between applications and could be used to handle communications. In the context of IEC 61499, the communication SIFB needs to be replaced or combined with the RTPS protocol. So the RTPS could be implemented within the SIFB and Figure 8 shows the applications connected by a RTPS-based SIFB. As shown in Figure 8, the applications are not directly connected since the RTPS is connectionless. Instead, publisher and subscriber applications communicate by subscriptions to the same topic in the global data space.

In this model, the communication of the events and data are achieved via creating publisher/subscriber/topic and publishing/subscribing data. Just as the normal publish-subscribe model, the publisher and subscriber need to associated to the same topic and then then they can get access to the message.

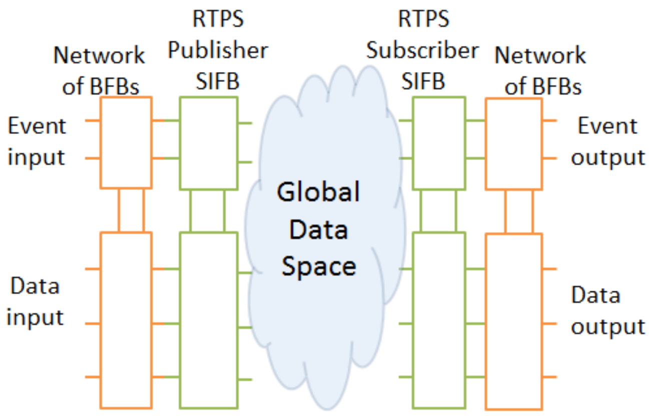

Figure 8. IEC 61499 application model with RTPS SIFBs

4.2 Mapping the IEC 61499 Interface Declaration to the DDS Topic Declaration

Since in Figure 8, a general mapping of the RTPS to IEC 61499 Standard is presented, now the focus is on how to achieve the event and data communication. The purpose of the communication in an IEC 61499 compliant system is to transfer data or events and the events are usually associated with some data. Meanwhile, the underlying of the data-centric DDS middleware is a data model, which is usually a specified data structure, topic. So a mapping between the DDS topic and the IEC 61499 interface is proposed.

The detailed mapping of the IEC 61499 interface declaration to the DDS topic declaration can be explained in an example in Figure 9. The application (networks of BFBs) has separate event and data flows, so the event output of the application is translated into a topic name while the data outputs associated with the event are translated into a topic type (data structure). This mapping ensures that the event and its associated data can be transferred at the same time.

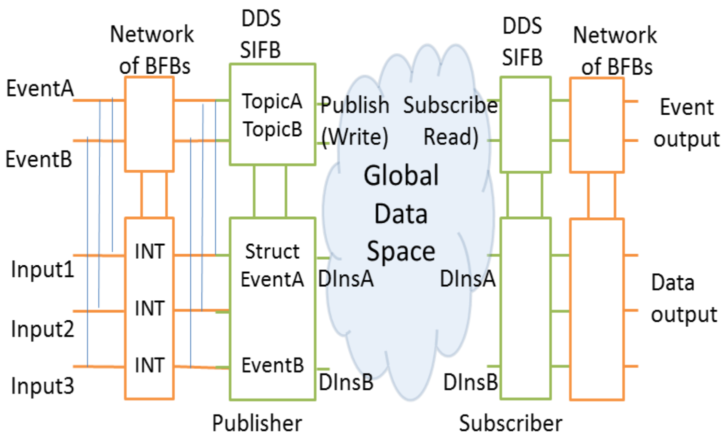

Figure 9. Mapping event and data inputs into DDS topic

Figure 9 shows an application with two event inputs EventA and EventB. Event A is associated with two data inputs Input1 and Input2 while EventB is associated with only one data input, Input3. In this case, two topic names and two topic types need to be declared. Event A and EventB are the topic names and the topic types. The data inputs, Input1 and Input2, consists a data structure as topic type. Sometimes in order to utilize the key of DDS topic, an extra data type (int tId) can be inserted to the structure. The topic types are usually defined as an OMG IDL structure.

DInsA and DinsB in Figure 9 are instances of the topic types EventA and EventB, which are the data objects that make the publisher and subscriber of the EventA associated.

The data and event inputs of the function block in Figure 9 can be structured as the model below:

```
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 two OMG IDL structures, EventA and EventB shown below, are mapped from the data and event inputs of the IEC 61499 function block. The extra field tID is defined to be used as key in case the key is defined. The pragma directives clause is used to define the key. For the OMG IDL structure Event A, the key is not defined, but the key is defined for EventB since the pragma directive specifies the keylist to be tID. The topic could be with multiple instances if the key is defined and the specific data writer can modify the specific data instance by referring to the key.

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


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

4.3 Mapping Temporal Requirements on IEC 61499 Communication to DDS QoS

Since IEC 61499 is an open standard for the next generation of distributed control and automation systems, systems compliant to this standard usually have real-time requirements. Meanwhile, DDS as a middleware based on RTPS protocol provides some real-time properties. An important feature of DDS is that it provides a rich set of QoS policies, which can be used to specify the real-time behavior of the communication. Among the rich set of QoS elements, the transport priority, deadline and latency budget QoS are particularly important to achieve real time performance.

The transport priority of DDS is a hint and the DDS specification defines the transport priority to be a 32-bit long integer. Any value within the 32 bit could be a transport priority and the policy is the higher value, the higher priority. So assume an event e of an application is transferred to another application with latency requirement- $\mathrm{Latency}^{\mathrm{e}}$. If the policy to assign priorities to the events is the lower latency requirements the higher the priority and assume the highest priority value is the maximum value of the 32-bit long integer, the transport priorities can be assigned to the events by using the following formula:

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

Here $\mathrm{MaxPrio}$ is the maximum value of the 32-bit long integer and $\mathrm{TransportPriority}$ is the priority to be assigned to the event based on the real-time requirements. Assuming that $\mathrm{Latency}^{\mathrm{e}}$ is in the unit of microsecond, then, there may be approximately 73 million different priorities. However, this is based on the absolute value of the long integer. For some of the DDS implementation vendors, they may treat the value at a specific range as just one transport priority.

Most industrial control and automation systems require that the messages should be delivered to the right place without missing the deadlines. DDS provides deadline QoS policy to monitor the temporal performance of data delivery. This policy is very useful when a data instance has to be updated periodically. When the application is assigned with a deadline, it can be notified when the deadline is missed by means of listeners or conditions.

Latency budget QoS policy defines the maximum acceptable delay of messages delivery and is a hint for the application to decide the “urgency” of the data communication to the middleware. This QoS policy helps to optimize internal operations and maximize the throughput. The default value of latency budget is zero, which indicates that the message should be delivered with a minimum latency. When the latency budget is defined by user with a specific value, then the DDS might automatically wrap several messages into one packet. This maximizes the throughput at the cost of possibly larger latency and there could be less network traffic in the network.

Chapter 5

TEST SETUP

Based on the mapping in Chapter 4, a set of tests are carried out to evaluate the performance and the applicability to the industrial automation system. This chapter first explains the main metrics to be measured and calculated in this project and then explains all the test setups.

5.1 Test Metrics

For real-time systems, the latency and jitter are considered to be the most important two performance metrics. Also in the network communications, they are essential to represent the communication efficiency and stability. So in this work, the focus is on the end-to-end latency and jitter as well.

Latency in this work is defined to be the duration since a message is written by a data writer until it is received by a data reader. Latency is mainly decided by the transmitting protocol, transmitting distance, network bandwidth and the load on the network. The implementation of the middleware might also cause some overhead during the communication.

In real-time systems, the worst case performance is a very important evaluation. However, it is very difficult to measure the worst-case. An approximated solution is to pick the worst in a large volume of samples, which is adopted in this work. The worst case latency here is defined to be the largest value measured in 1.000.000 latency samples.

Jitter is defined to be the time difference in two consecutive measurements, including the average and maximum jitter. The average jitter is calculated by average the latencies of sending 1.000.000 messages and the maximum jitter is the biggest one among them.

In the test, the distribution of the latency samples is also calculated.

5.2 Test Setup

The tests mainly focus on the performance of the DDS communication and the latency and jitter are the main metrics used to evaluate the performance. As a reference to the evaluation, the same setup is applied to a client-server based socket communication in each test. Several different scenarios are presented to evaluate the performance.

In order to model the real-time systems, all the tests are carried out on a network of work stations with real-time Linux Operating System (OS). These machines are running Ubuntu 11.10 and patched with the real-time kernel patch RT-29. After recompiling the kernel of the OS, all the nodes run over full pre-emptive Linux environments. An Ethernet switch is used to connect all these nodes to realize Ethernet communication. To simplify the evaluation, all the nodes have 1 Gbps Ethernet network adapters.

DDS is becoming more popular and there are quite a number of implementations targeted on different operating systems on the market. However, in this project, the open source version Opensplice DDS Community 5.4.1 for Linux from PrismTech is adopted. Considering the IEC 61499 compliant development tools, there are already quite a few available, but most of them lack documentations. So no IEC 61499 IDE is used; instead the function blocks are directly implemented with C++ classes. GCC compiler is used to compile the executable code.

Two different setups are used in the tests. One is to connect two nodes with an Ethernet switch, which constructs an exclusive network to measure the end-to-end performance. The other setup connects four nodes (two of them are used for generating extra network traffic) in the network to evaluate the scalability of the distributed system. In both setups, the applications are configured with different QoS policies to evaluate different aspects of the communication.

5.2.1 Network without Extra Load

The first setup utilizes two nodes and connects them with an Ethernet switch as Figure 10 shows. This network setup is an exclusive network since no other nodes generate any extra load in this network.

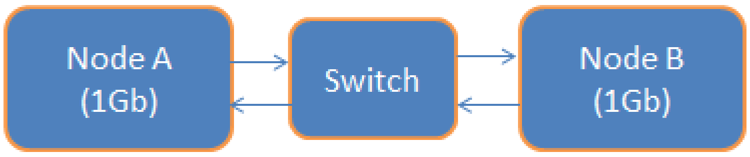

Figure 10 test with 2 nodes

Though no IEC 61499 compliant IDE is used, the implementation still tries to be compliant to IEC 61499 standard. In Chapter 4, the IEC 61499 SIFB interface is mapped to the DDS topic declaration and in Figure 9 an example shows how the mapping is achieved. So hereby the DDS based SIFB is implemented using a C++ class, named iec2dds. In the implementation, the features of function block are compliant with IEC 61499. The public fields and methods 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 simplify the DDS SIFB interfaces, all the entities that needed for DDS as internal fields are declared by the two public methods, createPublisher() and createSubscriber(). These two methods also set the QoS policies for these internal entities. CreatePublisher() method has two parameters, topicName and tQos; and it configures the QoS policies for both the publisher and data writer. CreateSubscriber() method configures the QoS policies for the subscriber and data reader. Reading and writing messages are achieved by the two methods, read() and write(). Since read() method needs to pass a message to the application, it has one pointer as parameter. The application can contain multiple publishers and subscribers.

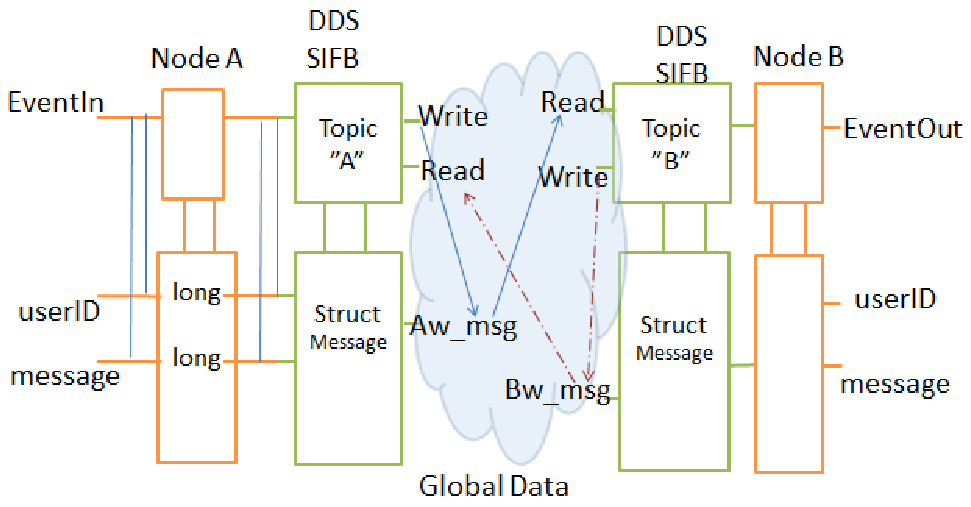

Figure 11. DDS Network communication

Figure 11 shows the detail view of the setup. In the setup, application at node A only contains one event (EventIn) which is associated with two data inputs (userID and message). So an OMG IDL structure is defined as below to represent the event and data inputs.

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

In this IDL structure, the filed message is defined as an array so that the message length can be easily changed in the test to evaluate how the performance changes when sending messages with different lengths. This IDL structure Message serves as the topic type and many topic instances can be based on this type.

Defined a topic "A", publisher at Node A and subscriber at Node B need to be associated to the same topic A and then they can communicate. Publisher at Node A writes an instance Aw_msg while subscriber at Node B reads the instance Aw_msg, of which the elapsed time of this process is just defined as the latency. The line arrows show the one-way message flow and together with the dashed arrows make up a round trip of the message. The latency in the test is not directly measured since it needs synchronization of two nodes. An easier way adopted is to measure the round trip time of the message at Node A. A time stamp preWriteTime is put before publisher at Node A writes the message and another time stamp afterReadTime after the subscriber reads the message. So the round trip time can be easily calculated by the following formula:

$$RoundTripTime = postReadTime - preWriteTime$$

In this measurement, Node B just receives and forwards the message. Though there might be overhead by the implantation of read and write methods of DDS, they are minimized.

A client-server socket communication is also established between the two nodes to act as a reference. The message structure and length used in the socket communication is the same as that of DDS.

5.2.2 Network with Extra Network Load

Figure 12 shows a network of four nodes connected by an Ethernet switch. Two more nodes (C and D) are added to the network to model extra network traffic only between them. The configurations of Node A and Node B are not changed as Figure 11 shows. This setup is intended to evaluate how the network load would affect the communication performance (measured in Node A).

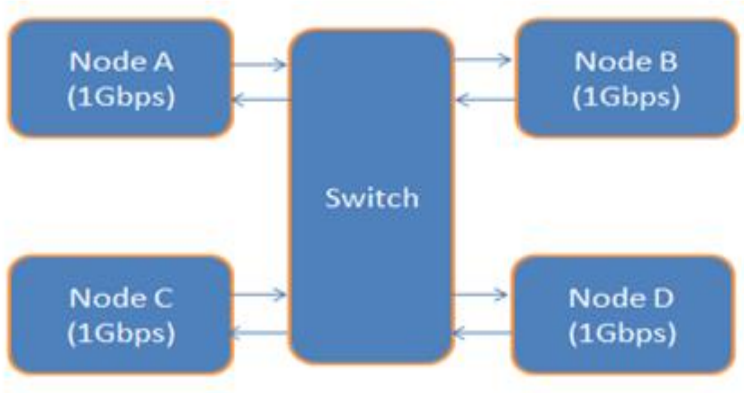

Figure 12. Network with 4 nodes

Both setups in figure 11 and figure 12 are applied to different configuration of DDS QoS. The main elements of this QoS set being looked into are reliability, transport priority and latency budget, which are the key factors of real-time communication.

Reliability: The reliability QoS is consisted with 2 elements, reliable and best-effort reliability QoS. Reliable QoS ensures that every message can be sent to the destination by certain acknowledgement and resending mechanism. Under best-effort reliability QoS configuration, the missing message will not be retransferred by the application.

Transport priority: Transport priority QoS assigns each topic a priority in which DDS can prioritize the data produced by a data writer. This Qos policy is considered to be a hint to the DDS to control the priorities of the underlying transport means. A higher value represents a higher priority and the full range of the type is supported. By default the transport priority is set to zero. The Transport Priority Qos policy is applicable to both topic and data writer entities. After enabling of the concerning Entities, this Qos policy may be changed by using the set_qos operation.

Latency budget: Latency budget Specifies the maximum acceptable additional delay to the typical transport delay from the time the data is written until the data is delivered to the data reader and the application is notified of this fact. This Qos policy provides a means for the application to indicate to the DDS the "urgency" of the data-communication. By having a non-zero duration, the DDS can optimize its internal operation. The default value of the duration is zero, indicating that the delay should be minimized.

Table 2 shows all the tests to be performed during this project. Different combinations of QoS elements and different number of nodes in the network are presented. In the table the number N represents different number of transport priority and different values of latency budget.

Table 2. Tests

Number of Nodes	Communication	QoS(DDS)		
2	DDS	Reliability	Transport priority	Latency budget
		Reliable	0	0
			N	0
			0	N
		Best-effort	0	0
	Socket	-	-	-
4	DDS	Reliability	Transport priority	Latency budget
		Reliable	0	0
	Socket	-	-	-

Chapter 6

RESULTS AND ANALYSIS

In the test, mainly the round trip times are measured to evaluate the latency. Each of the Round Trip Time (RTT) value is the average of 1.000.000 iterations. Jitter is calculated to evaluate the stability of the communication.

6.1 Results without Extra Networking Load

In this setup, the main focus is on the evaluation and analysis of the DDS communication based on different configuration of QoS.

Table 3 illustrates the average RTTs and latencies of transferring messages between the nodes A and B utilizing DDS and client-server based socket. The DDS QoS uses all default value except the reliability QoS which is configured to be reliable to ensure that all messages will eventually be delivered to the data reader. Besides, the DDS uses waitset (a DDS event handling mechanism that blocks a thread until a new message arrives) to read messages.

Table 3. Round Trip Times and latencies of reliable DDS and socket communication

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

In Chapter 5, the measurement of RTT is explained. Since the one-way latency is half the round trip time, when the RTT is measured the latency can be easily calculated by the simple formula:

$$Latency = RoundTripTime/2$$

Figure 13 plots the average latency values against the message sizes. A small circle in the figure represents one latency value of the DDS communication, while a star represents one latency value of the client-server socket communication. The lines connecting these points are just for illustration and comparison. From Figure 13, it is easy to see that the DDS communication latency is larger than the client-server based socket communication. Moreover, the DDS communication latency is more sensitive to the increase of the message size since the DDS communication latency increases sharply when message size increases from 2^{11} to 2^{14} bytes.

However, both the DDS and client-server based socket communication latencies do not change much when the message size increases from 2^3 to 2^{10} bytes. The reason for that could be each of the messages fits in one Ethernet frame (the maximum Ethernet frame is around 1500 bytes) and needs to transfer only once. Thus when the message size increases to a value larger than the maximum Ethernet frame, one message might need to be sent in several times. This will largely increase the latency.

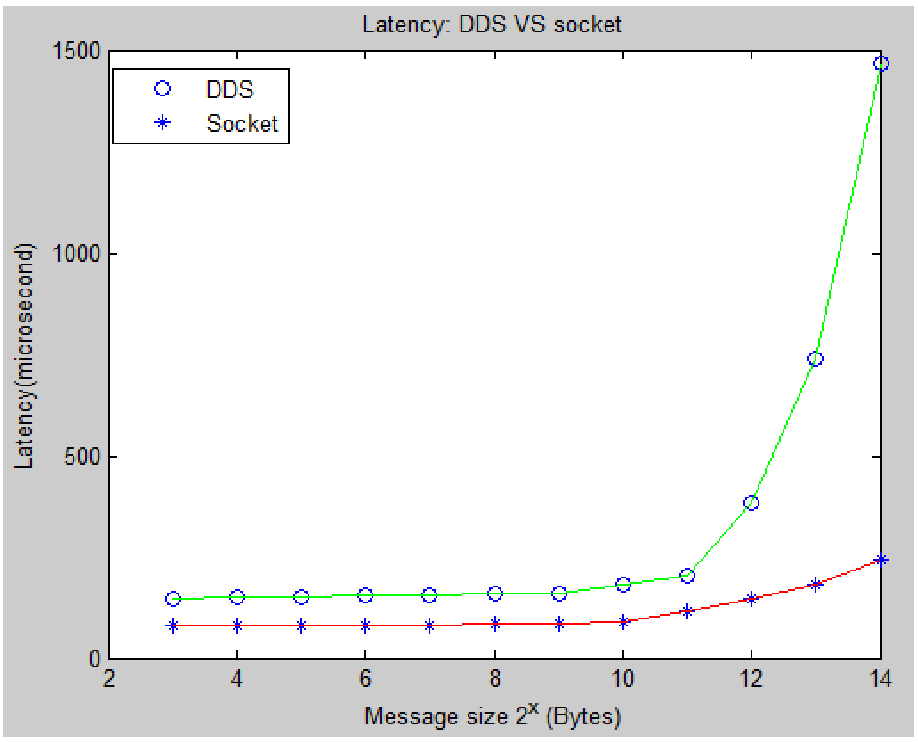

Figure 13. Latencies of reliable DDS and socket communication
(Note: lines connecting points are only used for illustration)

Table 3 shows that when sending a message of size 1024 bytes, the DDS takes 182.5 µs, while the socket communication takes 98 µs. Obviously, socket communication is much faster than the DDS as it only takes half the time the DDS takes to send a message. However, the DDS can still send 5479 messages of 1024 bytes in one second, which can meet the requirements of some industrial distributed systems that do not require extreme fast message transport.

Table 4. Jitter of reliable DDS and socket communication
sending 1024-byte message

Jitter	Minimum	Maximum	Average
DDS(µs)	0	7970	73.776
Socket(µs)	0	221	18.791

Table 4 gives the minimum, maximum and average jitter of the DDS and socket communication when messages with size of 1024 bytes are sent. The maximum jitter of the DDS communication is quite large, 7970 µs, which is 20 times more than the average RTT. But the maximum jitter of the socket communication is 221 µs, which is less than three times the RTT. So the socket communication is more stable than the DDS communication.

Table 5. Worst case latency of reliable DDS and socket communication

Message (bytes)		8	16	32	64	128	256
Worst - Case DDS (µs)	RTT	14347	13806	16720	12363	13624	16251
	Latency	7273.5	6403	8360	6181.5	6412	8125.5
Worst-case Socket (µs)	RTT	158	184	1007	162	165	193
	Latency	79	92	503.5	81	83.5	96.5
Message (bytes)		512	1024	2048	4096	8192	16384
Worst-case DDS (µs)	RTT	12453	14933	14888	17527	19928	14600
	Latency	6226.5	7466.5	7444	8763.5	9964	7300
Worst – case Socket (µs)	RTT	194	228	286	4178	2210	1546
	Latency	92	114	143	2089	1105	773

Table 5 gives the worst case latencies of DDS and socket communication. The worst case latency is defined as the maximum latency during the 1.000.000 iterations. As shown in the table, the worst-case DDS latency is much larger than the worst-case latency of socket communication. This in a large extent describes the reliability of socket communication is better than DDS.

DDS provides 3 different methods, polling, listener and waitsets to read messages. Waitsets blocks a thread until an event (in this case, message arriving) occurs. Table 6 and Figure 14 compare the waitsets and polling methods. It is shown in the table that when polling is used to receive a message the latency could be around 30 µs less than that of waitsets when the message size is at the range of 2^3 to 2^{10} bytes, which is rather fast. But polling keeps CPU checking the arrival of message and does nothing else, so it is not efficient for industrial applications.

Table 6. Latencies of reliable DDS communication utilizing polling and waitset to read message

Message (bytes)		8	16	32	64	128	256
DDS Latency (µs)	Waitsets	147	151	151	156.5	157.5	159
	Polling	108	114	115.5	117	117.5	123
Message (bytes)		512	1024	2048	4096	8192	16384
DDS Latency (µs)	Waitsets	161	182.5	205.5	383.5	737.5	1465
	Polling	128.5	141	195.5	375.5	730.5	1453

As the message size increases, the waitsets latency gradually gets close to polling and finally could work at the same latency. The reason could be that the waitsets implementation overhead becomes not so obvious when it compares to larger communication latency.

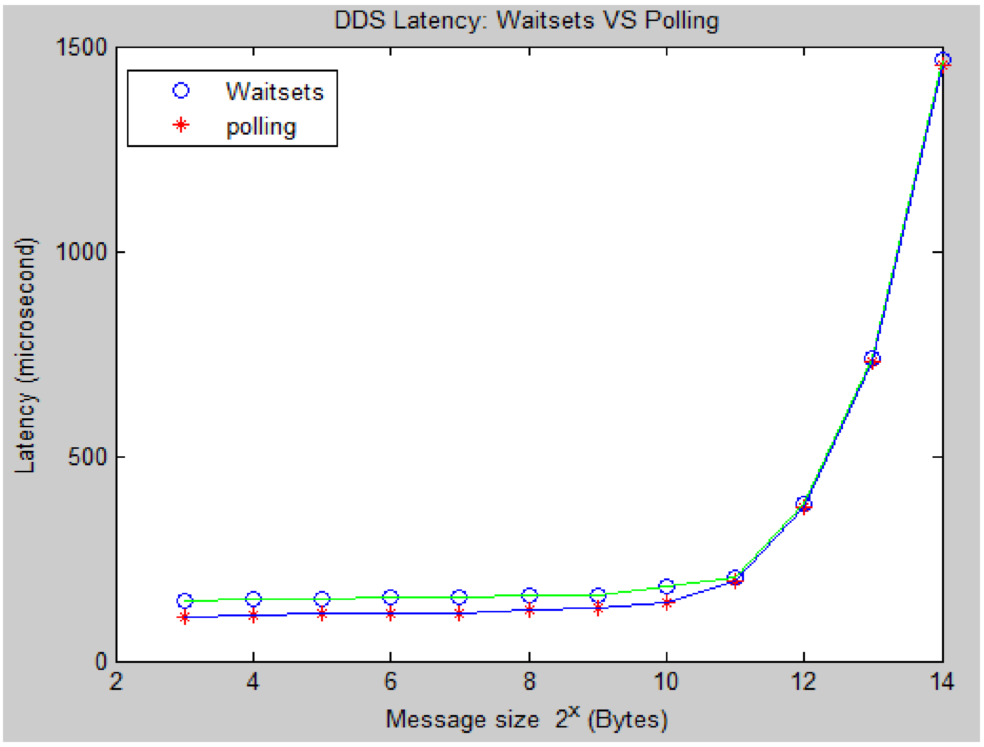

Figure 14. Latencies of reliable DDS communication utilizing polling and waitsets to read message (Note: lines connecting points are only used for illustration)

In order to evaluate how the reliability QoS would affect the DDS communication. Table 7 gives a comparison of the latencies between best-effort transport and reliable DDS communication.

Table 7. Latencies of reliable and best-effort DDS communication

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

Figure 15 plots the comparison of the average latency of the best-effort and reliable DDS communication. Though not so obviously, the figure shows that the best-effort transport is a little faster with lower latency, since the reliable mechanism could cause more overhead to the latency.

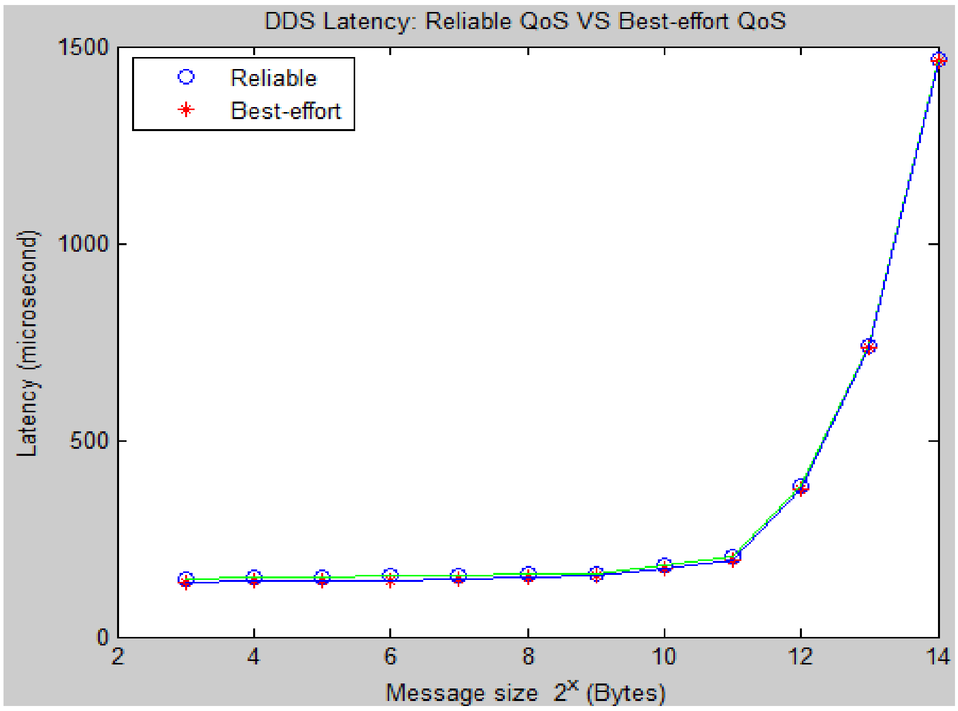

Figure 15. DDS latencies of reliable QoS and best-effort QoS communication (Note: lines connecting points are only used for illustration)

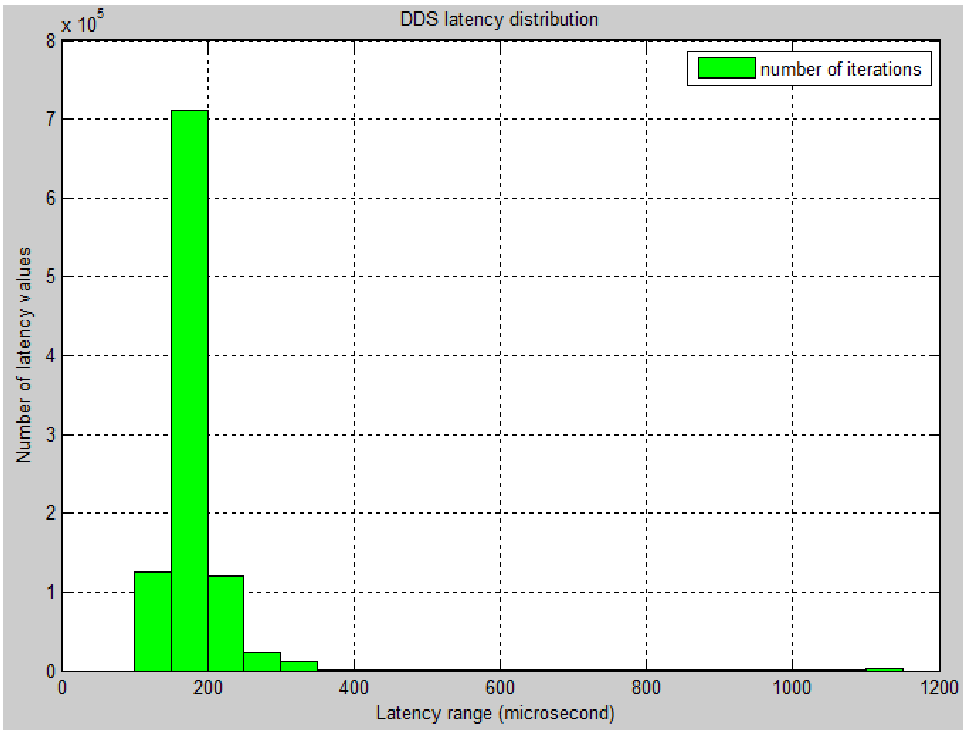

Figure 16. Distribution of the DDS latency values

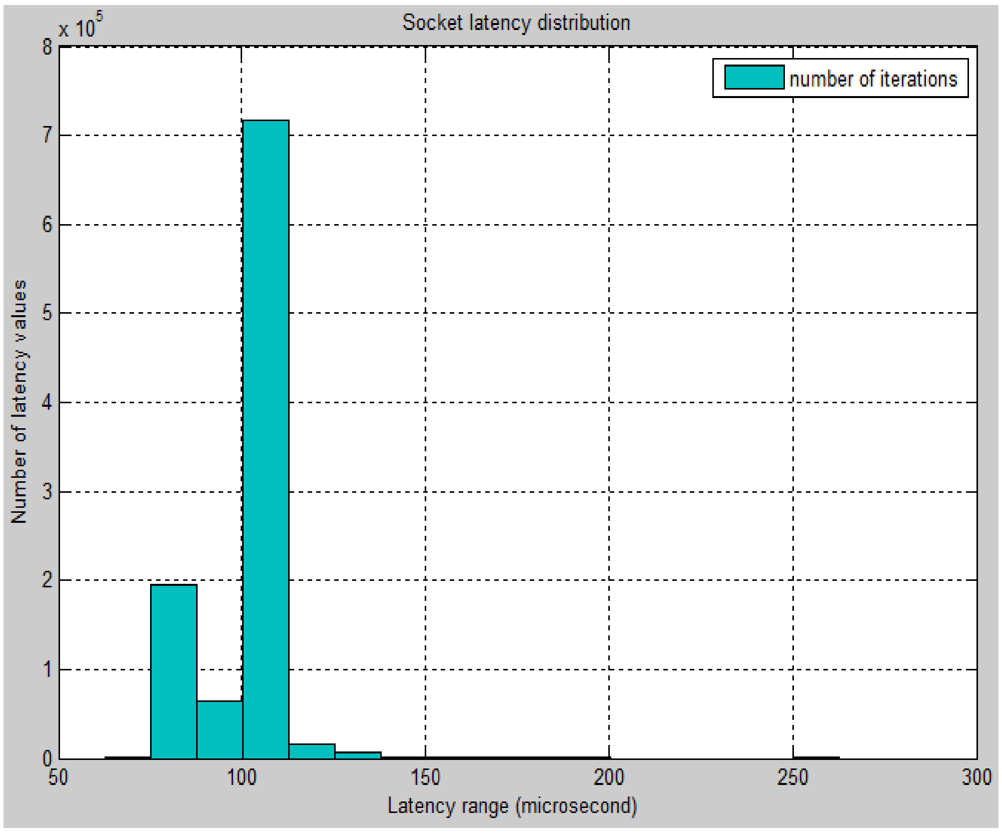

Figure 17. Socket latency values distribution

Figure 16 and Figure 17 show the distribution of the DDS latency values from 1.000.000 samples. The Socket communication latency values mostly located at the range of 62.5 to 112.5 µs and the DDS latency values are mostly in the range of 100 to 400 µs. As shown in Figure 16, the latency distribution of DDS is very close to normal distribution.

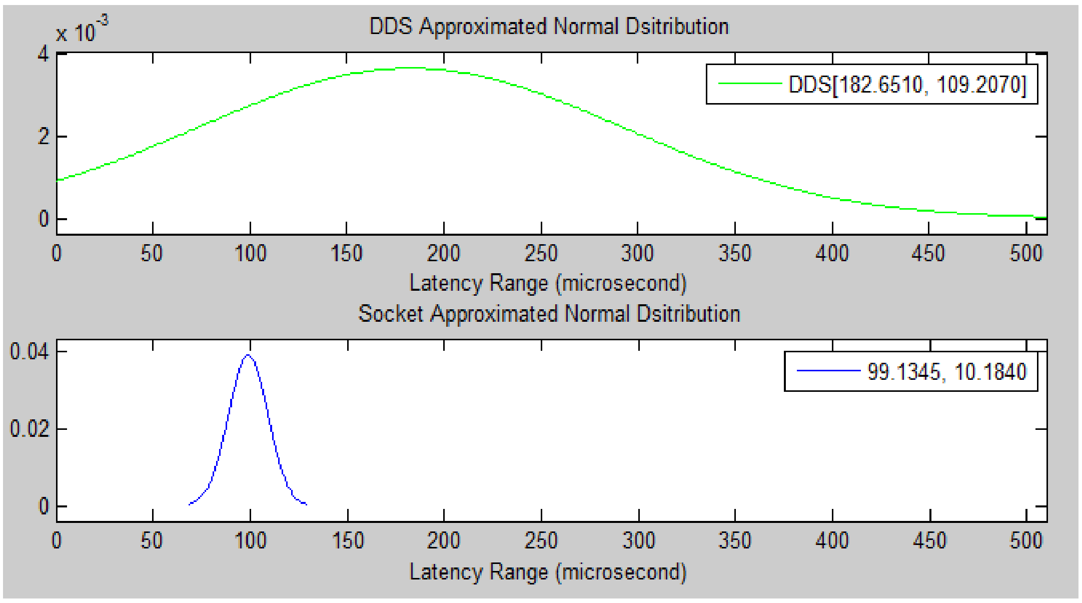

Figure 18. Curve fitting of DDS and socket latency normal distribution

Figure 18 shows the curve fitting of the DDS and socket latency normal distribution of 1.000.000 iterations transferring messages of 1024 bytes. According to the calculation, the standard deviation of the DDS latency is 109.207 μs, which takes up 59.8% of the average latency 182.6510 μs. The standard deviation of the socket latency is 10.184 μs, which takes up 10.3% of the average latency 99.135 μs. Hence considering both the value of the standard deviation and the percentage the standard deviation takes up its own latency, client-server based socket communication shows less variation of latency compared to the DDS communication.

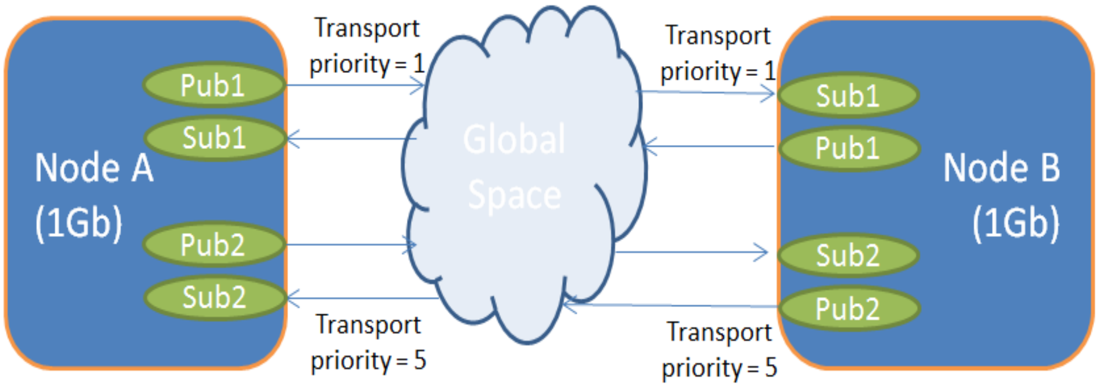

Figure 19. Publishers and Subscribers with different transport priorities

Figure 19 shows the scenario that communication happens between two nodes with topics of different transport priorities. Though the transport priority QoS is a hint to the transport, it still tries to follow the rule that the higher priority message could preempt the lower priority message.

Table 8 shows the latency results when two topics are assigned with the same or different transport priorities. At first, both topics are assigned to the same transport priority 1 and the communication latencies are 520 μs and 519 μs respectively, almost the same. Later Topic2 is assigned a priority 5 while that of the Topic1 remains 1. The latencies of Topic1 and Topic2 are 534 μs and 500 μs respectively, so the Topic2 has lower latency. So the transport priority QoS could help the users to better control which message should be deliver first.

Table 8. DDS Latency with different transport priorities

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

Latency budget QoS is another important DDS QoS to optimize the network throughput. Table 9 shows the difference of RTT, jitter and standard deviation when different latency budget QoS is set to the DDS communication. Again, the test focuses on sending messages with 1024 bytes data. Since the average RTT of sending a 1024-bye message is no more than 400 μs, the average latency would be less than 200 μs. Thus the deadline could be set to 200 μs. Usually the average latency should be between one-third to half of the deadline. In this test, the latency budget is set to be 100 μs, half of the deadline.

When the latency budget is set to 100 μs, the middleware will automatically optimize the throughput and possibly wrap several messages into one package. When the latency budget is set to zero, the middleware should send the message with minimum delay. So it is shown in the table that when the latency budget is set to zero, the average round trip time, average jitter and standard deviation of the RTT are all smaller than those when the latency budget is set to 100 µs. This QoS policy provides a trade-off between latency and throughput of the network communication, so the users could decide which aspect is more important according to the system requirements.

Table 9. Round trip time, jitter, standard deviation of reliable DDS communication with latency budget 0 and 100

Message (1024bytes)	RTT (µs)		Jitter (µs)		Standard deviation (of RTT)
	Average	Worstcase	Average	Maximum	
DDS(latency budget=0)	353	11719	58.047	11722	166.836
DDS(latency budget=100)	365	11983	76.152	11531	202.396

6.2 **Results with Networking Load**

Below are the results of the DDS and socket communication with networking load. In this experiment, two additional nodes are introduced into the network to generate extra networking traffic. The purpose of this setup is to simulate and evaluate the performance of DDS in real application with extra network traffic.

The two nodes used for generating networking load also have 1 Gbps Ethernet and they are intended to generate as much traffic as possible. In order to monitor the network traffic under Linux, the software IPTraf is used in Node C to get the statistics of extra traffic in the network. Applications on Node C and Node D are associated with a topic and keep writing and reading messages to each other. According to the IPTraf statistics, Node C and Node D generate 96.902 Mbps extra load when DDS communication is used. For reference, this setup is also applied to socket communication, Node C and D use client-server style to generate the network load and, according to the statistic, the generated network load is 94.242 Mbps. So the extra loads are almost the same over the network and then it is fair to compare the performance of the two situations.

Table 10. Round trip time, jitter, standard deviation of reliable DDS and socket communication with and without network load

Message (1024bytes)	RTT (µs)		Jitter (µs)		Standard deviation (of RTT)
	Average	Worstcase	Average	Maximum	
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

As shown in Table 10, the round trip time of DDS without networking load is even larger than DDS communication with networking load. That actually indicates that the DDS is of less variance when there is heavy load on the network. Inserting nodes into the network, the performance of the whole network will not be degraded. In contrast, with socket communication, the RTT becomes smaller when there is extra network traffic compares to the RTT when there is no extra traffic in the network. That indicates if there are more loads and traffic in the network, the socket communication performance is likely to be degraded.

The jitter of DDS communication in both situation remains almost the same, which means the DDS communication is relatively stable. For socket communication, when there is more traffic in the network, the jitter turns out to be larger. So the client-server socket based communication is significantly affected by the increased network load.

Overall, socket based communication has better performance than DDS does. However, the evaluation indicates that in the heavily loaded system, DDS turns out to be with less performance variance.

6.3 Integration Complexity

In the normal communications, it is necessary to establish a point-to-point connection between applications or subsystems. Suppose a network with n nodes, it needs $\boldsymbol{n} * (\boldsymbol{n} - \mathbf{1})$ connections to make sure each node can directly communicate with all the other nodes. This is very costly. In the test, for example, socket needs to be established between nodes so that they can communicate. To establish this connection, it is necessary to offer the IP address, port, etc. It could be very complicated when the system is consisted with thousands of applications.

As a middleware, DDS is a normalized model and all the applications or subsystems interact with it. Applications do not directly communicate with each other. In this case, any application only needs to be associated to the middleware once and make the complexity increase in a linear way [31]. For example, in the test, applications are not directly connected to each other, but they communicate through subscription to a topic in the distributed domain.

So DDS is a good solution to reduce the complexity of the point-to-point connections in large-scale distributed systems.

Chapter 7

CONCLUSIONS

The RTPS protocol provides appealing properties that could be used to realize the communication in real-time distributed control systems. Its connectionless feature makes it possible to reduce the complexity of the point-to-point connections between the software components in large-scale distributed control systems. DDS, as a specification of real-time publish-subscribe middleware, has been implemented and used in both commercial and administrative field.

This thesis first shows how the standard for next generation of distributed control and automation systems, IEC 61499, can be mapped to the DDS model. It maps the event and data interfaces of the IEC 61499 to the DDS topic declaration. Moreover, a mapping of the real-time requirements to the DDS QoS is presented.

Later, a performance evaluation of this mapping is presented and it shows how these DDS QoS policies would affect the performance of the distributed systems. A reference of point-to-point socket based connection is also evaluated to compare with the performance of DDS.

Finally, the thesis discusses why the DDS has the potential to reduce the complexity of the connections in large distributed control systems.

Chapter 8

FUTURE WORK

Although DDS has already been used in both commercial and administrative fields, there is still a lack of formal analytical model. In the future work, it could be a very interesting research direction to build an analytical model for DDS based on the QoS.

In the thesis, a comparison of DDS and socket communication has been presented. DDS has good features to reduce the complexity of the connections. To make it more practical and obvious, DDS will be used in an industrial distributed control system to check how it would perform in an industrial application.

As the low level transports in industrial distributed systems are usually fieldbus, like CAN or Profibus. So it could be very promising if the RTPS can be used to engineer the whole industrial network including both the low level transport and higher level possibly Ethernet based nodes. In addition, a performance evaluation based on such system would worth the efforts.

Chapter 9

REFERENCES

1 Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui and Anne-Marie Kermarrec, “The Many Faces of Publish/Subscribe”, *ACM Computing Surveys (CSUR),* Volume 35, Issue 2, June 2003, pp. 114 – 131

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

3 International Electro-technical Commission (IEC), *International standard IEC 61431-1*, second edition, IEC, September, 2003

4 International Electro-technical Commission, *International standard IEC 61499-1*, first edition, IEC, 2005

5 HOLOBLOC Inc., FBDK-the Function Block Development Kit, web page, http://www.holobloc.com/doc/fbdk/

6 Fbench Project - Open Tool for IEC 61499 Function Block Engineering, webpage, http://www.ece.auckland.ac.nz/~vyatkin/fbench/

7 4DIAC – Framework for Distributed Industrial Automation and Control, webpage, http://www.fordiac.org/

8 Oliver Vogel, Ingo Arnold et al*, Software Architecture: A Comprehensive Framework and Guide for Practitioners*, Springer, 2011, ISBN-13: 978-3642197352

9 Mary Shaw and David Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall Engineering/Science/Mathematics (April 12, 1996) ISBN 0131829572

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

11 Catalog Of OMG Data Distribution Service (DDS) Specifications, webpage, http://www.omg.org/technology/documents/dds_spec_catalog.htm

12 Real Time Innovation, webpage, http://www.rti.com/

13 Thales Group, webpage, http://www.thalesgroup.com/naval/

14 Twinoaks Computing Inc., *Interoperable DDS Strategies,* December, 2011.

15 PrismTech, webpage, http://www.prismtech.com/

16 Twinoaks Computing Inc., webpage, http://www.twinoakscomputing.com/

17 Twinoaks Computing Inc., CoreDX DDS Performance: Memory and CPU, webpage, http://www.twinoakscomputing.com/coredx/performance_resources

18 RTI, *Connext Core Libraries and Utilities Release Notes*, Version 4.5

19 RTI Connext Micro, webpage, http://www.rti.com/products/micro/index.html

20 PrismTech, *OpenSplice DDS Community V5.4.1 Release Notes*

21 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.

22 Gerardo Pardo-Castellote, "OMG Data Distribution Service: Architectural Overview", *Military Communications Conference*, October 13-16, 2003, Vol.1, pp. 242-247.

23 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.

24 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.

25 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.

26 Rajive Joshi, *A Comparison and Mapping of Data Distribution Service (DDS) and Java Message Service (JMS), Real-Time Innovations*, Inc. 2006.

27 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.

28 Ismael Etxeberria-Agiriano, Isidro Calvo, Federico Pérez and Oier García de Albeniz, "Mapping Different Communication Traffic over DDS in Industrial Environments", *Information Systems and Technologies (CISTI)*, 2011 6th: 1-6

29 F. Perez, D. Orive, M. Marcos, E. Estévez, G. Morán and I. Calvo, "Access to Process Data with OPC-DA using IEC61499 Service Interface Function Blocks", *IEEE Conference on Emerging Technologies & Factory Automation (ETFA),* September 22-25. 2009: 1-4.

30 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.

31 Object Management Group (OMG), *Data Distribution Service (DDS) Brief*, 2011.